Security Architecture and Specification Framework for Safe and Secure Industrial Automation

Sergey Tverdyshev[1], Holger Blasum[1], Ekaterina Rudina[2(✉)], Dmitry Kulagin[2], Pavel Dyakin[2], and Stanislav Moiseev[2]

[1] Sysgo AG, Am Pfaffenstein 14, 55270 Klein-winternheim, Germany
{sergey.tverdyshev,holger.blasum}@sysgo.com
[2] Kaspersky Lab, Leningradskoe Shosse 39A/3, Moscow 125212, Russia
{ekaterina.rudina,dmitry.kulagin,pavel.dyakin,
stanislav.moiseev}@kaspersky.com

Abstract. Today policy specification and enforcement mechanisms are often interwoven with the industrial control processes on which the security policy is enforced. This leads to interferences and non-secure behaviour as well as increases system attack surface. This paper presents a security system architecture and a framework where the processes, policies, and enforcement are strictly separated. The security architecture follows separation and least-privilege principles. The policy framework is based on a formal language and tools to specify and generate components for the security architecture. We illustrate our approach on an technological process and present how this solution is implemented in practice where security is mixed with safety requirements such as real-time, worst case execution time and certification.

Keywords: Security policy · Linear Temporal Logic · Industrial control system · Separation kernel

1 Introduction

Weaknesses of modern industrial control systems are caused by multiple reasons. Networked systems with historically grown architectures, made up from heterogeneous devices are very difficult to maintain, support and update. Usage rules and policies sometimes do not provide the necessary level of security, because the system has to operate (i.e. functional safety) and provide technological management interface.Weakly controlled access to the control interface of the critical-purpose system may have dire consequences.

In all use-cases security of the technological process shall be provided. This security should be gained not only by restricting access to the system. The human factor can thwart all measures of controlling the remote or physical access to the system (e.g. angry net-admin), particularly if the system should have emergency personnel access.

E. Rome et al. (Eds.): CRITIS 2015, LNCS 9578, pp. 3–14, 2016.
DOI: 10.1007/978-3-319-33331-1_1

Even for all seemingly authorized accesses, operational control must comply with the policy that will keep the technological process within its safe execution boundaries, e.g. building up or reducing pressure or temperature, or a process with big inertial or sensitive thresholds.

There are plenty examples where technological process got out of safe boundaries. On August of 2010 at Millard Refrigerated Services, in Alabama, U.S. the hydraulic shock caused a roof-mounted 12-inch suction pipe to catastrophically fail. This led to the release of more than 32,000 pounds of anhydrous ammonia [1]. There were more than 150 exposed victims, 32 of which required hospitalization, and 4 were placed in intensive care. Except of the failure of roof-mounted piping it also caused an evaporator coil inside the facility to rupture.

The hydraulic shock occurred during the restart of the plant's ammonia refrigeration system following a 7-hour power outage. While this incident was not a cyberattack, it is disturbing that the control program didn't prevent the dangerous attempt to restart and diagnose the system that caused the momentary pressure rise in the pipes. In case of the unauthorized access to the system the catastrophe may repeat.

As described in recent report of German government's Bundesamt für Sicherheit in der Informationstechnik [2] there was an incident where a malicious actor infiltrated a steel facility. To gain access to the corporate network the attacker used the spear phishing email. Then he moved into the plant network (it isn't reported how, but he probably traversed through trusted zones and connections between the corporate and plant network). The adversary showed knowledge of industrial control systems (ICS) and was able to cause multiple components of the system to fail. This specifically impacted critical process components to become unregulated, which has resulted in massive physical damage. This is one more case where the security control of the process operations would help to mitigate the harmful consequences of unauthorized access to the industrial system network.

The incidents with gathering unauthorized access to Internet-exposed SCADA systems are quite common now. But the process can be impacted also by insiders such as in Maroochy in 2000 [3] or due to complicated malware attack as was demonstrated by Stuxnet in 2010 [4].

2 Related Work

Common ICS security approaches are often inherited from the real-world constraints. They are pretty close to the principles of the role-based access control except that the access control on its own is unable to guarantee the proper execution of the process. Role-based security principles are unable to take into account the constraints on operations sequence and time-based issues. Such constraints may be very important in industrial control systems.

Mossakowski et al. [5] concluded that classical role-based access control (RBAC) provides the separation of duties principle but it doesn't factor the sequence of states in making a security decision. They proposed a security model extended by temporal logic to specify the execution sequences. However, they don't describe any implementation of the extended security models.

A similar approach is proposed by Mondal et al. [6]. They described a formal technique to perform security analysis on the Generalized Temporal RBAC (GTRBAC) model [7–9]. This model can be used to express a range of temporal constraints on different RBAC components like role, user and permission. In GTRBAC time is represented by a periodic expression [10]. To validate the GTRBAC model authors map it to the state transition system built using timed automata. Each of the constraints of GTRBAC model is also mapped to the timed automaton. As shown in the paper all the features of GTRBAC model can be represented using timed automata.

An interesting approach to expressing information flow policies for imperative programs is demonstrated by Balliu et al. [11]. Authors connect temporal epistemic logic and several security conditions studied in the area of language-based security. They claim that temporal epistemic logic appears to be a well suited logical framework to express and study information flow policies.

The state-based security conditions related to noninterference for confidentiality (absence of "bad" information flows) and declassification (intended release of information) are addressed in the paper. The considered attack in this approach is basically reduced to observing the system activities and deducing the information about process execution. This approach may be quite useful for monitoring the safety-critical processes but the security objectives addressed in this work are not really applicable to industrial automation because in ICS security the confidentiality of process execution is not an issue and often can be found in public sources (e.g. in form of Process Hazard Analysis [12]). The only considered impact of an attack is the direct influence on technological process, whether it was intentional or not.

Today policy specification and enforcement mechanisms are often interwoven with the process on which the policy is to be enforced [13,14]. That makes it hard to separate policy from enforcement objectives and lead to non-secure behaviour.

Flux Advanced Security Kernel (FLASK) [15] implements the idea of separation of access computation and decision enforcement. This implementation is based on two key concepts: a security server, that contains policy implementations, and object managers which are responsible for querying the security server and enforcing access decisions. Such a separation is an important step towards more flexible and reusable policies. We employ and enhance this idea by offering a number of architectural and language concepts, which, applied together, form a flexible security specification and enforcement system.

The principle of separation between the resource and application layers with minimal trusted base for the enforcement of security policies has been studied since 80s [21]. This principles are well known as Multiple Independent Levels of Security (MILS) architecture. The recently published technical report in the EURO-MILS project [20] defines a template to specify and design a MILS system with a precise operation of concerns and use-cases. The proposed security architecture follows the ideas summarised in the template.

3 Main Contribution

The objective of this research is to provide a solution to keep the execution of technological process in industrial control system safe and secure even in case of malicious activity. In this paper we

- propose a generic comprehensive security architecture with separated application execution, policy computation, and policy enforcement
- define an adaptive security framework to specify security policies
- present an implementation of the security architecture and the framework which preserves safety, real-time execution, reliability, components and resources availability etc.

The cornerstone of the solution is a security system architecture and a framework which are designed to provide support for diverse security policies. Flexibility is an essential requirement because there cannot be a single definition of security, e.g. different deployments even with one site. capability-based systems), but in practice, it is insufficient to rely on a single security policy or a fixed list of policies. We achieve this flexibility by separating a security-related logic from applications implementing the business logic/technological process. This approach has a number of advantages. From application point of view

- there is no need for applications to implement security policies
- there is no need to change applications if the security policy changes
- security policy is not limited to the means supported by applications.

We also strengthen the security part (i.e. the security of the security mechanics themselves) because

- policies are abstracted away from applications
- policies operate over abstract domains
- policies are not aware of differences between applications, resources, etc.
- policy may remain stable even if applications change significantly
- system-wide security policy is a composition of smaller policies.

The paper is structured as follows. In Sect. 4 we propose the architecture that allows different objects and subjects interact and coexist in a secure way. In Sect. 5 we define a framework which enables the system designer specify his system for both safety and security and instantiate the security architecture with his configuration. We illustrate usage of our framework on a simple technological process in Sect. 6. Finally, in Sect. 7 we show how the security architecture and the framework are implemented, i.e. how a secure industrial control system is created.

4 Security System Architecture

The architecture consists of three separated layers (cf. Fig. 1).

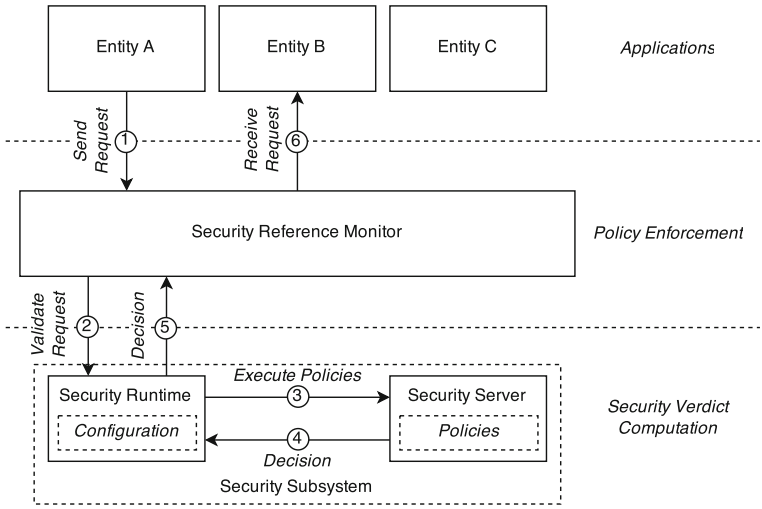

Fig. 1. Security system architecture

- Application layer: execution environments
- Security Reference Monitor (SRM): enforcement of the security verdict
- Security Subsystem: computation of a security verdict.

Application layer consists of separated execution environments, where technology specific applications are executed, e.g. interaction between processes, external communications, local computations, process control, human-machine interfaces etc. We employ the term *entity* to describe any part of application such as separated technological processes, domain, or subject whose behaviour can be described in terms of a security policy. Entities interact with each other by sending and receiving messages.

The layer for the security subsystem computes a verdict based on system state, input data, and configured security policy.

Each entity on its execution is associated with *a security context*. A security context is a data structure, which is used by stateful polices to keep security related attributes required to compute a decision. In fact, a policy knows nothing about the entity, except for its security context.

The security subsystem layer consists of two separated components a *security server* and a *security runtime*. The security runtime for each message, intercepted by SRM:

- builds the set of policies which have to be applied to this particular interaction according to security configuration
- requests the security server to compute those policies
- combines the results of the computation into the final access decision
- communicates the decision to the enforcement layer.

Security server is a component which contains implementations of all policies, manages security contexts and serves requests on policy calculation from security runtime. Security server provides interfaces to plug in external verdict engines for specific security requirements (e.g. file system with specific policy for files accesses).

The layer for the Security Reference Monitor (SRM) enforces the computed verdict on technological processes and the system as whole. The SRM controls the execution of entities, mediates interactions between entities, and communicates with the security runtime to receive and enforce the verdicts.

The verdict can be from simple "don't transfer this data", "don't allow open the file" to more complex such as "restart/pause/activate entities".

In short, the SRM can be viewed as the infrastructural level which enables entities to work, and sets up communication channels to let the information flow. SRM also guarantees that the channels do not leak, are correctly interconnected, and the channels endpoints are opened when it's allowed. The security subsystem is responsible to control content of the pipe with respect to the configured security policy. Thus, our architecture allows an easy separation of functionality, security policies, and security enforcement.

5 Framework for Security Policy Definition

The framework consists of a set of policies templates for the security server, interface definition language (IDL), security specification language (which is called CFG), and toolchain to translate specification into executable programs. Figure 2 illustrates how the framework works with the security architecture.

Security Server Templates. The framework provides a number of policies implementing specific access control approaches, e.g. type enforcement, time logic-based safety properties specifications, multilevel security. The security server can be easily extended with new policies.

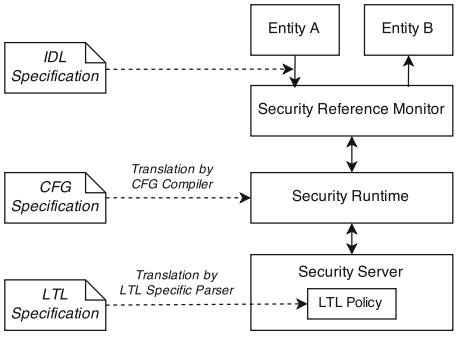

Fig. 2. Framework role in the security architecture

Each policy may have its own, policy specific configuration, and thus, many policies operate domain specific notions (for example, multilevel security models operate by labels, sensitivities and categories, type enforcement operates by domains, types etc.). These notations require support for policy-specific configurations. The framework allows a policy developer to define such policy-specific configurations as well as to integrate a parser implementation for these configurations. In Sect. 6.2 we show how this approach to custom policy

configuration works with a policy, which verifies a safety property specified as a temporal logic statement.

Interface Definition Language (IDL) is used to statically declare messages structure of every interaction between applications/entities in the system. Declared interfaces serves as anchor points in security specification to associate interactions and policies to validate them.

To configure system-wide security policy, the framework provides **specification language CFG**. It allows to specify for every interaction in the system, which policies should be applied. It also allows to specify custom policy configuration if a policy requires such a configuration. CFG is a declarative, suitable for static analysis language.

CFG compiler translates specification into C-code of security runtime component. Thus, the security runtime is a component, directly obtained from CFG specification.

6 Framework Application on a Typical Industrial Control System

6.1 Industrial Process Specification

In this section we introduce a simple processing unit which we use as the running example in this paper. This unit consists of the conveyor transferring the detail and the drill which makes a hole in this detail in the given location when the detail is under the drill.

The interesting part of the technological process consists of moving the belt to position the workpiece under the drill, running the drill, descending the drill (i.e. drilling itself), elevating the drill, and stopping the drill.

In this example, the system can be viewed as consisted of two communicating parties: entity, sending control commands (SCADA), and entity, responsible for command implementation and sending sensor information (Factory).

Control interface (IFactory) and sensor notification interface are defined using IDL:

```
package Factory
interface IFactory {    /* Factory control interface */
    BeltOn();           /* turn conveyor belt on  */
    BeltOff();          /* turn conveyor belt off */
    DrillOn();          /* turn drill on  */
    DrillOff();         /* turn drill off */
    DrillPlateDown();   /* let drill plate down */
    DrillPlateRaise();  /* raise drill plate */
    DrillPlateOn();    /* turn drill plate on */
    DrillPlateOff();    /* turn drill plate off */

}
interface IStatus {     /* Factory status interface */
```

```
    BeltOn();           /* conveyor belt was turned on  */
    BeltOff();          /* conveyor belt was turned off */
    DrillPlateRaised(); /* drill plate is at the highest position */
    DrillPlateDown();   /* drill plate is at the lowest position  */
}
```

Our purpose is to identify and specify safety properties for this small part of the process, thus, we are looking for conditions which are necessary to avoid breaking the drill or the detail. The identified constraints for this process are defined as follows:

- drilling is not allowed if the belt is currently moving on
- the conveyor can't be started while drilling
- the drill can't be descended if it is not run
- drilling can't be stopped if drill is not elevated.

Some conditions may depend on input provided by installed sensors. For example, drilling can't be run if the detail is not positioned properly, or the drill must be run at specific RPM.

6.2 Formalizing Safety Properties with Linear Temporal Logic

Linear Temporal Logic (LTL) is a useful tool in formal specification and runtime verification of temporal safety properties. LTL formulae define a set of event traces where each even has an index and an identifier of its type (such as observed command or action). In the running example, informal drilling safety properties for drilling (see Sect. 6.1) are expressed in LTL as a custom policy for security server and placed in a file "drill.tl":

```
C:DrillOn      ==> !S:BeltOn SINCE S:BeltOff
C:BeltOn       ==> !(!C:DrillOff SINCE C:DrillOn)
C:DrillPlateOn ==> !C:DrillOff SINCE C:DrillOn
C:DrillOff     ==> !C:DrillPlateOn SINCE S:DrillPlateRaised
```

where'|','!' and'==>' mean OR, NOT and IMPLY respectively.

6.3 Security Specification with CFG Language

So far we have two interacting entities SCADA and Factory, which use interfaces IFactory and IStatus to communicate. We have also formalized safety properties of the interesting part of the process encoded with LTL notation and stored in file "drill.tl". We have all pieces in place to provide CFG specification for the system.

```
01 use init policy drill_ltl_init = SecurityServer.LTL.init;
02 use call policy drill_ltl      = SecurityServer.LTL.call "drill.tl";
03 use call policy allow          = SecurityServer.Basic.allow;
```

```
04 entity Factory {
05    execute default = drill_ltl_init;
06    /* Status message */
07    send out IStatus.BeltOn           = drill_ltl "S:BeltOn";
08    send out IStatus.BeltOff          = drill_ltl "S:BeltOff";
09    send out IStatus.DrillPlateRaised = drill_ltl "S:DrillPlateRaised";
10    send out IStatus.DrillPlateDown   = drill_ltl "S:DrillPlateDown";
11    /* Control message */
12    receive in IFactory.BeltOn          = drill_ltl "C:BeltOn";
13    receive in IFactory.BeltOff         = drill_ltl "C:BeltOff";
14    receive in IFactory.DrillOn         = drill_ltl "C:DrillOn";
15    receive in IFactory.DrillOff        = drill_ltl "C:DrillOff";
16    receive in IFactory.DrillPlateDown  = drill_ltl "C:DrillPlateDown";
17    receive in IFactory.DrillPlateRaise = drill_ltl "C:DrillPlateRaise";
18    receive in IFactory.DrillPlateOff   = drill_ltl "C:DrillPlateOff";
19 }
20 entity SCADA {
21    execute default = allow;
22    send    in IFactory.* = allow;
23    receive in IStatus.*  = allow;
24 }
```

Let's each line in more detail. Line 1 declares that init policy `SecurityServer.LTL.init` is used in this specification under name `drill_ltl_init`. *Init policy* is a special kind of policy which is typically applied on entity execution and it's main purpose is to initialize the security context for newly created subject. Line 2 declares that call policy `SecurityServer.LTL.call` is used in this specification under name `drill_ltl`. The generic policy is parameterized with particular LTL statements from file `"drill.tl"`. *Call policy* is used to control interactions. Line 3 declares usage of a very basic call policy, which just permits any interaction if applied. Lines 4–19 describe security specification for entity Factory. It has three parts: execute section, sent messages control and received messages control. Execute section at line 5 specifies that on Factory-entity creation newly created security context has to be initialized with policy `drill_ltl_init`. Lines 7–10 specify what type of event should be fed into LTL policy when Factory sends particular sensor notification. For example, when Factory sends notification `BeltOn`, the policy emits an event of type `"S:BeltOn"`. Similarly, lines 11–18 specify what type of event should be fed into LTL policy when Factory accepts a command. But here, if a command violates safety properties stated in `"drill.tl"`, policy prohibits its execution. Lines 20–23 specify very permissive configuration for SCADA (all communications are allowed), since all interesting events are handled on Factory side.

As one can see, CFG allows us easily to bind formal symbols from LTL statements to particular commands and sensor notifications in a very concise way.

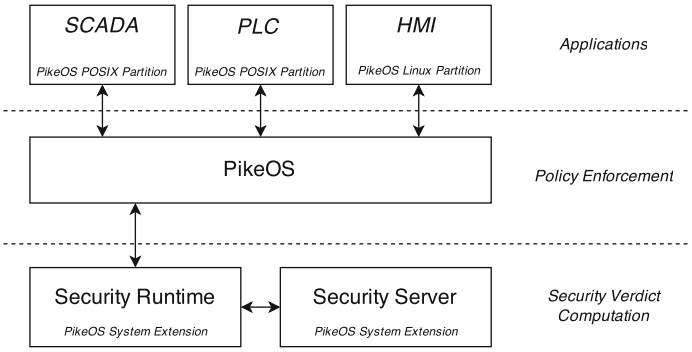

Fig. 3. Security system implementation (instantiation of Fig. 1)

7 Security System Implementation

The described solution is implemented as a security system collaboratively developed by SYSGO AG [16] and Kaspersky Lab [17]. Figure 3 depicts the implementation of the concept presented in Sect. 4.

The security system architecture is based on the virtualisation platform PikeOS [18] and covers safety and security aspects on resource and application execution layers. The PikeOS role is to

- implement the Security Reference Monitor (SRM)
- provide separated environment for entity execution
- provide separated and protected environment for execution of the security subsystem
- provide controlled communication channels between entities
- provide corset and interfaces for the security architecture defined in Sect. 4
- enforce safety policy for resource usage (e.g. memory, CPU) and guarantee real-time requirements
- provide high-assurance certification guarantees according to IEC61508 up to SIL4, IEC62443, and Common Criteria.

The implementation of the framework and security subsystem is based on a set of Kaspersky Lab technologies. The security runtime code is generated from CFG security specification for the SCADA and PLC domains as well as the domain for the operator's panel (HMI). The security runtime is a glue between SCADA, PLC, HMI and the security server and issues calls into the security server to calculate access decision with policies it contains.The verdict logic for the HMI is generated from a simple access control list.

The framework toolchain extensively uses model based specification and code generation (for both security runtime and parts of security server) to exclude human factor from the implementation of specification for the critical system.

8 Conclusions and Future Work

In this work we have presented a security system architecture for industrial control system which follows separation principles. We explicitly split between evaluation of security policies, enforcement of security verdicts, and safety critical computations. This approach allows system integrators to greatly reduce attack surface and increase security maintainability.

We have also proposed a framework accompanying the security system architecture to specify technological processes. We demonstrate the approach by employing LTL-based languages to express complex time- and event-based polices for the controlled processes.

Finally, we have implemented the security system architecture and the framework. The implementation of the security system architecture is based on the certified hypervisor PikeOS. The toolchain implementing framework is heavily utilising code generation and automatic deployment to decrease the human factor in critical parts to the minimum.

In the future we will explore and validate the developed solution on wider technological processes and adjacent domains such as railway automation, transportation and communications. We are already working on formal verification of the code generators of the security subsystem to provide ultimate correctness guarantees between the generated code and the specification. We will asses to bring formal semantics of the PikeOS as the Security Reference Monitor [19]. We will investigate how the framework's toolchain can be extended with deployment generation and interfaces to typical model-based integrated development environments.

Acknowledgement. A part of the research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) grant agreement no. 318353 (EURO-MILS, http://www.euromils.eu).

References

1. Hydraulic Shock Safety Bulletin. U.S. Chemical Safety Board, January 2015. http://www.csb.gov/assets/1/19/final_CSB_CaseStudy_Millard_0114_0543PM.pdf
2. Federal Office for Information Security. The IT security in Germany 2014 (Bundesamt für Sicherheit in der Informationstechnik. Die Lage der IT-Sicherheit in Deutschland 2014). https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2014.pdf?__blob=publicationFile
3. Abrams, M., Weiss, J.: Applied Control Solutions. Malicious Control System Cyber Security Attack Case StudyMaroochy Water Services, Australia. http://csrc.nist.gov/groups/SMA/fisma/ics/documents/Maroochy-Water-Services-Case-Study_report.pdf
4. Langner, R.: To Kill a Centrifuge. A Technical Analysis of What Stuxnet's Creators Tried to Achieve. Langner Blog (2013). http://www.langner.com/en/wp-content/uploads/2013/11/To-kill-a-centrifuge.pdf

5. Mossakowski, T., Drouineaud, M., Sohr, K.: A temporal-logic extension of role-based access control covering dynamic separation of duties. In: TIME, pp. 83–90. IEEE Computer Society (2003)
6. Mondal, S., Sural, S., Atluri, V.: Towards formal security analysis of GTRBAC using timed automata. In: Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, SACMAT 2009, pp. 33–42. ACM (2009)
7. Bertino, E., Bonatti, P.A., Ferrari, E.: TRBAC: a temporal role-based access control model. In: Proceedings of the Fifth ACM Workshop on Role-Based Access Control, Berlin, pp. 21–30. ACM, July 2000
8. Uzun, E., Atluri, V., Sural, S., Vaidya, J., Parlato, G., Ferrara, A.L., Madhusudan, P.: Analyzing temporal role based access control models. In: Proceeding of the 17th ACM Symposium on Access Control Models and Technologies (SACMAT), pp. 177–186. ACM, New York (2012)
9. Joshi, J.B.D., Bertino, E., Ghafoor, A.: Temporal hierarchies and inheritance semantics for GTRBAC. In: Proceedings of the 7th ACM Symposium on Access Control Models and Technologies, Monterey, pp. 74–83. ACM, July 2002
10. Joshi, J.B.D., Bertino, E., Latif, U., Ghafoor, A.: A generalized temporal role based access control model. IEEE Trans. Knowl. Data Eng. **17**(1), 4–23 (2005)
11. Balliu, M., Dam, M., Guernic, G.L.: Epistemic temporal logic for information flow security. CoRR abs/1208.6106 (2012)
12. Chemical Facility Security News. Control System Scenarios, April 2015. http://chemical-facility-security-news.blogspot.de/2015/04/control-system-scenarios.html
13. Banerjee, A., Naumann, D.A., Rosenberg, S.: Expressive declassification policies and modular static enforcement. In: IEEE Symposium on Security and Privacy, pp. 339–353. IEEE Computer Society (2008)
14. Rocha, B.P.S., Bandhakavi, S., den Hartog, J., Winsborough, W.H., Etalle, S.: Towards static flow-based declassification for legacy and untrusted programs. In: IEEE Symposium on Security and Privacy, pp. 93–108. IEEE Computer Society (2010)
15. Spencer, R., Smalley, S., Hibler, M., Andersen, D.: The flask security architecture: system support for diverse security policies. In: Proceedings of the Eighth USENIX Security Symposium, pp. 123–139, August 1999
16. SYSGO AG. www.sysgo.com
17. Kaspersky Lab. www.kaspersky.com
18. Brygier, J., Fuchsen, R., Blasum, H.: Safe and secure virtualization in a separation microkernel. In: Proceedings, Embedded World Conference. Nuremberg (2009)
19. Verbeek, F., Schmaltz, J., Tverdyshev, S., Blasum, H., Havle, O., Langenstein, B., Stephan, W., Wolff, B.: Formal functional specification of the pikeos separation kernel. In: Proceedings of 7th NASA Formal Methods Symposium. Pasadena (2015)
20. EURO-MILS Consortium: MILS Architecture, Technical report (2014). http://euromils.eu/downloads/2014-EURO-MILS-MILS-Architecture-white-paper.pdf
21. Rushby, J.: Design and verification of secure systems. In: Eighth ACM Symposium on Operating System Principles, pp. 12–21 (1981). http://www.sdl.sri.com/papers/sosp81/sosp81.pdf