Journal of Universal Computer Science, vol. 22, no. 9 (2016), 1249-1273
submitted: 12/7/16, accepted: 30/8/16, appeared: 1/9/16

Secure Control Applications in Smart Homes and Buildings

Friedrich Praus
(University of Applied Sciences Technikum Wien, Austria
praus@technikum-wien.at)

Wolfgang Kastner
(TU Wien, Automation Systems Group, Austria
k@auto.tuwien.ac.at)

Peter Palensky
(TU Delft, Intelligent Electrical Power Grids, The Netherlands
P.Palensky@tudelft.nl)

Abstract: With today's ongoing integration of heterogeneous building automation systems, increased comfort, energy efficiency, improved building management, sustainability as well as advanced applications such as active & assisted living scenarios become possible. These smart homes and buildings are implemented as decentralized systems, where embedded devices are connected via networks to exchange their data.

Obviously, the demands – especially regarding security – increase: Secure communication becomes equally important as secure software being executed on the embedded devices. While the former has been addressed by standardization committees, manufacturers and researchers, until now the problem of secure control applications in this domain has not been addressed extensively. This leads to insecure and unprotected software being executed on the embedded devices. Thus, adversaries are capable of attacking building automation systems.

This paper introduces an architecture for distributed control applications in smart homes and buildings, which tackles the problem on how to secure software running on different device classes. The following novelties are contributed: an application model capable of depicting control applications in a formal way, the concept of security attributes, being able to formally specify a security policy, and a framework, which allows the secure development and execution of control applications, and an enforcement of the defined security policies.

Key Words: Secure Software, Secure Control Applications, Security, Building Automation, Smart Homes
Category: D.4.6, K.6.5, C.2.0

1 Introduction and Motivation

In order to provide secure Building Automation Systems (BASs), comprehensive measures need to cover communication as well as device security. Mechanisms tailored to the use in Building Automation Networks (BANs) that counteract communication and network attacks are presented in [Granzer, 2010]. An overall device security needs to deal with software, side-channel, and physical attacks. An extensive survey on the latter two and a short discussion of countermeasures can be found in [Koeune and Standaert, 2004].

This paper focuses on software security for smart home and building devices since the current state of the art lacks adequate security mechanisms. Irrespective of the used technology, no sound protection against software attacks is deployed, thus enabling adversaries to successfully attack those devices. Existing protection techniques from other domains (e.g. the Information Technology (IT) domain) are insufficient and not applicable to BASs due to different functional and non-functional requirements.

The peculiarities in today's BAS rely on its structure consisting of a control network and a common backbone which together form the BAN. Sensor Actuator and Controllers (SACs) are located at the control level. Representatives of this device class interact directly with the physical environment and are responsible for data acquisition and for controlling the behavior of the environment. InterConnection Devices (ICDs) provide an interconnection between network segments or remote access to foreign networks. Management Devices (MDs) are used to configure and maintain a BAS. While the functionality of system components (i.e. ICDs and MDs) is usually fixed, SACs are highly customizable and manifold. Thus, in the BAS domain typically the approach exists to customize generic "template" network nodes with application specific hardware. Universally designed base platforms consisting of MicroController Units (MCUs) and network interfaces are used in conjunction with application specific components (e.g. switches, temperature sensors) to form a particular system. Similarly, the software is split into a generic Operating System (OS) or system software providing basic functionality and a customizable Control Application (CA) dealing with the specific hardware. While the former is usually fixed and non replaceable, the latter is implemented by the device manufacturer and may be downloaded by the system integrator, even after installation of the device. Thus, a CA is a configurable software being executed on a SAC with the purpose to control a process at the control level. Distributed CAs communicate via the BAN and implement a particular function of a BAS. A common way to model distributed CAs and their application models is with the help of Function Blocks (FBs). Sensor functions convert physical quantities to output information which in turn serves as input to application or actuator functions. Actuator functions convert input information obtained through the BAN to physical quantities. Application functions represent the functionality to be achieved by means of automation and control.

The paper is structured as follows. Section 2 covers CA security in current building automation standards and technologies, a software security threat and risk analysis and summarizes security requirements. Besides, existing software protection techniques are briefly investigated and evaluated for BASs with respect to their applicability and implied security gain. These sections are heavily based on [Praus et al., 2016]. The major contribution of this paper is contained in Section 3, which covers a comprehensive concept and process for secure and distributed CAs, describing how to fulfill the demands for security-critical smart homes and buildings. Section 4 describes the implementation and evaluation. Finally, future work is described in Section 5. The work behind this paper is based on the dissertation of the author [Praus, 2015].

2 Control Application Security and Software Protection Techniques

The ultimate goal of an adversary is to gain unauthorized access to control level functions by manipulating the software being executed on BAS devices. Such attacks can either be performed remotely via the network or locally, exploiting threats in a device's interface. First, an adversary may directly access SACs to manipulate the behavior of the hosted CAs by changing configuration parameters (e.g. setpoint), the control logic (e.g. algorithm), or the control data (e.g. output value). Second, an adversary may attack the application running on the ICD to get access to the data passing through the ICD. As ICDs may also provide an interconnection to foreign public networks (e.g. the Internet), an ICD can also be misused as access point to launch further attacks via the BAN. Finally, an adversary may attack a MD by manipulating the operator software and also impersonate a MD. The privileges of the compromised device can then be misused to gain management access to SACs or ICDs.

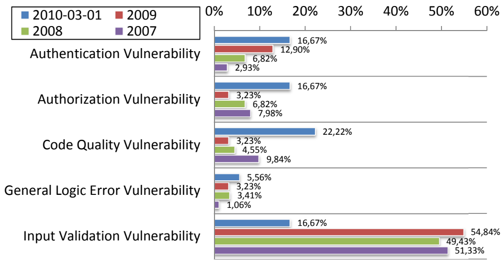
To identify the security threats in current installations, [Praus, 2015] targeted the question, whether and how many BASs based on BACnet and KNX are openly connected to the Internet and which security measures are currently implemented. A worldwide IPv4 address range scan for BACnet/IP and KNXnet/IP services has been carried out in 2014 by the author. A total of 17.259 vulnerable BAS installations have been detected (cf. Figure 1a). The installations ranged from business parks and towers, high schools, shopping plazas, water pollution control stations, fire stations, churches to smart homes with control of private saunas.

To be able to provide secure CAs in BASs, it is first necessary to identify the threats to CA software and analyze possible vulnerabilities. The Open Web Application Security Project Top 10[1] provides such an analysis covering over 500,000 vulnerabilities tailored to web application security. Although being slightly different from the security of CAs, a categorization can nevertheless be

[1] `http://www.owasp.org/`, Last access: 2016/02/02

Country	BACnet	Country	KNX
US	8989	Germany	627
Canada	2296	Netherlands	522
Finland	282	Spain	332
Australia	271	France	244
Spain	231	Austria	220
Total	13964	Total	3295

(a) Vulnerable BACnet and KNX Installations



(b) Vulnerabilities Breakdown 2010-03 – 2007

Figure 1: Control Application Security (Top 5 Countries and Categories) [Praus et al., 2016]

derived, if it is assumed similar security flaws are present. Based on this categorization, an analysis of the commonness of vulnerability types can be performed. Figure 1b shows a breakdown of the top 5 vulnerabilities being openly available at the US-CERT Vulnerability Notes Database[2]. All entries of the years 2007 to March 2010 (632 in total) have been analyzed, categorized and counted. Although being rather outdated, the numbers still give an adequate overview of the commonness of vulnerability types.

Due to the extreme broadness of threats and vulnerabilities to CAs, the software attack model is defined as follows: Any (malicious) CA, irrelevant whether it originates from trusted or non-trusted sources, being run on BAS devices may exploit weaknesses in security schemes and system implementations, intentionally or unintentionally. Accidental programming flaws in CAs may be present just like software being intentionally infected by trojans. Adversaries may use these manifold possibilities to access control level functions they usually are not allowed to.

Based on this attack model, security requirements dedicated to CAs are formulated (cf. [Praus et al., 2016]). They are derived out of security research in e.g. industrial communication systems [Dzung et al., 2005], embedded systems [Ravi et al., 2004] or cyber-physical systems.

Functional Requirements (FRs) are directly related to the security considerations for CAs. The utmost requirement is to prevent software attacks on CAs and, if not possible, at least detect those attacks. The following FRs can be derived to achieve this goal: *FR–memory access* (memory access must be controlled), *FR–low level functionality access* (limit actions and allowed operations a CA can perform), *FR–protection of environment* (CAs must not destruct hardware or waste resources), *FR–communication relationship* (CAs have a defined (static) communication relationship), *FR–availability* (Denial of Service (DoS) attacks need to be handled).

[2] `http://www.kb.cert.org/vuls`, Last access: 2010/04/03

Organizational Requirements (ORs) cover the special environmental conditions required for developing secure CAs in BAS: *OR–limited resources* (security overhead needs to be small), *OR–development* (CA development has to be simple and secure by design), *OR–high level language support* (high-level programming languages need to be supported), *OR–long lifetime* (BASs have to be kept operable for decades), *OR–scalability* (scalability of security mechanisms is essential), *OR–network technology* (mechanisms need to be geared towards the different requirements in BANs), *OR–compatibility* (integration of a security extension into an established BAS is preferable), *OR–physical access* (detect physical access to devices and networks), *OR–usability* (provide usability).

The ideal software protection technique allows to fully prevent vulnerabilities and hinders attacks to SACs, ICDs and MDs, no matter whether the attack pattern is already known or not. To minimize performance overhead, it is applied only during compile time, or at least does not have any performance overhead during runtime. Besides, it does not require updates and scales well. Considering today's available methods, it is however hardly possible to fulfill all these requirements at the same time due to e.g. limited system resources.

Software assisted protection techniques can be split into static and dynamic methods.

Static methods (e.g. Static Code Analysis (SCA) [Chess and McGraw, 2004], Code-Signing (CS) [Pfleeger and Pfleeger, 2002], Proof-Carrying Code (PCC) [Necula and Lee, 1998], Watermarking (WM) [Collberg and Thomborson, 2002]) are applied during compile or development time, respectively. Hence, they can prevent attacks at a point in time, where appropriate countermeasures or bug fixes can be applied without interfering with running software but for typical programming languages fundamental questions are undecidable [Landi, 1992]. *Dynamic methods* (e.g. Signature based Intrusion Detection System (SIDS) [McHugh, 2001], Anomaly based Intrusion Detection System (AIDS) [Hofmeyr et al., 1998], Software Monitoring Techniques (SMT) [Goldberg et al., 1996], Self Checking Code (SCC) [Aucsmith, 1996], Attack Specific Countermeasure (ASC) [Wilander and Kamkar, 2003], Sandbox (SB) [Tanenbaum and van Steen, 2002], OS [Science Applications International Corporation, 2008]) implicate a larger performance overhead than static ones, since additional processing has to be performed during runtime. In addition, special care has to be taken, that they are not bypassed by an adversary. *Hardware assisted methods* (e.g. Co-Processor (CP) [Arora et al., 2006], Physical Partitioning (PP) [Hiroaki et al., 2007], Harvard Architecture (HA) [Riley et al., 2007], CPU EXtension (CPUEX) [Suh et al., 2004]) use dedicated hardware for security checks. *Human assisted methods* (Inspection And Certification (IAC) [Holzmann, 2006], Formal Verification (FV) [Kinder et al., 2010]) rely on human security expertise during CA development. *Hybrid methods* (e.g. [Sekar et al., 2003]) try to combine the advantages of different software protection techniques. Thus, they can provide more powerful protection and overcome limitations of the software, hardware and human assisted methods mentioned before.

−: not applicable, ∼: applicable with restrictions, +: applicable
p: prevent, *d*: detect

	method	*FR–memory access*	*FR–low level functionality access*	*FR–protection of environment*	*FR–communication relationship*	*FR–availability*	*OR–limited resources*	*OR–development*	*OR–high level language support*	*OR–long lifetime*	*OR–scalability*	*OR–network technology*	*OR–compatibility*	*OR–physical access*	*OR–usability*	SAC	ICD	MD
static software methods	SCA	−	−	∼	−	−	+	∼	+	−	−	+	∼	−	−	∼	+	−
	CS	−	−	−	−	−	+	∼	+	+	+	+	∼	−	−	+	+	+
	WM	−	−	−	−	−	+	−	+	−	+	+	∼	−	−	+	+	+
	PCC	−	−	∼	−	−	+	−	+	+	−	+	∼	−	−	−	∼	∼
dynamic software methods	SIDS	−	−	*d*	*d*	−	∼	∼	+	−	−	+	+	−	−	+	+	∼
	AIDS	−	−	*d*	*d*	*d*	−	+	+	+	+	+	+	+	∼	+	+	∼
	SMT	−	−	∼	−	−	∼	−	+	−	−	+	∼	−	−	∼	+	−
	SB	*p*	*p*	*p*	*p*	−	∼	+	+	+	+	+	∼	−	+	+	∼	+
	SCC	−	−	−	−	−	∼	−	+	−	−	∼	∼	−	−	−	+	−
	ASC	−	−	−	−	−	∼	−	+	−	−	+	∼	−	∼	∼	+	+
	OS	*p*	*p*	*p*	*p*	−	−	+	+	+	+	+	∼	−	+	−	−	+
hardware supported	CP	−	∼	−	∼	−	+	−	+	−	−	+	∼	+	−	−	−	+
	PP	*p*	*p*	−	−	−	+	+	+	−	−	+	∼	+	−	−	−	+
	HA	−	−	−	−	−	+	+	+	−	+	+	∼	−	+	+	+	+
	CPUEX	−	−	−	−	−	+	+	−	−	+	+	∼	−	+	+	+	+
human	IAC	∼	−	*p*	−	−	+	−	+	−	−	+	∼	−	−	−	∼	−
	FV	−	−	*p*	−	−	+	−	+	−	−	+	∼	−	−	−	∼	−

Table 1: Comparison of Software Protection Techniques with Respect to Security Requirements and Applicability to Sensors, Actuators and Controller Devices, Interconnection Devices, and Management Devices [Praus et al., 2016]

Until now however, no reliable and secure approach for BASs is available. It is not clear, which combinations of software protection techniques seem reasonable and fulfill the security requirements. Summarizing, the presented software protection techniques have several aspects in common, which hinder their seamless use in BASs.

– First, they are not able to offer full protection against the discussed threats discussed. A hybrid approach, however, seems promising to provide an overall security.

– Second, they are designed for general purpose and do not cover the specialties concerning security for BASs, SACs and their CAs, ICDs or MDs in any way.

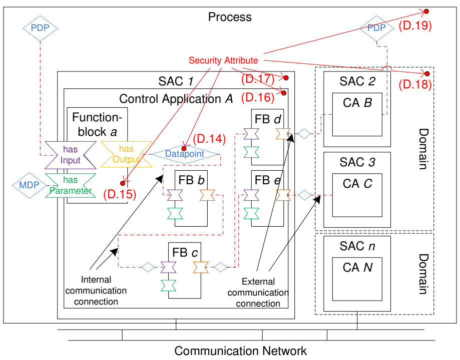
A detailed discussion on the current state of the art is presented in [Praus et al., 2016]. The summary is shown in Table 1 to demonstrate that current methods do not provide an extensive protection with respect to the requirements in BAS.

3 Secure Control Application Architecture

A secure architecture being adaptable to all common BAS standards needs to cover BAS specific constraints and be capable of detecting possible attacks. Thus, only hybrid software protection mechanisms can provide an overall CA security.

3.1 Generic Application Model

The first step towards secure CAs within the heterogeneous open BAS standards is to define distributed applications in a general and abstract way and provide a unified system view. This is achieved by the definition of an abstract model (e.g. the abstract BAS device description) and concrete instances therefrom (e.g. a BAS device instance representing a particular technology with specific parameters). Additionally, protocol-specific and domain-specific knowledge (e.g. BAS specific vocabulary, security attributes) are part of the model. All configuration and management tasks and definition of a security policy can now be performed directly on the abstracted representation and be automatically distributed to the different underlying technologies.



$$\begin{aligned}
\text{DP}: &\quad < p, \ldots > && \text{(D.1)}\\
\text{FB}: &\quad \{dp_i | dp_i \in \text{DP}, i \in \mathbb{N}\} && \text{(D.2)}\\
\text{CA}: &\quad \{fb_i | fb_i \in \text{FB}, i \in \mathbb{N}\} && \text{(D.3)}\\
\text{SAC}: &\quad \{\text{CA}\} && \text{(D.4)}\\
DOMAIN: &\quad \{sac_i | sac_i \in \text{SAC}, i \in \mathbb{N}\} && \text{(D.5)}\\
PROCESS: &\quad \{domain_i | domain_i \in DOMAIN, i \in \mathbb{N}\} && \text{(D.6)}
\end{aligned}$$

Figure 2: Generic Application Model [Praus, 2015]

This generic application model is expressive enough to be able to model all different types of distributed CAs typically found in the building automation domain. The nomenclature is based on the one used in IEC 61499 [IEC, 2011], but modifications were made with respect to the building automation domain vocabulary. It is specified using a formal way accompanied by a textual representation.

The application model [Praus and Kastner, 2010] (cf. Figure 2) consists of SACs which are linked by a communication network and interact with a process (i.e. a building) under control. The BAS controls this process which can be split into various control domains (cf. Definition (D.6)). A control domain can logically be seen as a grouping of distributed BAS nodes (i.e. SACs), which realize a common functionality (e.g. Heating, cooling, Ventilation, and Air Conditioning (HVAC), lighting). It consists of at least one SAC, which itself may arbitrarily belong to multiple other domains and which interacts with the environment (cf. Definition (D.5)). Each SAC hosts exactly one CA[3] (cf. Definition (D.4)). CAs implement at least one FB, which contributes its particular part to functionality of the CA (cf. Definition (D.3)). A FB is a data structure with algorithms, internal variables and an arbitrary combination of the binary input and output relations *hasInput*, *hasParameter* and *hasOutput*. These relations link a single FB to a single DP (cf. Definition (D.2)). A communication connection between FBs, however, is established, if two different relations link to exactly the same DP. This sometimes is also referred to as binding and can e.g. be seen between FB a and FB b in Figure 2. Physical or Process Datapoints (PDPs), Management Datapoints (MDPs) as well as DPs of external communication connections are located outside any particular SAC for modeling. It is clear, that PDPs and MDPs are assigned to their corresponding FBs when implementing a SAC and processing (e.g. memory allocation) takes place there. Likewise, a DP of an external communication connection needs to be split since on the one hand the sending node needs to process it and on the other hand the receiving node as well. A machine readable listing of the generic application model can be found in [Praus, 2015].

3.2 Software Security Policy

The second step is to define a security policy, which states whether the condition of a BAS is security critical and violates some defined constraints or not. For executing this policy it can be split down to involved present values p_i of DPs DP_i, where security requirements for the conditions derived from the policy can be defined, formulated and finally evaluated.

The generic application model is enriched with so called security attributes. A security attribute s is a tuple consisting of a present value p, conditions of (possibly other) present values $cond_1 \dots cond_n$ and boolean operations $\bullet_1 \dots \bullet_{n-1}$

[3] Since SACs typically are low end embedded nodes with limited resources, they are not considered to be multiprocess capable.

which relate to these conditions:

$$security\_attribute\ s :< p, cond_1, \ldots, cond_n, \bullet_1, \ldots, \bullet_{n-1} > \tag{D.7}$$

A condition is formulated using a function on a present value at some instant $f(p_\alpha(t))$, a function on a second present value at some point in time $g(p_\beta(t))$ and a third function $\circ$ relating them (A, B, C being the value range of present values, $f, g, \circ$ targeting the relation of the present values).

$$\begin{aligned} &cond(f(p_\alpha(t)), g(p_\beta(t)), \circ) \mid \\ &\qquad p_\alpha(t) \in A, p_\beta(t) \in B, \\ &\qquad f : A \to C, g : B \to C, \\ &\qquad \circ : f \times g \to \{true, false\} \end{aligned} \tag{D.8}$$

Note, that for many use cases only the current present value is relevant and so $p(t)$ reduces to p. It is further defined:

$$\begin{aligned} &cond(check\_access(p_\alpha(t)), \{read\}, =) \\ &\quad = check\_access(p_\alpha(t)) = \{read\} \\ &\quad = \begin{cases} true \text{ if } p_\alpha(t) \text{ is readable} \\ false \text{ otherwise} \end{cases} \\ &cond(check\_access(p_\alpha(t)), \{write\}, =) \\ &\quad = check\_access(p_\alpha(t)) = \{write\} \\ &\quad = \begin{cases} true \text{ if } p_\alpha(t) \text{ is writeable} \\ false \text{ otherwise} \end{cases} \end{aligned} \tag{D.9}$$

$$\begin{aligned} &cond(check\_value(p_\alpha(t)), g(p_\beta(t)), \circ) \\ &\quad = \circ(p_\alpha(t), g(p_\beta(t))) \end{aligned} \tag{D.10}$$

$$\begin{aligned} &cond(check\_history(p_\alpha(t)), g(p_\beta(t)), \circ) \\ &\quad = \circ(p'_\alpha(t), g(p'_\beta(t))) \end{aligned} \tag{D.11}$$

$$\begin{aligned} &cond(check\_accesstime(p_\alpha(t)), g(p_\beta(t)), \circ) \\ &\quad = \circ(check\_accesstime(p_\alpha(t)), constant) \\ &\quad = \begin{cases} true \text{ if } check\_accesstime(p_\alpha(t)) \circ constant \\ false \text{ otherwise} \end{cases} \end{aligned} \tag{D.12}$$

The condition of p can thus be controlled with respect to:

- Access rights: (D.9) enforces a basic access restriction policy but can also be used to limit the communication relationship of a CA to the desired communication partners. The required information can be automatically generated from the binding relationship present within the application model.
- Value range: (D.10) limits the values of p. To e.g. define, that p has to be larger than a minimum value, define $g(p_\beta(t)) = \{min\}$ to be constant and the relation as $\circ = \{\geq\}$. To e.g. define, that p shall be twice as large as the present value p_β define $g(p_\beta(t)) : 2 * p_\beta$ and the relation as $\circ : \{=\}$.
- Historical data: p may depend on past values and so minimum and maximum difference per time unit may be limited. (D.11)
- Time dependencies: Finally the access frequency of p may be limited. A minimum frequency declares how often a CA has to update a DP (e.g.

periodic transmission of alarm sensor values), and a maximum frequency limits the required resources (e.g. malicious exhaust of network bandwidth in case of too frequent DP updates). (D.12)

Finally, a security attribute is evaluated by checking all involved conditions:

$$eval(s) = \begin{cases} true \text{ if } cond_1, \ldots, cond_n = true \\ false \text{ otherwise} \end{cases} \tag{D.13}$$

The evaluation function returns $true$ if the present value p satisfies the policy with all conditions, and is $false$ otherwise.

Six conceptual locations can be identified, where security attributes have to be located (cf. Figure 2):

$$sec\_attr_{DP} : s \mid \{\forall p_i \in DP \wedge p_i = p, \forall i = 1, \ldots, n\} \tag{D.14}$$
$$sec\_attr_{FB} : s \mid \{\forall p_i \in \mathrm{FB}\} \tag{D.15}$$
$$sec\_attr_{CA} : s \mid \{\forall p_i \in CA\} \tag{D.16}$$
$$sec\_attr_{SAC} : \{security\_attribute_{systemcalls}\} \tag{D.17}$$
$$sec\_attr_{DOMAIN} : s \mid \{\forall p_i \in DOMAIN\} \tag{D.18}$$
$$sec\_attr_{PROCESS} : s \mid \{\forall p_i \in PROCESS\} \tag{D.19}$$

- Security attributes at the DP level cover only local constraints of the present value p. (D.14)
- A FB may consist of multiple relations to DPs. Security relevant dependencies between these entities can be modeled and expressed within security attributes. The security of a present value p_1 of DP_1 can be evaluated under the condition of the present value p_2 of a second DP DP_2. (D.15)
- Similar to dependencies between relations within a FB, dependencies between DPs DP_1 and DP_2 of different FBs within a CA can be modeled and expressed in security attributes. Consider a single-room control CA, which is used to cool or heat a room. A security attribute can be formulated which prohibits the simultaneous activation of the cooling function via DP_1 with p_1 and the heating appliance via DP_2 with p_2. (D.16)
- Since only a single CA is executed on SACs, no dependencies between CAs are formulated. Assuming more powerful devices being able to execute CAs in parallel, traditional OS (security) mechanisms have to be applied (e.g. memory protection, interprocess communication). This is, however, not in the scope of this work. Nevertheless security attributes can be engaged at device level to provide additional protection for e.g. the device's hardware. No DP is associated with these attributes, they can rather be seen as limits to system calls within the CA software. Such attributes may e.g. prevent a wearout of the devices flash memory or a flooding of its storage space due to a malicious CA. (D.17)
- Security attributes dedicated to dependencies between SACs can be attached to the control domain. As an example, consider the security system domain,

when a central close door locks functionality is requested. It has to be guaranteed, that all involved doors really close their locks and an adversary does not bypass the request. (D.18)

- Finally, security attributes can be generated for the whole process. Within the building, a security system can be a typical use case. Upon detecting an alarm condition, all lights of the house shall be turned on to banish a possible pick lock. In this case, an information flow from the alarm sensor to the light actuator needs to take place. This information flow, however, must not be abused by malicious CAs. (D.19)

The mightiness of the established security attributes depends on the location where they are attached. Security attributes attached to DPs are rather limited with respect to providing overall BAS security. They simply can be used to enforce local conditions. However, the expressiveness of security attributes rises the more manifold the set of the involved present values is, opening the possibility to define global security conditions. To put it differently, the expressiveness follows the relation $DP \subseteq \mathrm{FB} \subseteq CA \subseteq SAC \subseteq DOMAIN \subseteq PROCESS$.

To be able to evaluate the presented security attributes, a concept is needed which is able to monitor and enforce them at runtime and must not be bypassed [Praus and Kastner, 2010]. The basic idea behind security attributes is that each entity is able to evaluate upon reading or writing a present value if it satisfies all conditions. It is obvious, that this entity also needs to have access to all involved present values. Process or control domain security attributes, however, may affect multiple devices or CAs, which are not necessarily linked via communication connections. Therefore, the enforcement of such global properties may not be possible for a single CA. Nevertheless, these security attributes can be used as inputs for additional security devices or mechanisms, such as Intrusion Detection Systems (IDSs) [Li et al., 2005]: Although any violation of a process or control domain security attribute can not be recognized by a single CA, it can very well be monitored by an IDS being connected to the network and having a global view of the exchanged process data. Further actions such as alarming the operator can then be taken.

3.3 Secure Software Environment

To ease the development of secure CAs, a secure software environment is needed [Praus et al., 2009]. The idea is to separate the system software running on the device from the CA as well as the node configuration (cf. Figure 3). Each component imposes an additional security barrier to the overall security and limits possible security threats.

A simple, tight and secure system software provides controlled access to system resources. It consists of various layers and intends to provide building blocks which can be mixed and matched to support different hardware configurations.

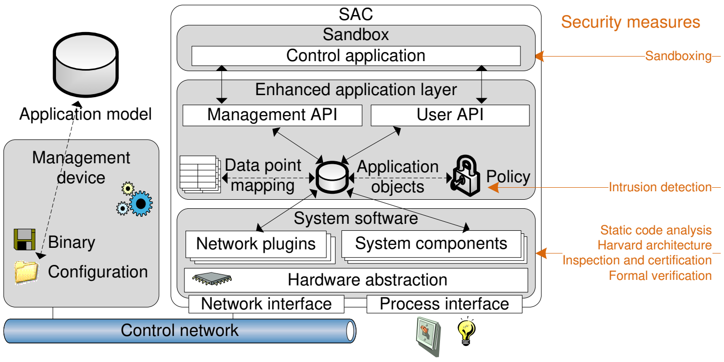


Figure 3: Architecture of Sensor Actuator and Controllers [Praus, 2015]

To access the hardware in an independent and modular way at the lowest level, a Hardware Abstraction Layer (HAL) hides the peculiarities of basic Input/Output (I/O) handling and on-chip peripherals. It allows to easily deploy the developed software to other MCU architectures allowing flexibility in design and to fulfill the differing resource requirements. (Secure) network protocol stacks can be integrated according to the requirements of a particular application, which handle the mapping of technology specific application models to the generic application objects. Further system components (e.g. for controlling peripherals) are also located on top of the HAL. Besides, the system software runs the SB and manages its required memory. To provide security, the system software is analyzed using human being based IAC, code reviews as well as SCA using automated tools. This long lasting process has to be done very thoroughly since mistakes in this stage may easily squash any later efforts in developing a secure platform. However, this (extra) effort is not for nothing since such an established common system software for a particular architecture/processor, may – once considered to be secure – serve as a common code base for SACs.

The enhanced application layer stores the application objects, their data point mappings as well as the security policy for the CA. Any network plugin or system component exclusively interacts with the CA via these shared objects. The security policy defines the normal behavior of the CA. Any abnormal behavior can be detected using an AIDS. Thus, limits to e.g. network or processing resources may be defined, which are enforced at runtime. The user Application Programming Interface (API) provides various services to access the application objects (e.g. network access, access to on-chip peripherals such as timers, process interaction) and allows to control the possibilities of a CA. The management API interfaces with a management tool, which pre-processes the binary of a CA as well as its corresponding configuration and allows access to the system software to support the total replacement and download of CAs.

A SB executes the CA. Its behavior can be monitored and controlled and mechanisms to provide memory protection can be integrated. Additionally, the execution of CAs can be limited to those being signed and containing valid cryptographic signatures to provide support for Digital Rights Management (DRM). Besides, a SB may also be designed to support the rapid development of CAs.

A MD is used to configure the parameters of the system software and enhanced application layer (even during runtime) and securely deploy the CA into the SB. Communication takes part via the well-defined management API. Since BASs typically consist of a large amount of SACs, management access to them should be possible over the BAN. The configuration is based on a generic application model, that hosts the technology specific application models (i.e. BACnet objects, KNX standardized FBs, LonWorks Standard Functional Profile Templates (SFPTs), ZigBee objects) as well as a definition of generic application objects. Besides, it provides a mapping of the technology datapoints to the generic datapoints. This way a device specific configuration can reference this knowledge base and provide the necessary additional information such as network address(es) for actually implementing a SAC. In addition, the configuration contains the security policy, which defines the normal behavior of a CA. Any abnormal behavior or attacks can thus be detected by the system software and limits to e.g. network or processing resources may be defined, which are enforced during runtime. It is important to note, that the configuration forms a major part regarding security since it allows to enforce complex security policies if properly designed. With its help additional valuable information can be supplied to the SB which otherwise could never be gained from methods described in Section 2. Consider e.g. the intended frequency of traffic a CA is about to generate on the network. A temperature sensor may once a minute want to transmit the temperature. The according configuration could thus supply exactly this information to the SB, which on its part may enforce this limit. Obviously, a CA designer has to provide reasonable values along with a CA and the user has to inspect these values. However, it is always possible for the user to deny the execution of a CA if the configuration does not fulfill the demands (e.g. supplies “limits=none”). Besides, it is possible for a wide range of SACs to define generic device profiles, which may be shared among applications of the same purpose. In such a way, the device class of sensors may share a single profile with generic limits and the CA designer does not have to provide an individual configuration which eases and speeds up the development of CAs and increases their security. In fact, standardization within the BAS domain already provides a generic view with valuable information regarding CA behavior. This information solely has to be provided to the security system.

4 Implementation and Evaluation

To validate the feasibility of the presented architecture, various orthogonal analyses have been performed. First, a use case for a distributed CA has been defined to show an implementation of the application model and security policy. Second, an accompanying discussion shows how to enable security in today's already existing BASs. Concluding, an exhaustive analysis evaluates whether the functional and organizational requirements can be fulfilled or not.

4.1 Use Case ⊗

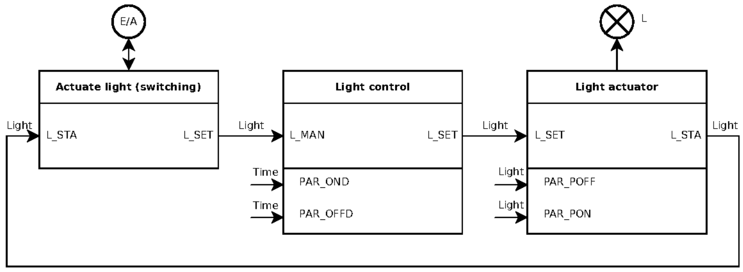
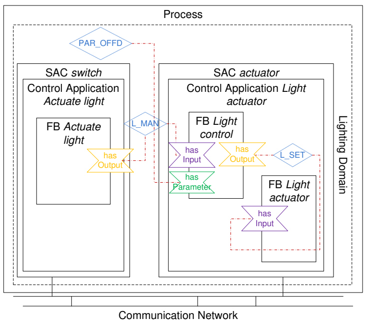


Figure 4: Use Case ⊗: Distributed Control Application: VDI 3813-3: D-1-2 Lighting Control Manual with Time-Controlled Switching Off (Stairwell Light) [Praus, 2015]

To be able to describe the security concept presented in this paper in a better way, the following use case ⊗ is defined according to VDI 3813-3 [Verein Deutscher Ingenieure, 2015]. Figure 4 shows a minimalistic light actuator with delayed on/off behavior being used for a stairwell light.

Use case ⊗ *Stairwell light: When pressing the "on" button* `E` *of the light sensor, the attached light control immediately triggers the light actuator to switch on lamp* `L`*. Upon pressing the "off" button* `E` *of the light sensor, the attached light control waits the time configured with the off delay parameter* `PAR_OFFD` *and then triggers the light actuator to switch off the lamp* `L`*. The output* `L_STA` *of the light actuator is used as feedback to the actuate light sensor, to be able to display the current status using output* `A`*.*

Figure 5 shows an example implementation of the generic application model for use case ⊗. Its formal specification is shown in Definition D.20. A process of control domain lighting is defined. Two SACs (`SAC actuator`, `SAC switch`) are deployed, each of them implementing a CA. The `CA Light actuator` actually implements the function blocks `FB Light actuator` and `FB Light control`. It is thus responsible for switching the physical output according to the present value contained in `MDP PAR_OFFD` and the present value contained in `PDP L_MAN`. The `CA Actuate light` is responsible for reading a physical switch and setting the `PDP L_MAN` appropriately.



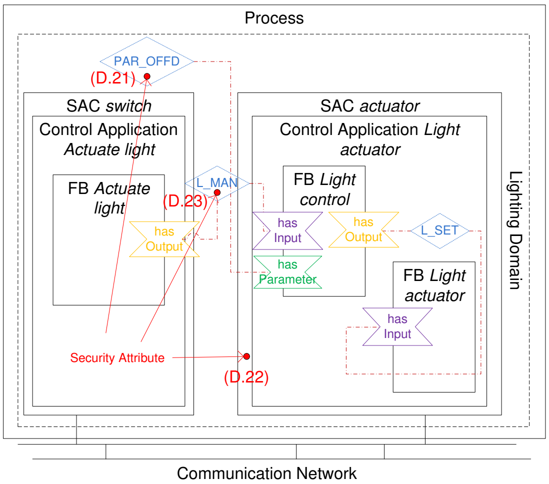
$$\begin{aligned}
\text{MDP } PAR_OFFD: &\quad < p, \ldots > \qquad\qquad\qquad\qquad (\text{D.20})\\
\text{PDP } L_MAN: &\quad < p, \ldots >\\
\text{PDP } L_SET: &\quad < p, \ldots >\\
\text{FB } Actuate\ light: &\quad \{\text{DP } L_MAN\}\\
\text{FB } Light\ control: &\quad \{\text{DP } PAR_OFFD,\\
&\quad \text{DP } L_MAN,\ \text{DP } L_SET\}\\
\text{FB } Light\ actuator: &\quad \{\text{DP } L_SET\}\\
\text{CA } Actuate\ light: &\quad \{\text{FB } Actuate\ light\}\\
\text{CA } Light\ actuator: &\quad \{\text{FB } Light\ control,\\
&\quad \text{FB } Light\ actuator\}\\
\text{SAC } switch: &\quad \{\text{CA } Actuate\ light\}\\
\text{SAC } actuator: &\quad \{\text{CA } Light\ actuator\}\\
LIGHTING\ DOMAIN: &\quad \{\text{SAC } switch,\ \text{SAC } actuator\}\\
PROCESS: &\quad \{Lighting\}
\end{aligned}$$

Figure 5: Generic Application Model: Example for Use Case $\otimes$ [Praus, 2015]

Figure 6 shows an example a software security policy again for use case $\otimes$. The formal specification is illustrated in Definitions D.21–D.23. A security attribute on DP level is shown in Definition D.21. The present value of the MDP needs to be within a certain range. Thus, it needs to be greater than a defined minimum and smaller than a defined maximum. Definition D.22 describes a security attribute on SAC level. To prevent damage, the physical output of the SAC must not exceed a maximum switching frequency. Finally, a security attribute on domain level is shown in Definition D.23. The present value of PDP `L_MAN` shared between two SACs needs to be zero or one. Since `L_SET` is a DP of an internal communication connection, no security attribute needs to be formulated.

4.2 Enabling Security in Existing Installations

To validate that the presented CA architecture can be used to enhance security in current technologies and installations, additional case studies demonstrate that such attacks can be prevented or at least be detected. The case studies are selected based on the following criteria: Case study 1 covers BACnet/IP and KNXnet/IP since more than 17.000 installations are connected to the Internet unprotectedly. Case study 2 and Case study 4 deal with today's most common



$$s_{DP\ PAR\_OFFD} :< p_{PAR\_OFFD}, cond1_{p_{PAR\_OFFD}}, cond2_{p_{PAR\_OFFD}}, AND > \tag{D.21}$$
$$eval(p_{PAR\_OFFD}) \rightarrow cond1_{p_{PAR\_OFFD}}\ AND\ cond2_{p_{PAR\_OFFD}}$$
$$cond1_{p_{PAR\_OFFD}} : cond(p_{PAR\_OFFD}, maximum, \leq)$$
$$cond2_{p_{PAR\_OFFD}} : cond(p_{PAR\_OFFD}, minimum, \geq)$$
$$s_{SAC\ actuator} :< systemcall\_switchoutput, check\_accesstime(systemcall\_switchoutput, maximum\_frequency, \leq) > \tag{D.22}$$
$$s_{Lighting\ DOMAIN} :< p_{L\_MAN}, cond_{p_{L\_MAN}} > \tag{D.23}$$
$$cond_{p_{L\_MAN}} : cond(p_{L\_MAN}, \{0,1\}, =)$$

Figure 6: Software Security Policy for Use Case ⊗ [Praus, 2015]

software vulnerabilities (cf. Figure 1b). Case study 3 has been selected, since KNX seems to be one of the most spread technologies today[4].

Case study 1 *Prevent attacks on current installations*

The analysis in Section 2 relied on the fact, that SACs have been directly connected to the Internet and no secure ICDs have been deployed, which check the communication relationships. In fact, attack prevention is easy, since only a single ICD with firewall needs to be installed at the interconnection point between the Internet and the BAN. Also, attack detection is possible, if an IDS is installed in the BAN. The required security policy can simply be gained out of the Engineering Tool Software (ETS) for KNX based BAS or the corresponding configuration tool for BACnet based BAS.

[4] `http://knx.org/knx-en/knx/technology/introduction/index.php`, Last access: 2015/09/29

Case study 2 *Prevent attacks on KNX based SACs*

```
1  void main (void) {
2   // do some initializations
3   AppInit();
4   // declare BYTE buffer of size 1
5   BYTE buf[1];
6   // set port to output
7   PM0=0xFF;
8   // start the main loop of the application program
9   for(;;) {
10   // if pdp L_MAN (id 0) has changed, copy 2 bytes into buf
11   if(U._TestAndCopyObject
12           (0, (void*) buf, 2)) {
13    // if it is set to on
14    if (buf[0] == 0x01) {
15     // immediatley swich lamp 0 on
16     P0_bit.no0 = 1;
17    }
18   }
19   // handle switch off light after PAR_OFFD
20  }
21 }
```

Listing 1: Implementation of Use Case $\otimes$ on a KNX Bus Interface Module-M130 [Praus, 2015]

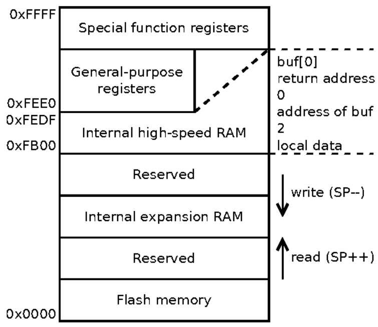


Figure 7: Memory Map (μPD78F0534) and Stack [Praus, 2015]

Listing 1 shows an implementation of the SAC light actuator of use case $\otimes$ on a standard KNX Bus Interface Module (BIM)-M130 using a μPD78F0534 MCU. An *Input validation vulnerability* is present: Line 5 declares a byte array `buf` of size 1 byte. The `U._TestAndCopyObject` function in line 11 tests, if there was an update for object `0`. If this is the case, 2 bytes of the object value are copied to the byte array `buf`. A buffer overflow happens, since 2 bytes are copied into an array, which only is of size one byte. Figure 7 shows what is happening in the memory. The internal high-speed RAM starts at address 0xFEDF right below the address space of the general-purpose registers. It contains the data such as the byte array `buf` as well as the stack. Before calling the `U._TestAndCopyObject` function the return address is pushed on the stack, which grows down towards the lower addresses. Likewise the arguments (i.e. `0, address of buf, 2`) are pushed onto the stack. Then the function is called, which copies 2 bytes of data into the array. Since the array grows towards the higher memory addresses, the general-purpose registers are being overwritten, which might cause memory corruption or in the worst case direct access to control functions, if an I/O register is hit.

Such an attack can be prevented by the use of a secure SAC. The SB controls the execution of the CA. Thus, also memory access can be limited. In the case of the secure SAC actuator mentioned before, an `ArrayIndexOutOfBounds-Exception` will be thrown by the SB and the CA will not have direct access to the control functions.

Case study 3 *Prevent attacks on KNX based BASs using secure ICDs*

Using the KNX technology, couplers are used to interconnect different physical lines. They already allow to filter telegrams based on group addresses and also to prohibit physical addressing. Configuration for the devices is automatically generated out of the ETS when they are being programmed and can also be extracted when showing the filter table of a coupler. Thus, only simple security policies can be monitored by those couplers. No mechanisms are present in todays devices, which allow context-based filtering. Using the presented secure ICDs, however, complex policies can be defined and be implemented an enforced by ICDs.

Case study 4 *Prevent Attacks on LonWorks based SACs*

Listing 2 shows an implementation of the SAC light actuator of use case ⊗ on a standard LonWorks device using Neuron C. The Neuron Chip implements an event driven scheduler, which executes code blocks when a given condition becomes `TRUE`. When an update of the PDP `L_MAN` is received, the condition `nv_update_occurs` stores its value in a flag. The condition in line 6 is then triggered to switch the lamp on. Unfortunately, an *General logic error vulnerability* is present in this line. An assignment is performed instead of a comparison. Thus, this condition always evaluates to `TRUE` and the I/O is switched as often as the scheduler checks the condition.

```
// NV input: pdp L_MAN
network input SNVT_switch nviLampValue;
// lamp is connected to I/O 0 pin 0
IO_0 output bit io_lamp_control = 0;
// if flag is equal to TRUE
when (flag=TRUE) {
  // switch lamp on
  io_out(io_lamp_control, 1);
  flag = FALSE;
}
// if flag is equal to FALSE
when (flag==FALSE) {
  // handle switch off light after PAR_OFFD
}
// when new value of L_MAN is received
when (nv_update_occurs(nviLampValue)) {
  // store value in flag
  if (nviLampValue == 1) {
    flag = TRUE;
  }
  else {
    flag = FALSE;
  }
}
```

Listing 2: Implementation of Use Case $\otimes$ on a LonWorks Device using Neuron C

Such an attack can be prevented by the use of a secure SAC with appropriate security policy (cf. D.22), since the SB controls the execution of the CA and the policy specifies a maximum execution rate of the `switchoutput` function of 1Hz.

4.3 Security Guidelines

When deployed to SACs, ICDs and MDs, the presented architecture allows the development, upload, and execution of arbitrary, non-inspected and uncertified (and possibly erroneous or malicious) CAs without compromising the overall BAS security. Not only attacks evolving from accidental software faults can be prevented and detected, but also attacks resulting from intentional malicious software. In the following, the strengths and limits of the architecture – with respect to which requirements for secure CAs are fulfilled, and which requirements need additional research or organizational attention – will be discussed (cf Table 2 and Table 3).

FR–memory access: To fulfill this FR, a secure software environment is required. The system software and enhanced application layer together with the execution of the CA in the SB guarantee, that memory areas (i.e. code and working memory) between system software and CA are separated. CAs can only access defined memory locations. This way it is possible to store information invisible and unaccessible to the CA on the system and provide content security and secure resource access. Vulnerabilities in the code of system software or SB caused by incalculable side effects can be minimized by the use of SCA, IAC and FV.

FR–low level functionality access and *FR–protection of environment*: Attacks targeting these requirements can be prevented by providing the SB together with the security policy. Since memory areas on the SACs are separated, CAs have no access to low level functions. CAs can only issue a defined set of operations and may not interfere with the system software. Additionally, the actions a CA intends to perform can be limited (e.g. in terms of issuing frequency) and a malicious CA is not able to e.g. waste resources.

FR–communication relationship: To prevent and detect attacks targeting the communication relationship, various security measures need to be provided on the different device classes. As presented, a CA communicates with other CAs by accessing DPs in an abstracted, object oriented way. No explicit telegrams need to be sent and no destination addresses need to be considered by a CA. The communication relationship is contained in the configuration, and thus the SB is able to limit it. Using the same configuration/policy, it is possible for an ICD to detect and prevent attacks to other CAs within a reasonable time. To be able to monitor CA communication within the whole BAN, IDSs are required. Using an SIDS configured by the security policy, it can detect known attacks. Using AIDS, IDSs can be used to learn the normal CA behavior of a BAS and later on detect attacks. Monitoring MDPs can be achieved using a MD.

FR–availability: Availability attacks are always hard to handle. In fact, they cannot be prevented in general. Using the secure architecture and the security policy, however, it is at least possible to detect these attacks (e.g. high bus load or abnormal telegram telegram rates)-

OR–limited resources: The deployment of dynamic security measures being executed during runtime clearly imposes a performance overhead. Depending on the application, a suitable balance between required level of security and tolerable overhead needs to be found. For CAs in BAS, it has been shown in the prototype implementations, that such a balance can be achieved on SACs.

OR–development: The SB approach eases the development of secure CAs by providing a limited but controllable programming interface to secure SACs. CA development is simplified, since the application programmer does not have to cope with details concerning the network protocol, the system software or hardware specific code and thus can focus on the CA itself.

OR–high level language support: The architecture supports the use of high-level languages, e.g. Java. Standard Java toolchains can be used for development, offering object oriented development on SACs.

OR–long lifetime: Due to the SB approach, CAs can be downloaded into a SAC. Thus, also updates to CAs are possible. Whether an update of the system software is possible depends on the deployed hardware.

OR–scalability: The presented architecture relies on security measures being distributed to the different devices within a BAS. Thus, also the implied overhead is being distributed among the SACs. Additionally, it does not require any hardware modifications. This makes the architecture scale well with respect to performance and costs of the single devices.

OR–network technology: The secure CA architecture is geared towards the small amount of control data in the BAS domain. In fact, the mechanisms only target the monitoring of the present value being exchanged between CAs. No overhead is generated on the network, except when active MDs are used.

OR–compatibility: The presented architecture relies on a security policy, defining the normal and abnormal behavior of a BAS. Since control data exchange via the BAN is not affected, an easy and compatible integration into existing BASs becomes possible by changing or adding single devices. If legacy devices are still used in an installation, at least attack detection by additional security devices (i.e. ICDs, MDs) becomes possible using the security policy.

OR–physical access: Detecting physical manipulations of SACs, requires special treatment in hardware. Alarming of intrusions, however, can be performed by e.g. an IDS, monitoring other security related issues.

OR–usability: Providing usability of security measures seems to be hardest task, when they are being put into practice. As shown in Section 2, thousands of installations are directly connected to the Internet. No encryption or passwords are set. It can be assumed, that additional security mechanisms, which are not usable, will not be enabled in practice. Regarding the presented architecture, it is essential that the security policy is defined in a sound way. Therefore, typical configurations need to be provided by standards and manufacturers and integrators need to enable the security features.

5 Summary and Future Work

This work concentrates on the problems of secure CAs in BAS and contributes the following novelties. The application models of today's BAS technologies and contained background knowledge have been analyzed and a formalism to express this knowledge has been described. The developed application model is the first to cover BAS software security at all. Its expressiveness allows to describe any desired CA, which can be formulated using FBs. As has been shown in this paper, the available domain knowledge can then be utilized to formulate a security policy and to provide an overall security. A framework allows the secure development and execution of CAs and enforcement of the defined security policy. Finally, evaluation prototypes describe the experimental results of an integration of all components into a common secure BAS and validate the presented secure CA architecture.

Additional work, however, is required to formulate generic security policies. In fact, it has been shown, that security attributes can be mapped onto today's

Secure CA Architecture	*FR–memory access*	*FR–low level functionality access* *FR–protection of environment*	*FR–communication relationship*	*FR–availability*
SAC	{SCA, HA, IAC, FV}+SB	SB+IDS	SB	
ICD			IDS	
IDS			SIDS+AIDS	SIDS+AIDS
MD			IDS	IDS

Table 2: Security Evaluation of Functional Requirements [Praus, 2015]

Secure CA Architecture	*OR–limited resources*	*OR–development*	*OR–high level language support*	*OR–long lifetime*	*OR–scalability*	*OR–network technology*	*OR–compatibility*	*OR–physical access*	*OR–usability*
	$\sim$	$+$	$+$	$\sim$	$+$	$+$	$\sim$	n/a	n/a

Table 3: Security Evaluation of Organizational Requirements [Praus, 2015]

open BAS technologies. A generic policy being ready to deploy on real installations, however, is missing. Besides, the security of the security measures needs to be considered, when the devices are installed in practice. The security of the system software needs to be provided, likewise as the secure implementation of an ICD or MD. Otherwise, adversaries will first attack or disable those devices and circumvent the security measures. An overall device security also needs to include side channel and physical attacks. Without providing protection against these attacks, adversaries also can be assumed to bypass software protection techniques.

Acknowledgements

This work was partly funded by the City of Vienna, under grant number MA23-Projekt 15-03.

References

[Arora et al., 2006] Arora, D., Ravi, S., Raghunathan, A., and Jha, N. K. (2006). Hardware-Assisted Run-Time Monitoring for Secure Program Execution on Embedded Processors. 14(12):1295–1308.

[Aucsmith, 1996] Aucsmith, D. (1996). Tamper Resistant Software: An Implementation. In *Proceedings of the First International Workshop on Information Hiding*, pages 317–333, London, UK. Springer-Verlag.

[Chess and McGraw, 2004] Chess, B. and McGraw, G. (2004). Static Analysis for Security. 2(6):76–79.

[Collberg and Thomborson, 2002] Collberg, C. S. and Thomborson, C. (2002). Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection. In *IEEE Transactions on Software Engineering*, volume 28, pages 735–746.

[Dzung et al., 2005] Dzung, D., Naedele, M., Von Hoff, T., and Crevatin, M. (2005). Security for Industrial Communication Systems. 93(6):1152–1177.

[Goldberg et al., 1996] Goldberg, I., Wagner, D., Thomas, R., and Brewer, E. A. (1996). A Secure Environment for Untrusted Helper Applications. In *Proceedings of the 6th Usenix Security Symposium*, San Jose, CA, USA.

[Granzer, 2010] Granzer, W. (2010). *Secure Communication in Home and Building Automation Systems*. PhD thesis, Vienna University of Technology.

[Hiroaki et al., 2007] Hiroaki, I., Edahiro, M., and Sakai, J. (2007). Towards Scalable and Secure Execution Platform for Embedded Systems. In *ASP-DAC '07: Proceedings of the 2007 Conference on Asia South Pacific Design Automation*, pages 350–354, Washington, DC, USA. IEEE Computer Society.

[Hofmeyr et al., 1998] Hofmeyr, S. A., Forrest, S., and Somayaji, A. (1998). Intrusion Detection Using Sequences of System Calls. *Journal of Computer Security*, 6(3):151–180.

[Holzmann, 2006] Holzmann, G. (2006). The Power of 10: Rules for Developing Safety-Critical Code. 39(6):95–99.

[IEC, 2011] IEC (2011). Function Blocks – Part 1: Architecture. IEC 61499-1.

[Kinder et al., 2010] Kinder, J., Katzenbeisser, S., Schallhart, C., and Veith, H. (2010). Proactive Detection of Computer Worms Using Model Checking. 7(4):424–438.

[Koeune and Standaert, 2004] Koeune, F. and Standaert, F.-X. (2004). A Tutorial on Physical Security and Side-Channel Attacks. In *Foundations of Security Analysis and Design III, FOSAD 2004/2005 Tutorial Lectures*, pages 78–108.

[Landi, 1992] Landi, W. (1992). Undecidability of Static Analysis. *ACM Letters on Programming Languages and Systems*, 1(4):323–337.

[Li et al., 2005] Li, Z., Das, A., and Zhou, J. (2005). Theoretical Basis for Intrusion Detection. In *Proceedings of 6th IEEE Information Assurance Workshop (IAW)*, West Point, NY, USA. IEEE SMC Society.

[McHugh, 2001] McHugh, J. (2001). Intrusion and Intrusion Detection. *International Journal of Information Security*, 1(1):14–35.

[Necula and Lee, 1998] Necula, G. C. and Lee, P. (1998). Safe, Untrusted Agents Using Proof-Carrying Code. In *Mobile Agents and Security*, pages 61–91, London, UK. Springer-Verlag.

[Pfleeger and Pfleeger, 2002] Pfleeger, C. P. and Pfleeger, S. L. (2002). *Security in Computing*. Prentice Hall Professional Technical Reference.

[Praus, 2015] Praus, F. (2015). *Secure Control Applications in Smart Homes and Buildings*. PhD thesis, Technische Universitt Wien.

[Praus et al., 2009] Praus, F., Granzer, W., and Kastner, W. (2009). Enhanced Control Application Development in Building Automation. In *Proc. 7th IEEE International Conference on Industrial Informatics (INDIN'09)*, pages 390–395.

[Praus and Kastner, 2010] Praus, F. and Kastner, W. (2010). Secure Control Applications in Building Automation Using Domain Knowledge. In *Proc. 8th IEEE International Conference on Industrial Informatics (INDIN'10)*, pages 52–57.

[Praus et al., 2016] Praus, F., Kastner, W., and Palensky, P. (2016). Software security requirements in building automation. In *Sicherheit 2016: Sicherheit, Schutz und Zuverlässigkeit, Beiträge der 8. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), 5.-7. April 2016, Bonn*, pages 217–228.

[Ravi et al., 2004] Ravi, S., Raghunathan, A., Kocher, P., and Hattangady, S. (2004). Security in Embedded Systems: Design Challenges. *Transactions on Embedded Computing Systems*, 3(3):461–491.

[Riley et al., 2007] Riley, R., Jiang, X., and Xu, D. (2007). An Architectural Approach to Preventing Code Injection Attacks. In *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 30–40, Washington, DC, USA. IEEE Computer Society.

[Science Applications International Corporation, 2008] Science Applications International Corporation (2008). INTEGRITY-178B Separation Kernel Security Target Version 1.0. Common Criteria Testing Laboratory.

[Sekar et al., 2003] Sekar, R., Venkatakrishnan, V., Basu, S., Bhatkar, S., and DuVarney, D. C. (2003). Model-Carrying Code: A Practical Approach for Safe Execution of Untrusted Applications. In *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 15–28, New York, NY, USA. ACM.

[Suh et al., 2004] Suh, G. E., Lee, J. W., Zhang, D., and Devadas, S. (2004). Secure Program Execution via Dynamic Information Flow Tracking. In *ASPLOS-XI: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 85–96, New York, NY, USA. ACM.

[Tanenbaum and van Steen, 2002] Tanenbaum, A. S. and van Steen, M. (2002). *Distributed Systems: Principles and Paradigms*. Prentice Hall.

[Verein Deutscher Ingenieure, 2015] Verein Deutscher Ingenieure (2015). Building Automation and Control Systems (BACS) – Application Examples for Room Types and Function Macros of Room Automation and Control. VDI 3813-3.

[Wilander and Kamkar, 2003] Wilander, J. and Kamkar, M. (2003). A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention. In *Proceedings of the 10th Network and Distributed System Security Symposium*, pages 149–162, San Diego, California.

Acronyms

AIDS Anomaly based Intrusion Detection System
API Application Programming Interface
ASC Attack Specific Countermeasure
BAN Building Automation Network
BAS Building Automation System
BIM Bus Interface Module
CA Control Application
CP Co-Processor
CPUEX CPU EXtension
CS Code-Signing
DoS Denial of Service
DP Datapoint
DRM Digital Rights Management
ETS Engineering Tool Software
FB Function Block
FR Functional Requirement
FV Formal Verification
HA Harvard Architecture
HAL Hardware Abstraction Layer
HVAC Heating, cooling, Ventilation, and Air Conditioning
IAC Inspection And Certification
ICD InterConnection Device
IDS Intrusion Detection System
I/O Input/Output
IT Information Technology
MCU MicroController Unit
MD Management Device
MDP Management Datapoint
OR Organizational Requirement
OS Operating System
PCC Proof-Carrying Code
PDP Physical or Process Datapoint
PP Physical Partitioning
SAC Sensor Actuator and Controller
SB Sandbox
SCA Static Code Analysis
SCC Self Checking Code
SFPT Standard Functional Profile Template
SIDS Signature based Intrusion Detection System
SMT Software Monitoring Techniques
WM Watermarking