J Ambient Intell Human Comput (2016) 7:705–719
DOI 10.1007/s12652-016-0371-6

ORIGINAL RESEARCH

Automated design, verification and testing of secure systems with embedded devices based on elicitation of expert knowledge

Vasily Desnitsky[1] · Igor Kotenko[1]

Received: 9 October 2015 / Accepted: 29 March 2016 / Published online: 9 April 2016

Abstract The rising significance and widening of embedded systems stipulate the importance of the security means against a great deal of computer security threats. Such systems involving a diversity of an-hoc embedded and mobile electronic devices functioning with the use of a broadband Internet access and even cloud technologies, are referenced conventionally as Internet of Things systems (IoT). Due to specificity of IoT systems the application of the combined security mechanisms requires their efficient energy and computing resource consumption, identification of potential conflicts and incompatibilities, control of information flows, monitoring anomalies of data in the system and other issues. At that an increased design complexity of IoT systems is determined by a low structuring and formalization of security knowledge in the field. We proposed an approach to identification of embedded security expert knowledge for its subsequent use in automated design, verification and testing tools for secure IoT systems. The paper encompasses the core elements of the proposed technique, namely security component configuring, revelation of implicit conflicts, verification of network information flows and abnormal data from sensors. The domain specific analysis of the field of embedded security is described. We also present the revealed expert knowledge used for configuration, verification and testing of embedded devices. Issues of software implementation and discussion are covered.

Keywords Embedded device design · Internet of Things · Security · Verification and testing of IoT systems · Security components · Expert knowledge

✉ Igor Kotenko
ivkote1@mail.ru; ivkote@comsec.spb.ru

Vasily Desnitsky
desnitsky@comsec.spb.ru

1 Laboratory of Computer Security Problems, St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS), 39, 14 Liniya, St. Petersburg 199178, Russia

1 Introduction

Lately contemporary computing systems are evolving towards systems of various complex embedded and mobile devices rich of network and functional connections, Internet of Things systems assuming joint and agreed functioning of lots of heterogeneous devices, sensors, servers, clouds and databases, etc. As a rule such systems are targeted at solving some highly specialized objectives for a concrete business scenario. Typical examples of such systems are the logistic and warehouse management systems, specific medical systems for incessant tracking vital parameters of the human body, systems being deployed for quick emergency response, security systems in railway transport, smart home systems and others.

The key features of embedded devices include, firstly, resource constraints imposed on energy and hardware capabilities and, secondly, large subjection of embedded devices to malicious actions of intruder on the physical level (Hwang et al. 2006). For example, a microcontroller ATmega328 commonly used for development of a variety of embedded devices is characterized by an 8-bit RISC CPU, 32 KB ISP flash memory, 2 KB SRAM, 1 KB EEPROM, distinguishing it from other types of computing systems. Such devices as smartcards, portable barcode scanners and other ones are characterized by their mobility and the ability to use multiple network protocols to connect to other devices, which determine specific security threats the device is subject to.

Restrictions on hardware capabilities of IoT devices stipulate significantly the use of existing cloud based solutions, enabling the developers to, firstly, reduce the computational load of the resource limited embedded devices, transferring some computations partly to the cloud and, secondly, increase the level of trust of the whole system, shifting some critical procedures or even services from individual devices to the secure cloud.

In contrast to general purpose devices, such as personal computers (PCs), embedded devices can also be regarded as narrow specific single purpose devices, which generally grant the only functional feature specified by the manufacturer and often presume restricted capabilities to install new software on the device. The structure and functionality as well as the security subsystem of systems with embedded devices are directly depending on the particular application domain under consideration. For example, specific medical devices (Burleson et al. 2012; Hwang et al. 2006), including particular implantable devices controlling the insulin in a human body, devices for correction of cardio arrhythmias (defibrillators), a variety of biosensors and other ones, essentially influence security requirements, threat models, power consumption, physical properties and other characteristics that substantially affect the development process and decision making processes. All this implies the need to consider specific domains of knowledge on security of IoT systems and embedded devices in the design of their security subsystems.

The most important design challenges and major concerns in the field of information systems with embedded devices include dissipation of power and other kinds of resources (Hwang et al. 2006), security of data located on the device under assumption it can be lost or stolen (Hwang et al. 2006), exposure to side-channel attacks, reduced performance of IoT devices (Potlapally 2011), user identification, arrangement of highly secure communications between devices (Ravi et al. 2004) and others. In the paper we analyze particular bits of expert knowledge in the field of embedded security for their use in development of specific design, verification and testing techniques for IoT systems as security patterns (Schumacher et al. 2006) to advance towards solving the design challenges. In the paper we search for new ones and adopt existing techniques and automated software tools for their subsequent use by developers of embedded devices.

Our main contribution here is provision of specific expert knowledge for design, verification and testing of systems with embedded devices and construction of tools to improve the security within the concept of the Internet of Things. In particular we proposed (1) a technique and a software tool for combining security components of IoT device, considering non-functional characteristics, (2) a technique and a software tool for checking information flow security policy, taking into account specific kinds of anomalies between the rules, (3) a technique for testing anomalous data from sensors, and (4) revealed a few kinds of security component conflicts, which can arise due to insufficient consistency of the system specifications.

The paper represents an expanded version of the paper accepted at the 4rd IFIP International Workshop on Security and Cognitive Informatics for Homeland Defense (SeCIHD 2014) (Desnitsky and Kotenko 2014). In contrast to (Desnitsky and Kotenko 2014) the paper includes the study on testing of information systems to reveal abnormal data from sensors, expansion of the literature review, use of additional sources of expert knowledge and advance in development of the software implementation. This paper is a logical continuation of the previously published works on design and analysis of secure systems with embedded devices (Chechulin et al. 2012; Desnitsky et al. 2012; Ruiz et al. 2012a, b) and has the following structure. In Sect. 2 the related work is surveyed. Section 3 comprises the basic elements of the proposed technique, including configuring security components, detection of hidden conflicts, verification of network information flows and abnormal data from sensors. Section 4 encompasses the domain specific analysis of the field of embedded security. It outlines a fragment of the case study used as expert knowledge sources for the proposed technique. Section 5 exposes the revealed expert knowledge used in configuration, verification and testing. Issues of software implementation and discussion are presented in Sect. 6.

2 Related work

Following (Henzinger and Sifakis 2006) we note that the development of Internet of Things systems can not be reduced solely to the models and methods traditionally used in Electrical Engineering. At that the software part of such systems is getting more and more expensive, plays more essential role and is subject to risks of information security in a wide range of critical application fields, such as enterprise management information systems, logistic systems or systems of medical implantable devices (Burleson et al. 2012).

In (Henzinger and Sifakis 2006) embedded devices in IoT systems are defined as devices, whose computing process is closely related to reaction to a physical environment of the devices. More specifically these devices run within a physical platform including modules interacting with technical objects in the environment, a variety of sensors, motors, scanners for text, audio and other information from various media display devices, a variety of communication devices, household and industrial heating and ventilation devices, monitoring and diagnostic devices, navigation pumps, navigation modules, etc. The presence of extra ties between the software part of the system, on the one hand, and its hardware and technical environment, on the other hand, determine physical constraints significantly affecting the process of such systems design.

A multi-component based approach to design systems with embedded devices got a relatively wide application (MARTE 2011) particularly within Android operating systems, platforms Arduino and Raspberry Pi. The protection system is represented as a set of interacting software and software/hardware components, each of them being in charge of particular functional security requirements. At that the process for combining security components, taking into account their peculiarities, into a single mechanism is called as configuration of security components (Desnitsky et al. 2012). The disadvantages of the approach represent presumable hidden connections and conflicts between security components arising from the absence of their a priori joint coherency.

In (Arbaugh and van Doorn 2001; Kocher et al. 2004; Rae and Wildman 2003; Henzinger and Sifakis 2006; Koopman 2004) the main issues in the field of embedded device security are presented as particular security domain problems such as user identification, local secure data storage, software resistance to modifications, secure remote access, side channel attacks protection and others. Contemporary protection means for IoT devices are targeted in the main on protection against specific security vulnerabilities. In (Abraham et al. 1991; Kommerling and Kuhn 1999; Rae and Wildman 2003; Ruiz et al. 2012a, b) various classifications of vulnerabilities, embedded device intruders are proposed, exposing intruder capabilities, competence and access type. At that combining various heterogeneous protection means within a single device, interrelations between them and issues of their integration correctness are not presented in existing works to the full extent.

The importance of the embedded device development, taking into consideration acceptable energy and computational expenses along with higher security level are uncovered in (Dick and McCallum 2004; Knezevic et al. 2009; Ravi et al. 2004). Furthermore granting necessary hardware and energy resources to the device and its services, a special issue is DoS attacks targeted on exhaustion of device energy resources (Moyers et al. 2010; Wang et al. 2012). At that this kind of attacks is not detected by wide spread antivirus solutions and aimlessly waste energy resources by the use of the most energy expensive hardware components like Wi-Fi and Bluetooth modules or screens, complicating the subsequent device work. Thus, a complex security mechanism should contain both software and hardware modules against various relevant security vulnerabilities, considering possible implicit relations and inconsistencies between concrete security components.

For embedded devices characterized by computation within specific physical constraints Henzinger and Sifakis justify the urgency of achieving trade-offs between security and non-functional characteristics (MARTE 2011) of embedded systems such as performance, applying optimization approaches (Henzinger and Sifakis 2006) and combining the embedded devices from the individual components according to their properties and requirements (Henzinger and Sifakis 2006). At that to form a specific embedded device this takes into account issues of their correct interoperation in the form of sequential and parallel component execution models, which are typical for software and hardware systems, respectively.

Configuration processes together with analysis of hardware resource limitations and time needs are of significance for end-product development (Juengst and Heinrich 1998; Wei and Qin 2009; Yu and Skovgaard 1998). At that configuring facilitates a shift from development of a mass product to a customized one adjusted to the demands of a client (Sabin and Weigel 1998).

As design case tools the specific UML profiles are applied in the industry, holding relevant embedded security peculiarities, particular requirements, vulnerabilities, security components and their properties and connections between them. In particular in (Ruiz et al. 2012a, b; SecFutur) Domain Specific Models are introduced to model and analyze security mechanisms for systems with embedded devices. In essence each domain is aimed at representation of the device in terms of a specific security feature, such as secure storage domain, secure communication domain, user authentication domain, etc. An advantage of the approach is delimitation of the design process tasks, responsibilities and roles involved as well as the use of expert knowledge in embedded security field to produce a device protection system. Software tool SPT (SecFutur Process Tool) implementing the concept of domain specific models represents an extension to the general purpose design environment MagicDraw.

Mainly the international standards in the domain of information security of embedded systems are MARTE (MARTE, 2011) and ISASecure EDSA. Model-driven design and analysis of embedded devices and real time systems are presented in MARTE framework (MARTE, 2011) defining a complex UML based conception of software and hardware qualities of a device to support its specification, synthesis, verification, validation, performance evaluation, quantity analysis and device certification with the use of UML profiles. ISASecure Embedded Device Security Assurance Certification represents an international standard proposed for provision of certification and specification processes towards secure embedded devices.

Note however UML based software tools for design and verification are targeted more on development of static structure of devices, their specification and successive software/hardware implementation without evaluation of dynamically changing characteristics such as resource consumption laying beyond the scope of conventional UML apparatus.

3 Design, verification and testing approaches

In this section we expose the core elements of the technique for design, verification and testing of systems with embedded devices. The technique comprises the following phases:

1. configuring security components of an embedded device;
2. verification of its protection system to reveal hidden conflicts;
3. verification of network information flows;
4. testing IoT systems to detect anomalous data from sensors.

The essence of the technique lays in application of heuristically obtained embedded security knowledge as completed design and verification patterns together with the use of methods in the fields of model checking, discrete optimization and decision making.

3.1 Configuring security components

Design of protection mechanisms for embedded devices includes issues of security component selection among the sets of available alternatives according to the characteristics of the particular system and its scenarios.

The belonging of the target system to IoT determines the importance of considering various non-functional constraints, such as limitations to available hardware resources of the device, restrictions on energy resources, restrictions on physical characteristics such as size, weight of the device and its individual components, cost restrictions, etc. in the design process.

Along with the similar functional requirements, the difference in non-functional constraints affects significantly the choice on the set of alternative security components. For example, selection of a module for the protected data storage greatly depends on, first, the type of the device it is embedded to (for example, a portable barcode scanner based single board computer or ATM) and, second, the volume of the stored data. As a result, even under similar requirements to the protection functional, eventually various sets of security components appear to be the most effective for different systems.

A configuration represents a bundle of security components, their properties and ties specified between the components. As an example, the triple $(e_{len}, a_{tm}, s_{vol})$ specifies a security configuration for a mobile communication device in a mesh network, where e is the symmetric encryption algorithm AES (with the given key length $len = 192$), a represents an attestation component within the module TPM (Trusted Platform Module) with the frequency value of the checking $tm = 120$ s), s is a specific security token for secure storage on the device (with the values of the storage size $vol = 2$ MB).

In the paper we exploit a so called multi-component approach to organize the combined embedded security mechanisms, considering security and resource based requirements as well as specific resource consumption criteria to get the most efficient security components customized by resource based constraints of a specific embedded device.

In essence, in the process of combining a set of security components the available non-functional restrictions are ordered by their criticality degree for the device. As a result such development of a security mechanism for embedded devices is transformed to formulation and solution of a discrete optimization problem on a set of combinations of security components. The solution of this problem allows getting the optimal security configuration (Desnitsky et al. 2012).

Starting from the extracted information on the security components and resource constraints the approach is aimed at finding a configuration that consumes device system resources in the most effective manner. Eventually being embedded into the device's security mechanism such optimal configuration will allow enhancements in security efficiency as well.

Configuration process runs with the use of the produced decision making software for choosing security components and therefore it is characterized by an inherent level of automation. At that, non-functional criteria are assigned as input data, depending on the resource constraints. Besides we offered a specific heuristic governing the order of application of the particular non-functional constraints in the configuration process.

3.2 Detection of hidden conflicts between security components

The use of the multi-component approach to embedded and real-time system development (MARTE 2011) in design of embedded security mechanisms results in possibility of errors and incompatibilities between various security components used. Specifically the absence of any interior mistakes inside particular security components does not mean no hidden conflicts appear at the components integration into an entire security mechanism.

As a result the presence of these conflicts leads to security vulnerabilities, instable running of both security and target functions of the embedded devices. The core difficulty here is that often such conflicts turn out to be detected in the system exploitation phase solely. Thus resolution of such conflict may request much industrial expenses and extra charges. Revelation of already known types of latent conflicts of security components become a serious design-time concern. In Sect. 4 we expose a range of typical conflicts as a specific expert knowledge as well as some examples.

Due to distinct nature of each particular one these conflicts have been identified individually through a heuristic analysis of a number of available industrial IoT with the use of some theoretic works (Burleson et al. 2012).

3.3 Verification of network information flows

Generally analysis of network flows in IoT systems is applied as an element of the whole security analysis mechanism. The verification allows examination of a security policy conformity, proving if the recognized network information flows between the devices do not cross the security policy.

In according to the conventional comprehension an information flow is treated as a piece of data transferred from one object in the system to another one. At that an information flow security policy embodies a range of rules for allowance and forbiddance of particular kinds of information flows.

In general information flow analysis is conducted at three levels, (1) hardware—as an analysis of ties between the microcircuits (Braghin et al. 2011), (2) software—as an analysis of the source code running on the device (Pistoia et al. 2007), and (3) network—as analysis of network connections in systems with embedded devices. At these three levels information flow analysis is covered in detail in existing literature (Pistoia et al. 2007; Hwang et al. 2006).

However quantity of papers on verification of network information flows is significantly less than ones on software and hardware flows. The concept of information flow is broadly exploited in security evaluation of a route and network effectiveness analysis (Agaskar et al. 2010; Sprintson et al. 2009).

Although these studies are not directly connected to the types of data passed by information flows in the network, but nonetheless, they can be used for modeling information flows. Conventionally information flows between nodes are specified as a directed acyclic graph. Thus, to expose covert channels the topological analysis described in (Rae and Fidge 2005) can be applied to this graph. Model checking was applied in the paper to check correctness of security policy rules for information flows.

In general, checking correctness of network information flows is an integral part of the design process. Conducting such kind of verification at beginning stages of the design process ensures early revelation of contradictions in the security policy and inconsistencies of the information system topology.

In essence verification of network information flows is a part of static analysis and can be applied at early phases of the system development process. Unlike dynamic analysis realizing checking physical device instances by modeling specific attacks the verification is targeted at declining amounts of design bugs and security incompatibilities in the system. Thus static analysis deals more with design and security specifications and models of various abstraction levels as well as behavior scenarios (Kotenko and Polubelova 2011).

In the paper to verify a security policy by checking network information flows we propose the use of model checking, namely SPIN tool and PROMELA language.

To verify information flows we exploited a model of the system instead of using a physical IoT implementation, the more technically complex task involving ad hoc software/hardware modules and qualified staff. During verification the policy rules being ordered according to their priority are listed sequentially until a particular rule holds.

3.4 Testing IoT systems to detect anomalous data from sensors

A peculiarity of IoT is that it is characterized by a possibility of cyber-physical attacks combining both network protocol, software and data targeted actions and physical impacts on hardware circuits. An important instrument to reveal such cyber-physical attacks is to analyze data streaming from embedded device sensors in the system, catching specific anomalies, i.e. wrong values of such data. In the paper we regard the following three types of cyber-physical attacks:

1. either direct or indirect physical influences on a sensor, taking into account the further informational effect on the system components as a consequence of data destruction. An example of such physical influence is a laser effect on a temperature sensor at a visibility distance (Ruiz et al. 2012a, b). Another example is an influence on a QR or barcode scanner or biometric authentication component by changing the given sample to a fake one, regarding malefactor's preparatory computational elaboration of false content and estimation of the expected system reply to such changing;

2. delivering altered electrical signals directly to the particular digital or analog pins of the device, sensor substitution attack;
3. a diversity of impacts on devices in the system not only specific to IoT systems (informational attacks), including an impact on common communication interfaces of a device (Bluetooth, Wi-Fi, etc.); substitution of data from sensors during their storage in the form of specific data structures in the memory (e.g. by means of a malicious software); substitution of such data when they are transferred to other embedded devices in the system.

Generally such anomalies in data from sensors are identified dynamically in the runtime by application of previously defined rules and restrictions (expert knowledge) with the use of the history of sensor readings, values of the current time, date and month, admissible restrictions on the measured value, internal consistency of these data as well as their conformity to any exterior data sources (such as databases, clouds and so on.), business rules of the system, behavioral features of end users of the devices, physical geo location of the devices and their movement in space.

4 Analysis of expert knowledge sources

We used a number of industrial systems with embedded devices (case study) as a source of expert knowledge in the field of embedded systems, namely a system of remote automated control of energy consumption by consumers (*abr. MD*), a quickly deployable emergency management system (*abr. TMN*) and a system providing consumers with digital media services with the use of a Set-top-box (*abr. STB*) as well as a non-commercial (research oriented) system of devices for a room protection perimeter (*abr. PPS*) (Desnitsky et al. 2015).

The choice of these case studies is conditioned by their different structure, business goals, functional and security features. The expert knowledge got through the analysis of these systems can be generalized and exploited as a completed design and verification patterns in the development of new systems.

The following patterns, constituting the suggested technique, are connected directly to expert knowledge. These represent concrete security requirements in the form of functional security properties and possible security component alternatives; information on non-security features and interior connections of both an embedded device and its security system to be the stem of resource consumption construction; possible types of conflicts that security components are involved in; possible types of information flow anomalies and methods for their revelation.

A short exposition of the system STB developed by Technicolor for providing services of digital streaming media data to customers is described below.

The central unit of the system for providing services of digital streaming video and audio data to customers is a digital receiver Set-top-box, which is responsible for decrypting protected media data and provides users with an access to multimedia services such as PayTV, Video on Demand, surfing the web, purchasing and running applications from AppStore and others. Expanding the functionality of such devices associated with the new communication and multimedia capabilities in particular is driven by the need to their further protection and protection of the services they provide (see Fig. 1).

As key elements of the implemented functionality the developers of the system single out the following ones: DRM service (Digital Right Management) allowing a user to load and display certain protected media content and AppStore functions that allows downloading and installation of software applications on the digital receiver.

In fact in the former the digital receiver represents a DRM player receiving an encrypted video data from a remote server (Movie Store Server). In the latter it is assumed the user is able to download applications from the remote server, using a software client AppStore, which verifies signatures of the application and installs it.

The digital receiver represents a device Asus EEE Box PC based on processor Intel Atom Pine Trall D510 1.6 GHz and operating system Ubuntu Linux Kernel v3.5, which implements a principle of separation of system resources between processes.

The basic requirements to the protection of the system are:

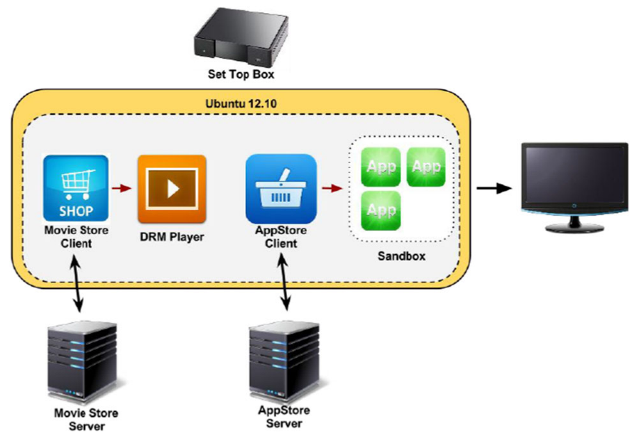


Fig. 1 A system providing services of digital streaming media data to customers (SecFutur)

Table 1 A heuristic to choose security components

Resource type according to MARTE	Features of embedded devices and its services	Abbreviation of the systems with devices of the feature	Rank
HW_PowerSupply (energy consumption resource)	The presence of a permanent power source	*MD, STB, PPS*	0
	Possibility of replacing the device or battery without damage to the provided services	*TMN*	1
	Sporadic access to a centralized power supply	*TMN*	1
	High dependency of the mission goal achievement on energy resources	*TMN, PPS*	2
HW_StorageManager (storage resource)	The device does not store large amounts of data, loss of data is not critical	*MD*	0
	Storing large amounts of data, loss of data is not critical	*STB*	1
	Storing large or unlimited amounts of data, the loss is critical	*TMN, PPS*	2
HW_Computing (computational resource)	No complex calculations, no requirements of message delivery timeliness	–	0
	No complex calculations, major timeliness	*MD, PPS*	1
	Complex calculations, minor timeliness	*STB*	2
	Complex calculations, major timeliness	*TMN*	2
HW_Communication (communicational resource)	No communications (or they are not obligatory for the device services)	–	0
	Importance of communications for the device services, minor data volumes	*MD, PPS*	1
	Importance of communications, large data	*STB, TMN*	2
HW_Physical (physical characteristics of the device and components)	The devices are of the size and weight of permanently installed systems	*TMN*	0
	The devices are of the size and weight of portable systems	*MD, STB, TMN*	1
	The devices are of the size and weight of single-board computers	*MD, TMN, PPS*	2

- the need to verify authenticity of the device to counteract a device substitution attack;
- the need to constantly check authenticity of software and hardware devices during its mission;
- integrity of data transferred to the device and sent from it;
- integrity of data stored at the device;
- confidentiality of data transferred to the device and sent from it;
- confidentiality of data stored at the device;
- the need for end user authentication on the device;
- the need for monitoring anomalies in data and traffic.

The process of combining security components of devices in the system should be conducted to meet the security requirements to its devices and services as well as compatibility and resource constraints. The process of the search of typical conflicts of security components and anomalies in the system represents a heuristic analysis of specifications and system models, considering already known types of conflicts and anomalies mentioned in Sect. 5. In particular for the STB use case the policy rules forming a shadowing anomaly have been investigated.

5 Expert knowledge

5.1 Configuring security components

Lexicographic ordering of the specified non-functional constraints is used in order to select an optimal configuration The ordering is based on a heuristic to determine the order of consideration of non-functional constraints in the configuration process, depending on the functional and non-functional features of the configurable device. The heuristic is based on expert knowledge derived from the analysis of four complex systems (system MD, TMN, STB, PPS).

The heuristic defines a general algorithm for prioritization of non-functional constraints of an embedded device. A set of features of embedded devices and the services they provide, having the influence on resource consumption, physical characteristics, cost and other device and component properties is specified. A three-point ranking was proposed for resources according to their criticality to execution of the target device functions (0 means the resource is noncritical, 1 means low criticality and 2 means high criticality). By experts a rank value is specified for each feature of the core device of each of the four systems in use.

Table 1 shows the five types of non-functional constraints in accordance with the methodology MARTE (2011), a set of features for each of them, references to the analyzed systems that have devices with the regarded features and the corresponding ranks. Thus, the ranks obtained on the basis of expert evaluation of the analyzed systems are taken as ranks of the features themselves. Hence these rank values can be used for express ranking of non-functional constraints of the device by its developer without additional participation of experts.

Thus, in configuring process the device characteristic features are identified from the list of available ones. After that each resource is assigned a maximum value of rank over all held features corresponding to a given constraints. As a result, the considered non-functional constraints are ordered according to decreasing their ranks. If two or more non-functional constraints have the same rank value, the default order <*HW_PowerSupply, HW_StorageManager, HW_Computing, HW_Communication*> is used. It was defined by experts a priori and is typical for the most existing systems. It is assumed if necessary this heuristic may be refined by adding additional features, constraints, analyzed systems and devices to consider as expert knowledge.

5.2 Hidden conflicts of security components

Development of combined a complex security mechanism for embedded devices includes design-time isolation of implicit security conflicts between its particular components. In fact such revelation represents a heuristic based search in particular system specifications for the conflicts already known in the field (Desnitsky and Kotenko 2014).

In general a conflict is regarded as a relationship between two or more security components and represents a contradiction between the functional of several security components, any of their non-functional limitations and/or software/hardware platform of the device. The specificity of such conflicts is that as a rule they become visible under certain conditions only. Hence it is difficult to detect them by fuzzing or any other type of testing on physical devices. Revelation of conflicts in the beginning will reduce the complexity of the whole IoT system development process, lessening an amount of its iterations.

Besides some conflict emergence is determined not only by the fact of integration of a number of specific security components but also the way of their integration. In particular two components with opposite protection features are in a conflict if they are performed simultaneously and interact within a common hardware/software context, for example, they use share data structure, memory, file, communication channel and so on.

Knowledge of known types of conflicts is produced by expert analysis, modeling and development of new information systems with embedded devices. It seems appropriate to keep a list of previously discovered conflict types, considering domain-specific nature of each particular system. As a consequence the specification of the combined device protection system as well as specifications of considered security components should be analyzed together by the developers for the presence of conflicts from the list.

Differences between the nature of each particular conflict, amounts of the involved security components and their protection functionality, peculiarities of the components interactions and their integration as well as domain-specific character cause the development of comprehensive classification covering all possible implicit conflicts seems infeasible at the moment. However, in the design process a particular classification of conflicts (e.g. according to the type of the involved objects) can be used as an expert knowledge by the device developer of the combined protection system to realize a directional search of possible conflicts. Table 2 depicts three possible types of security component conflicts with examples.

The conflicts assume a highly specific manner of their elimination, depending on peculiarity of the involved security components and their restrictions. To resolve a conflict a reconsideration of one or several security components, modification of the method they are integrated or even changing the system specifications are possible.

5.3 Network information flow verification

For verification of network information flows the instances of security policy anomalies and methods for their detection as expert knowledge are used. Consider one type of anomalies more in detail, "shadowing" anomaly. The presence of this anomaly supposed that a rule never works because there are one or more rules with higher priorities "overlapping" it. This anomaly indicates a probable error in the policy, which should be reviewed.

Network information flows and policy rules are specified by the following tuples:

$$\begin{aligned} \mathit{InformationFlow} = {} & < \mathit{host}1, \mathit{host}2, \mathit{user}1, \mathit{user}2, \\ & \mathit{interface}1, \mathit{interface}2, \mathit{type} >, \end{aligned}$$

$$\begin{aligned} \mathit{FilteringRule} = {} & < \mathit{host}1, \mathit{host}2, \mathit{user}1, \mathit{user}2, \\ & \mathit{interface}1, \mathit{interface}2, \mathit{type}, \mathit{action} >, \end{aligned}$$

where *host1*, *host2*—sending and receiving hosts, respectively; *user1*, *user2*—user sending and receiving user; *interface1*, *interface2*—types of hardware Interfaces of the sender and recipient; *type*—type of the information flow.

Table 2 Security component conflicts

Conflict type	Conflict description	Conflict example
Type 1	Conflict due to a lack of consistency between a security component and the device specification	*Security_component* = "TPM based secure module for storing confidential customer data"; *Safety_requirement* = "to double customer data by an extra hardware storage module"; *Conflict* = "assuming the only TPM in the device the unprotected doubling violates data confidentiality"
Type 2	Conflict between the protection functions of several security components	*Security_component_1* = "backup component for critical customer data"; *Security_component_2* = "component for secure guaranteed deletion of critical customer data after some specific event happens"; *Conflict* = "inconsistent application of the both components to the same data causes a conflict due to a logical contradiction of their security features"
Type 3	Conflict between several basic components within a complex security component	*Security_requirement* = "to implement RAID based redundant and high-performance storage of business data by two (or more) secure hardware units"; *Assumption* = "the inconsistent parameters of the units (e.g. different capacity of the units or their writing speeds)"; *Conflict* = "the units are correct themselves, but they do not implement RAID"

Type of information flow refers to a kind of data that the flow encapsulates. Information flow types by both the kind of transmitted information (e.g., user data, critical data, checksums, encryption keys, security certificate, etc.) and the format which the information is presented in (e.g., unencrypted and encrypted messages, compressed message).

The essence of model checking, applying to anomaly detection consists in iterating states the system can move into, depending on the emerging information flows and responses from the component making decisions on policy based permission or rejection of such requests.

When iterating the sequence of actions depends on conditions formulated in a language of linear temporal logic and express correct states of the system (Chechulin et al. 2012). A state of the system is determined by a set of variables and state change is caused by concurrent processes running in the system.

A process to be started in the next time is chosen randomly. The system considers all the possible sequences of steps for specific processes and signals potentially incorrect state. After that, the user is given a track, i.e. a sequence of steps leading to an incorrect state of the system with respect to given conditions.

Basic input of verification of network information flows includes, first, descriptions of policy rules and, second, the structure of the network in the system description language and detectable types of anomalies.

At the first stage of verification input data is converted into an internal format of the verification system. Then, at a second stage, a general model of the system is built to verify prohibiting and permitting rules for information flows. The model is presented in the form of a finite state machine and initialized by the input data in internal format. In the model the anomalies are expressed by formal statements. According to model checking paradigm these formal statements represent properties of correctness, whose violation would lead to an incorrect system state. At the third stage the general model is verified by a model checker tool. In the verification process all incorrect states of the system are revealed. At the final verification stage the obtained results are evaluated. If any anomaly instances are detected, a description is created that contains the situation and the information flow leading to the appearance of the anomaly and its type.

For the case of a shading anomaly the verification consists of: (1) generating a set of testing flows (the flows are formed on the basis of the so called "boundary" values of the policy rules, i.e. the flows are constructed through any possible combinations of parameters taken from the rule statements); (2) sequential application of the policy to each information flow; thus each time the rule holds it is marked as *held*; (3) search on the set of rules to identify rules did not hold even once.

Therefore, verification allows getting a set of results, each of them being a pair $<A, (B_1,\ldots B_n)>$, where A represents an anomalous rule, $B_1,\ldots B_n$ are higher priority rules shadowing it. $B_1,\ldots B_n$ are isolated by an extra pass of the policy by running those testing flows that meet the conditions of rule A.

5.4 Testing of IoT systems to reveal anomalous data from sensors

We produced a software module for runtime tracing anomalies in data from IoT sensors on the base of the obtained expert knowledge. A prototype of Smart Home system are used to demonstrate it. Primarily it is presumed data anomalies appear mostly in consequence of a purposeful attack on data from sensors. The monitoring is organized as an iteratively repeated checking feasibility of each assigned constraint on values of data streaming from sensors. In Table 3 there are instances of particular constraints formed for the mart Home prototype anomaly monitoring.

Breaking such constraint is insufficient to deduce automatically some attack on IoT sensors mounted. Besides malicious actions the anomaly may occurred due to various technical failures or non-security specific exceptions. At that this monitoring functionality allows IoT system operator to pay more attention on the system for deeper security investigation and subsequent reaction.

6 Implementation

A software prototype produced to support the technique of design, verification and testing of systems with embedded devices. The prototype comprise a design-time means for making decisions on finding optimal configurations and for verification of network information flows.

A fragment of a software interface of the developed tool is shown in Fig. 2. Input data including functional security requirements, non-functional constraints (Target System properties) and properties of the software and hardware compatibility (target system platform), are used for combinatorial enumeration of available alternatives of security components (called as available SBBs, security building blocks) to find and give information on the optimal security configuration (optimal configurations) to the user.

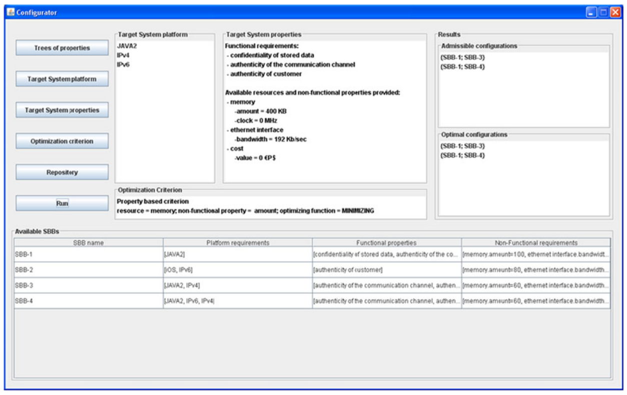
The proposed verification of network information flows with the use of model checking and Promela language has been implemented for the analysis of security level of the system providing consumers with digital media services with the use of a Set-top-box (system *STB*) with the following limitations (SecFutur).

As a step towards creation of a software component for monitoring data anomalies we produced a fragment of a software prototype of a Smart Home system on the base of Raspberry Pi (Fig. 3). This prototype includes a microcircuit completed by a temperature sensor and an infrared LED for remote control of an air conditioning appliance located inside the Smart Home. A developed component for monitoring anomalous data getting through it is installed on the controller. Actions of an attacker are modeled with the use of Raspberry Pi as well singling out the following four basic possible attacks. These are (1) physical effect on the temperature sensor, (2) increased voltage attack to the IR LED for its failure, (3) attack on pins of the device or its communication interface (e.g. an Ethernet interface), (4) an attack on the anomaly detector software component.

An attack on the temperature sensor of the system aimed at exceeding the real readings was realized. In compliance with the business logic of the system the attack results in switching on energy-consuming air conditioning appliance, aimlessly wasting energy (Moyers et al. 2010) and resulting in extra costs of the owner Smart Home. In essence the modeled attacking actions represent a cyber-physical attack. Its physical part is a direct replacement of the genuine sensor by a modified propagation one, while the information part presents a preliminary investigation of the object of the attack by the intruder and preparation of the necessary attacking software/hardware, including definition of the corresponding electrical parameters to assign on the controller. Continuous tracing by the monitoring component through checking the specified constraints on

Table 3 Expert knowledge of anomalies in data from sensors

Type of constraints	Constraint examples
1. Constraints of business logic of a particular Smart Home system	In accordance with the target requirements the indoor temperature of Smart Home must not exceed 30 degrees Celsius and fall below 5°
2. Restrictions, reasoning from the whole structure of the system as a result of links between the different parts of the system in the process of integration	If two or more analogous sensors show a fundamentally different values (assuming the absence of reasons for this difference), this case is considered as an anomaly, and one could draw a conclusion on a possible attack at least to one of these sensors
3. Constraints, reasoning from the time flow	Light and temperature outdoors sensors in Smart House demonstrating obviously incorrect values of brightness and temperature for this time of year, considering its geo location data (including unexpected changes variations in sensor readings)
4. Constraints based on the previous history of the sensor readings	A motion sensor in the office stopped passing data (or output incorrect data) on the movement of employees in working hours. It is a possible sign that the sensor was attacked for the purpose of further illegal entry into the office
5. Constraints due to natural technical conditions of electronic circuits of the system and/or its security components	The value of the voltage passed to hardware pins of embedded device should not exceed 3,3 (or 5) volts



Target System platform
JAVA2
IPv4
IPv6

Target System properties
Functional requirements:
- confidentiality of stored data
- authenticity of the communication channel
- authenticity of customer

Available resources and non-functional properties provided:
- memory
  -amount = 400 KB
  -clock = 0 MHz
- ethernet interface
  -bandwidth = 192 Kb/sec
- cost
  -value = 0 €P$

Results
Admissible configurations
(SBB-1; SBB-3)
(SBB-1; SBB-4)
Optimal configurations
(SBB-1; SBB-3)
(SBB-1; SBB-4)

Optimization Criterion
Property based criterion
resource = memory; non-functional property = amount; optimizing function = MINIMIZING

Available SBBs

SBB name	Platform requirements	Functional properties	Non-Functional requirements
SBB-1	[JAVA2]	[confidentiality of stored data, authenticity of the co...	[memory.amount=100, ethernet interface.bandwidt...
SBB-2	[iOS, IPv6]	[authenticity of customer]	[memory.amount=80, ethernet interface.bandwidth...
SBB-3	[JAVA2, IPv4]	[authenticity of the communication channel, authen...	[memory.amount=60, ethernet interface.bandwidth...
SBB-4	[JAVA2, IPv6, IPv4]	[authenticity of the communication channel, authen...	[memory.amount=60, ethernet interface.bandwidth...

Fig. 2 GUI fragment of the configuration tool

temperature sensor values enabled success revelation of this attack.

In more detail the realized attack included substitution of the sensor to a prepared attacking Raspberry Pi based unit with the output pins connected to the corresponding input pins of the controller. At that the attacking unit generates the desired electrical signal and is controlled by a remote attacker using Wi-Fi module.

To detect abnormal data sets of '*if (A) → warn (descr)*' rules are specified, where condition *A* causes generation of a warning *warn* about a possible attack with its text description *descr* (Table 4). Values t, t1, etc. indicate corresponding readings in Celsius taken from the temperature sensor. Below there is a fragment of the monitoring algorithm implemented in Java with regards to checking assertions on data from the sensor. *Assertion* interface encapsulates validation of data to check particular types of constraints, if necessary allowing adding new ones.

```
interface Assertion{
        public abstract boolean assert();
        public abstract String getWarning();
} ...
while(true){
        for(Assertion a : assertions)
        if(a.assert()){
                send(a.getWarning());
        } ...
        break; ...
}
```

The attack was conducted on the fly (without a suspension of the target system). Rule #3 was applied to detect it by the monitoring component. This rule enabled us to trace a short-term anomaly, namely for a few seconds neither the temperature sensor nor the attacking unit was connected to the pin of the controller.

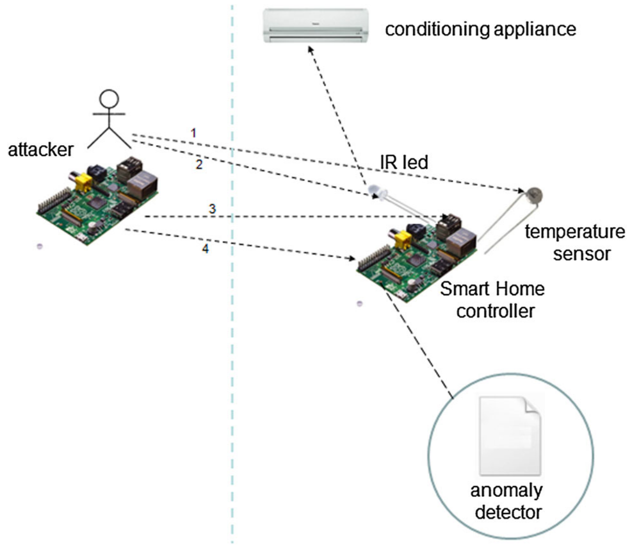


Fig. 3 Fragment of Smart Home system

Table 4 Implemented rules for monitoring anomalous data from a Smart Home sensor

Num.	Rule
1	if($t > 22$) → warn$_1$ ("Excessive temperature")
2	if($t < 18$) → warn$_2$ ("unacceptably low temperature")
3	if(t_2-$t_1 > t_0$ && no(event$_0$)) → warn$_3$ ("sudden temperature change, e.g. possible illegal penetration into the Smart Home without any recorded event (event$_0$) leading to the change")
4	if($t_2^{(1)}$-$t_1^{(1)} > t_0$ && $t_2^{(2)} = t_1^{(2)}$) → warn$_4$ ("readings change occurred only on one of the two redundant sensors, so possible there was an attack on one of them")
5	if($t < 5$) → warn$_5$ ("lowering the temperature below the critical limit")
6	if(switched_on/off(conditioning) && $t_1 = t_{2)}$ → warn$_6$ ("the air conditioner switching on/off mechanism may not functioning, e.g. IR LED burned out due to an attack on it") & warn$_7$ ("a hardware failure may happen in the air conditioner")
7	if(started(controller)) → warn$_8$ ("improper shutdown of the controller with a possible subsequent attack on it")

Note the attacker is assumed to get physical access to the target device during the attack. Within our prototype it is assumed the attacked device has no seals and secure boot as well as it does not use any calibration certificates. Generally the latter one would ensure the device is considered as a trusted one.

Propagation of this attack can be organized by malware and ultimately can lead to major blackouts (Koopman 2004).

7 Evaluation and experiments

Evaluation of the results includes experimental studies based on the developed prototypes as well as confirmation of the applicability in practice and proof-of-the-concept on fragments of the IoT systems implemented.

In particular, for the configuration approach suggested we carried out its experimental comparison to an alternative method of selecting security components on the base of the greedy algorithm.

The network information flow verification part was evaluated in a proof-of-the-concept manner. As an example on a fragment of a security policy for an IoT system we showed detection of an error in the policy by application of the proposed verification.

Identification of hidden conflicts between the security components is based on purely heuristic specifics. At that both types of the conflicts and their resolution have an exceptionally individual character. Thus such conflicts are revealed and resolved by an expert analysis without any automation, therefore this issue is not disclosed in this section.

For the problem of monitoring of anomalies of data from sensors we performed its proof-of-the-concept, providing experimental based test of how the four specific types of attacks can be detected by means of the proposed approach.

7.1 Configuring security components

Experiments on modeling strategy for selecting pseudo optimal sets of security components on the base of greedy algorithms have been implemented. The strategy represents a procedure for a sequentially organized choice and refinement of security components of the sought configuration iteratively for each security requirement. In fact, this procedure works successively, for each functional protection property choosing a security component from the available ones that consumes the least amount of hardware resources according to their order determined by the heuristic. The experiments carried out showed that in average the produced technique yields more efficient security configurations than the alternative strategy. Here the security configuration efficiency is regarded as minimal resource consumption by the resultant security configuration. The more itemized demonstration of produced security component design tool as well as its assessments are presented in (Desnitsky et al. 2012).

The developed tool can be compared with the following alternatives, SPT (SecFutur), MagicDraw, UModel, Modelio, etc. These tools are mostly UML based and are coupled with specific UML profiles to present and model relevant information on security requirements, components, templates and security evaluation of embedded devices. Particularly SPT tool implements a conception of domain security metamodels (DSM) allowing construction a complex model of the system on the base of combination of single models for each required security functionality. An advantage of such tool is delimitation subtasks of the design process, responsibilities and involved roles.

However these software tools are directed more on development of the static structure of the devices, their specification and subsequent software/hardware implementation, while the developed configuration tool besides organization data on devices and security components implements design-time decision making procedures on this information, considering various non-functional features, possible conflicts and anomalies.

7.2 Verification of network information flows

The experimental studies involved testing shadowing anomalies in an IoT system case study. These anomalies simulate potential mistakes in the process of the policy development. All the anomaly instances put into the policy within the experiment were detected successfully. When the verification process finished the initial policy was corrected and after that the repeated verification stated the corrected policy was free of shadowing anomalies.

Figure 4 uncovers a fragment of *STB* system security policy, containing three rules specified in PROMELA language. *Rule1*, *rule2* and *rule3* together form an instance of shading anomaly. According to the priority of the rules (*rule 1* has priority over *rule 2* that has priority over *rule 3*) rule 1 and rule 2 individually do not overlap *rule 3*, but together they make it unfeasible. Virtually this anomaly is the result of an incorrect specification of user groups in the information flow policy.

Figure 5 shows the logs of the verifier SPIN, which provides information about this anomaly.

The experiments on modeling a large number of involved objects, roles, data types and permitting/prohibiting rules confirmed the effectiveness of the proposed verification for the design of the system providing consumers with digital media services with the use of a Set-top-box (*STB*).

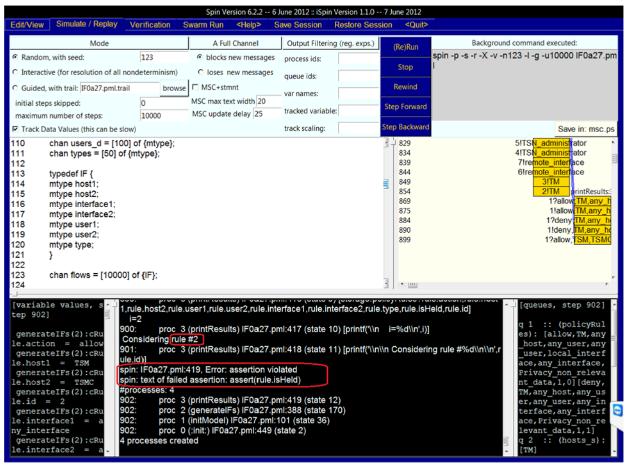
Due to the fact that basically such conflicts and anomalies are revealed in a heuristic manner there is no a unified method to resolve them. Elimination of a conflict/anomaly is determined, first of all, by its context including specific security requirements and assumptions, information security risks, modes of the device, involved security components, used interfaces, etc. To verify network control information flows of a security policy it is not sufficient to use pairwise comparisons of the policy rules only. In reality an analysis of the policy rules holdings in dynamic (i.e. model checking) is needed. Generally in comparison with

```
rule1.user1 = User_group1;
rule1.user2 = any_user;
rule1.interface1 =  local_interface;
rule1.interface2 = any_interface;
rule1.host1 = Terminal1;
rule1.host2 = any_host;
rule1.type = Privacy_non_relevant_data;
rule1.action = allow;
rule1.isHeld = false;
rule1.id = 1;
storage.policyRules!rule1;

rule2.user1 = !User_group1;
rule2.user2 = any_user;
rule2.interface1 = local_interface;
rule2.interface2 = any_interface;
rule2.host1 = Terminal1;
rule2.host2 = any_host;
rule2.type = Privacy_non_relevant_data;
rule2.action = allow;
rule2.isHeld = false;
rule2.id = 2;
storage.policyRules!rule2;

rule3.user1 = any_user;
rule3.user2 = any_user;
rule3.interface1 = local_interface;
rule3.interface2 = any_interface;
rule3.host1 = Terminal1;
rule3.host2 = any_host;
rule3.type = Privacy_non_relevant_data;
rule3.action = deny;
rule3.isHeld = false;
rule3.id = 3;
storage.policyRules!rule3;
```

Fig. 4 Example of rules containing a shadowing anomaly



Mode
- (●) Random, with seed: 123
- ( ) Interactive (for resolution of all nondeterminism)
- ( ) Guided, with trail: IF0a27.pml.trail
- initial steps skipped: 0
- maximum number of steps: 10000
- ☑ Track Data Values (this can be slow)

A Full Channel
- (●) blocks new messages
- ( ) loses new messages
- ☐ MSC+stmnt
- MSC max text width 20
- MSC update delay 25

Output Filtering (reg. exps.)
- process ids:
- queue ids:
- var names:
- tracked variable:
- track scaling:

Background command executed:

```
spin -p -s -r -X -v -n123 -l -g -u10000 IF0a27.pml
```

Save in: msc.ps

```
chan users_d = [100] of {mtype};
chan types = [50] of {mtype};

typedef IF {
mtype host1;
mtype host2;
mtype interface1;
mtype interface2;
mtype user1;
mtype user2;
mtype type;
}

chan flows = [10000] of {IF};
```

```
829    5!TSN_administrator
834    4!TSN_administrator
839    7!remote_interface
844    6!remote_interface
849    3!TM
854    2!TM           printResults:
869    1?allow,TM,any_h
875    1!allow,TM,any_h
884    1?deny,TM,any_h
890    1!deny,TM,any_h
899    1?allow,TSM,TSMC
```

```
[variable values, step 902]

 generateIFs(2):cRule.action = allow
 generateIFs(2):cRule.host1 = TSM
 generateIFs(2):cRule.host2 = TSMC
 generateIFs(2):cRule.id = 2
 generateIFs(2):cRule.interface1 = any_interface
 generateIFs(2):cRule.interface2 = a
```

```
1,rule.host2,rule.user1,rule.user2,rule.interface1,rule.interface2,rule.type,rule.isHeld,rule.id]
  i=2
900:     proc  3 (printResults) IF0a27.pml:417 (state 10) [printf('\\n    i=%d\\n',i)]
 Considering rule #2
901:     proc  3 (printResults) IF0a27.pml:418 (state 11) [printf('\\n\\n Considering rule #%d\\n\\n',rule.id)]
spin: IF0a27.pml:419, Error: assertion violated
spin: text of failed assertion: assert(rule.isHeld)
#processes: 4
902:     proc  3 (printResults) IF0a27.pml:419 (state 12)
902:     proc  2 (generateIFs) IF0a27.pml:388 (state 170)
902:     proc  1 (initModel) IF0a27.pml:101 (state 36)
902:     proc  0 (:init:) IF0a27.pml:449 (state 2)
4 processes created
```

```
[queues, step 902]

q 1  :: (policyRules): [allow,TM,any_host,any_user,any_user,local_interface,any_interface,Privacy_non_relevant_data,1,0][deny,TM,any_host,any_user,any_user,any_interface,any_interface,Privacy_non_relevant_data,1,1]
q 2  :: (hosts_s): [TM]
```

Fig. 5 The technique output

the classical network architecture, the specificity of information systems with embedded devices in the task of verification of network information flow contains presence of a branched network topology based on heterogeneous embedded devices with different types of communications and types of hardware/software interface being entry and exit points for information flows, and variability of the structure of such systems throughout its work. An advantage of the proposed verification of information flows is to ensure the system security, assuming the same behavior of the model and the real system.

Conventionally information flow analysis of a system with embedded devices is carried out both on hardware level, analyzing physical links between particular microcircuits (Braghin et al. 2011), and software one, when the source code of a device is analyzed by the use of control flow and data flow graphs (Pistoia et al. 2007). Unlike existing software/hardware tools such as SIFA (McComb and Wildman 2006), implementing verification of software and/or hardware flows (Rae and Fidge 2005), the network level information flow control produced by us allows detection of anomalous behavior of the system and errors in the policies of network information flow control.

7.3 Testing of Internet of Things systems to detect anomalous data from sensors

Evaluation of the proposed monitoring approach includes an analysis of its applicability in practice and proof-of-the-concept (see Table 4). The approach is aimed at detecting attacks on a particular IoT system with sensors being data sources for the business processes of the system and information security processes. A limitation of the approach is direction to detect those attacks that are directly related to generation of false data from sensors or its manipulation. The conducted studies allowed formulating the following recommendations.

The development of an IoT system should include an analysis of subjection of the system to attacks on data from its sensors. Namely, it is required to perform the following five steps: (1) make a list of the sensors of the system, (2) determine for each sensor the process of its data processing/transfer, (3) determine possible attacks on such data in terms of the business logic of the system, (4) form a list of constraints on data values, i.e. the constraints to be broken by the attacks (in fact to form a subject-dependent monitoring rules) and (5) implement the procedures of checking the stated constraints and embed them into the monitoring module as means for counteraction to the attacks.

The correctness of this approach was successfully confirmed in practice by the developed Raspberry Pi based prototype realizing a fragment a smart home system (Fig. 5). Table 5 shows the results of checking the rule based monitoring module with constraints against four attacks specified above. Columns 1 and 2 depict an attack sequence number and its description respectively. Column 3 shows the possible additional conditions of the attack. Column 4 depicts if the attack was successfully detected or

Table 5 Experimental results on modeling attacks

#	Attack description	Extra conditions	Attack detection	Conclusions
1	Physical effect on the temperature sensor	A significant deviation from the expected values of the temperature sensor	Detected	The proposed protection demonstrated its effectiveness against this attack within the established conditions
		A small deviation from the expected values of the temperature sensor	Not detected	Protection against such attacks requires a feedback on IoT system by its architecture adjustment. For example, redundant sensors or history data checking (i.e. some accumulated statistics on the sensor data) can be proposed. Note that each of these improvements can increase the complexity of an attack, but does not exclude it completely
2	An increased voltage attack to the IR LED for its failure	–	Detected	The experiments confirmed the discovery of this attack in 100 % of cases. The attack is detected by an establishment of the absence of any signal from the corresponding pin of the microcontroller
3	An attack on pins of the device or its communi-cation interface	–	Detected	The physical on-the-fly sensor replacement was detected by a short-term deviation (actually some unpredictable splash of the electro signal) of recorded sensor's values at the time of the physical switch between the real sensor and the fake one. Thus, such attack was successfully detected n the framework of the experiments
				However, we should mean a possibility of a potential intruder to build a more smart fake programmable sensor connected in parallel with the genuine one to exert gradually increasing influence on the pins of the device. Such advances sensor would make minimal side effects therefore it would be difficult to detect
4	An attack on the anomaly detector software component	High dependency of the attack on the monitoring module structure, its particular implementation and the used software/hardware platform	Not detected	Generally the anomalous data monitoring does not assume checking integrity and operability of the monitoring module itself
				To detect an attack on modification of the software code of the monitoring module a secure container, which the monitoring module will be carried out in can be used
				Forced blocking of the monitoring functions should be caught by software attestation mechanisms

not within the conducted monitoring experiments on the developed smart home prototype. The last column presents the findings on the applicability of the offered protection in practice and possible need for its further improvements.

8 Conclusion

The paper contributed a technique for design, verification and testing for IoT systems. The technique is targeted on construction and evaluation of IoT embedded protection means. The technique is based analysis security components, both their specifications and runtime models and implementations, hidden inconsistencies in the specifications and incorrect data. The technique utilizes a conducted analysis of three industrial IoT use cases to extract specific expert knowledge it is based on. The technique could be applied to a wide range of telecommunication systems including various mobile and embedded devices as well as cloud-based solutions. As a new features of the technique one can single out exploitation of ad-hoc heuristic embedded security data directly connected to design, verification and testing issues. In the forthcoming research we are going to move towards application of the concept of security patterns in order to transform the obtained expert knowledge to more flexible software/hardware embedded security solutions to be naturally used by system engineers by various existing mobile and embedded platforms. Besides it is planned to identify knowledge to develop test suites for IoT-systems using fuzzy-testing.

Acknowledgments This research is being supported by the Grants of The Ministry of Education and Science of The Russian Federation (contract # 14.604.21.0147, unique contract identifier RFMEFI60414X0147).

References

Abraham DG, Dolan GM, Double GP, Stevens JV (1991) Transaction security system. IBM Syst J 30(2):206–228

Agaskar A, He T, Tong L (2010) Distributed detection of multi-hop information flows with fusion capacity constraints. Signal Process IEEE Trans 58(6):3373–3383

Arbaugh WA, van Doorn L (2001) Embedded security: challenges and concerns. Comput J 34(10):40–41

Braghin C, Sharygina N, Barone-Adesi K (2011) A model checking-based approach for security policy verification of mobile systems. Form Asp Comput 23(5):627–648

Burleson W, Clark SS, Ransford B, Fu K (2012) Design challenges for secure implantable medical devices. In: Design Automation Conference (DAC), 49th ACM/EDAC/IEEE, pp 12–17

Chechulin A, Kotenko I, Desnitsky V (2012) An approach for network information flow analysis for systems of embedded components. LNCS 7531:146–155

Desnitsky V, Kotenko I (2014) Expert knowledge based design and verification of secure systems with embedded devices. Lecture notes in computer science (LNCS), vol 8708. Springer, Cham, pp 194–210

Desnitsky V, Kotenko I, Chechulin A (2012) Configuration-based approach to embedded device security. LNCS 7531:270–285

Desnitsky V, Kotenko I, Nogin S (2015) Detection of anomalies in data for monitoring of security components in the internet of things. In: XVIII international conference on soft computing and measurements (SCM'2015). IEEE Xplore

Dick N, McCallum N (2004) High-speed security embedded security. Commun Eng J 2(2):37–39

Henzinger TA, Sifakis J (2006) The embedded systems design challenge. LNCS, vol 4085. Springer, Berlin Heidelberg, pp 1–15

Hwang DD, Schaumont P, Tiri K, Verbauwhede I (2006) Securing embedded systems. IEEE Educ Act Dep IEEE Secur Priv 4(2):40–49

http://www.isasecure.org. Accessed 4 April 2016

Juengst WE, Heinrich M (1998) Using resource balancing to configure modular systems. Intell Syst Appl IEEE Comput Soc 13(4):50–58

Knezevic M, Rozic V, Verbauwhede I (2009) Design methods for embedded security. Telfor J 1(2):69–72

Kocher P, Lee R, Mcgraw G, Ravi S (2004) Security as a new dimension in embedded system design. In: Proceedings of the 41st design automation conference (DAC'04), pp 753–760

Kommerling O, Kuhn MG (1999) Design principles for tamper-resistant smartcard processors. In: Proceedings of the USENIX workshop on smartcard technology, pp 9–20

Koopman P (2004) Embedded system security. IEEE Comput 37(7):95-97

Kotenko I, Polubelova O (2011) Verification of security policy filtering rules by model checking. In: Proceedings of IEEE fourth international workshop on "intelligent data acquisition and advanced computing systems: technology and applications" (IDAACS'2011), pp 706–710

http://www.nomagic.com. Accessed 4 April 2016

McComb T, Wildman L (2006) User guide for SIFA v.1.0. Technical report

http://www.modelio.org. Accessed 4 April 2016

Moyers BR, Dunning JP, Marchany RC, Tron JG (2010) Effects of wi-fi and bluetooth battery exhaustion attacks on mobile devices. In: Proceedings of the 43rd Hawaii international conference on system sciences (HICSS'10), IEEE Computer Society, pp 1–9

MARTE. Object Management Group (2011) The UML profile for MARTE: modeling and analysis of real-time and embedded systems, Version 1.1

Pistoia M, Chandra S, Fink S, Yahav E (2007) A survey of static analysis methods for identifying security vulnerabilities in software systems. IBM Syst J 46:265–288

Potlapally N (2011) Topics in secure embedded system design. A Dissertation presented to the Faculty of Princeton University in Candidacy for the Degree of Doctor of Philosophy by the Department of Electrical Engineering, Published 2011.10.19 by ProQuest, UMI Dissertation Publishing, ISBN:1244946192, Paperback 86 pages

Rae A, Fidge C (2005) Identifying critical components during information security evaluations. J Res Pract Inf Technol 37:391–402

Rae AJ, Wildman LP (2003) A taxonomy of attacks on secure devices. In: Australian information warfare and IT security, 20–21 November 2003, Australia, pp 251–264

Ravi S, Raghunathan A, Kocher P, Hattangady S (2004) Security in embedded systems: design challenges. ACM Trans Embed Comput Syst 3(3):461–491

Ruiz JF, Harjani R, Maña A, Desnitsky V, Kotenko I, Chechulin A (2012) A methodology for the analysis and modeling of security threats and attacks for systems of embedded components. In: Proceedings of the 20th Euromicro international conference on parallel, distributed and network-based computing (PDP2012). Munich, Germany, February 15–17

Ruiz JF, Rein A, Arjona M, Mana A, Monsifrot A, Morvan M (2012) Security engineering and modelling of set-top boxes. In: Proceedings of biomedical computing (BioMedCom), 2012 ASE/IEEE international conference, pp 113–122

Sabin D, Weigel R (1998) Product configuration frameworks-a survey. Intell Syst Appl IEEE Comput Soc 13(4):42–49

SecFutur. Design of Secure and energy-efficient embedded systems for Future internet applications, FP7 Project Web site, http://www.secfutur.eu. Accessed 4 April 2016

Schumacher M, Fernandez-Buglioni E, Hybertson D, Buschmann F, Sommerlad P (2006) Security patterns: integrating security and systems engineering. Wiley, Hoboken

Sprintson A, El Rouayheb S, Georghiades C (2009) A new construction method for networks from matroids. In: Proceedings of the 2009 symposium on information theory (ISIT'09)

Trusted Platform Module. http://www.trustedcomputinggroup.org/resources/tpm_main_specification. Accessed 4 April 2016

http://www.altova.com/umodel.html. Accessed 4 April 2016

Wang Z, Johnson R, Murmuria R, Stavrou A (2012) Exposing security risks for commercial mobile devices. Comput Netw Secur LNCS 7531:3–21

Wei G, Qin Y (2009) An approach of product configuration based on decision tree and minimum conflicts repair algorithm. In: Proceedings of the International Conference on Information Management, Innovation Management and Industrial Engineering (ICII '09), vol 1, pp 126–129

Yu B, Skovgaard HJ (1998) A configuration tool to increase product competitiveness. IEEE Intell Syst 4:34–41