M. Meier, D. Reinhardt, S. Wendzel (Hrsg.): Sicherheit 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 217

Software Security Requirements in Building Automation

Friedrich Praus[1], Wolfgang Kastner[2], Peter Palensky[3]

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.

Obviously, the demands – especially regarding security – increase: Secure communication becomes equally important as secure software being executed on the devices. While the former has been addressed by standardization committees and manufacturers, until recently no scientific research is available, that targets the problem of secure control applications in this domain. No attack model has been defined, no security measures have been recommended, existing measures from other domains are either too expensive or time intensive to deploy, cannot be trivially applied to or do not cover specific demands and constraints of the building automation domain.

This paper provides an extensive survey of the security requirements for distributed control applications and analyzes software protection methods. An architecture tackling the problem on how to secure software running on different device classes and preventing attacks on smart homes and buildings is briefly introduced at the end.

Keywords: Secure Software, Security Process, Secure Control Applications, Smart Homes, Security, Building Automation

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 [Gr10]. 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 [KS04]. Until recently, no scientific research is available that extensively targets security requirements and software attacks in the BAS domain.

Figure 1 briefly describes the life cycle of a BAS and involved stakeholders over the building lifetime. A building owner defines the requirements and instructs a planner to plan the building. The system integrator selects and commissions the devices and an installer deploys them in the building. The facility manager finally maintains the building. Basically, security is essential in all steps, for all stakeholders and during the complete lifetime. To

[1] University of Applied Sciences Technikum Wien, Höchstädtplatz 6, 1200 Wien, praus@technikum-wien.at,
This work was partly funded by the City of Vienna, under grant number MA23-Projekt 15-03.
[2] TU Wien, Automation Systems Group, Treitlstr. 1-3/4, 1040 Wien, k@auto.tuwien.ac.at
[3] TU Delft, Intelligent Electrical Power Grids, Mekelweg 4, 2628 CD Delft, P.Palensky@tudelft.nl

focus on secure Control Applications (CAs), the following use cases are identified as being security critical (blue and bold-italic markings in Figure 1): Starting from secure CA development, committees need support in standardizing mechanisms, frameworks and generic policies, while CA manufacturers need support in creating CAs. System integrators and facility managers need support in adapting security policies during CA operation.

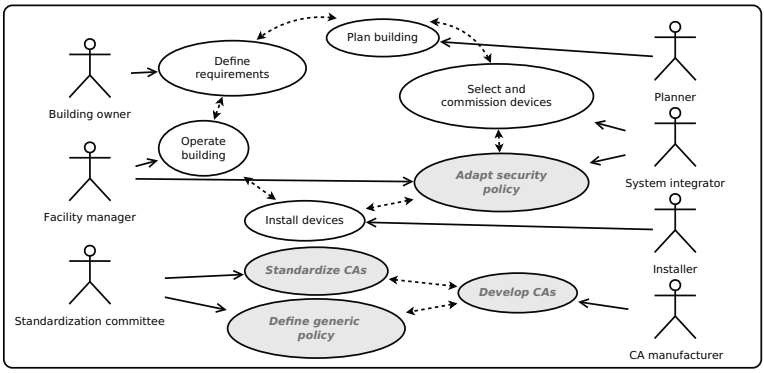


Fig. 1: Life Cycle of a Building Automation System and Involved Stakeholders

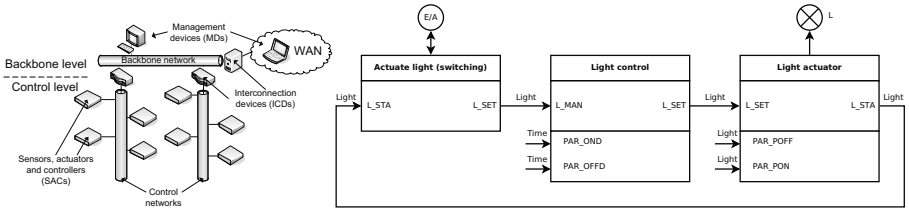
Today's software for smart home and building devices lacks adequate security mechanisms. Irrespective of the used technology, no sound protection against software attacks is deployed, thus enabling adversaries to successfully attack those devices. Existing protection techniques from other domains (e.g. the Information Technology (IT) domain) are insufficient and not applicable to BASs due to different functional and non-functional requirements. Thus, a secure architecture being adaptable to all common BAS standards needs to be established. This architecture needs to cover BAS specific constraints, provide a security policy and a secure software environment. Besides, mechanisms for attack detection are needed [Pr15].

The paper is structured as follows. Section 2 analyzes domain constraints in distributed CAs and the structure of modern BASs and their devices. Based on a short analysis of CA security in current building automation standards and technologies and a software security threat and risk analysis, security requirements are identified (cf. Section 3). Existing software protection techniques are briefly investigated and evaluated for BASs with respect to their applicability and implied security gain (cf. Section 4). Then, a first concept for secure and distributed CAs, describing how to fulfill the demands for security-critical smart homes and buildings is presented (cf. Section 5). Finally, an outlook on the evaluation and future work is described in Section 6.

2 Control Applications in Building Automation Systems

Today's BAS are implemented following a two-tier architecture (cf. Fig. 2a). It consists 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.



(a) Building Automation Network [GPK10]

(b) Use Case ⊗: Distributed Control Application: VDI 3813-3: D-1-2 Lighting Control Manual with Time-Controlled Switching Off (Stairwell Light)

Fig. 2: Control Applications in Building Automation Systems

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 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. 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 and will be used throughout the remaining sections. Figure 2b 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`*.*

Today's open BAS technologies (i.e. BACnet, EnOcean, KNX, LonWorks and ZigBee) achieve interoperability each by standardizing their own CA model. As shown in [Pr15], these application models differ significantly even for simple use cases such as a stairwell light or an individual room control. A transparent (i.e. translation/gateway-free) communication across technology borders is likewise impossible, as common security mechanisms are missing. The actual functionality of a BAS, however, is always similar and most of the traditional functions can likewise be realized by any of these technologies. An overall software security in smart homes and buildings can only be established, if the required measures are applicable to all common standards and technologies. Therefore, to be able to develop a secure CA architecture being adaptable to all different standards and technologies, a consistent application model is required. Security can then be investigated for this model and appropriate measures can be derived and developed for the different technologies and standards.

3 Control Application Security

The ultimate goal of an adversary is to gain unauthorized access to control level functions by manipulating the software being executed on BAS devices. A possible attack scenario is an adversary trying to hack a door switch to illegally access a building or to neutralize a light actuator by crashing its CA. 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.

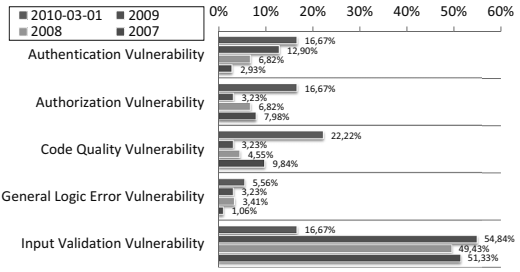
3.1 An Overview of Security in BAS

To identify the security threat in current installations, recent research [Pr15] targeted the question, whether and how many BASs based on BACnet and KNX are openly connected to the Internet and what 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. Table 3a). BACnet is being widely used in the US and Canada whereas KNX is very popular in Europe. 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

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

Fig. 3: Control Application Security (Top 5 Countries and Categories)

Project Top 10[3] 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 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 3b shows a breakdown of the top 5 vulnerabilities being openly available at the US-CERT Vulnerability Notes Database[4]. All entries of the years 2007 to March 2010 (632 in total) have been analyzed, categorized and counted [Pr15]. Although being rather outdated, the numbers still give an adequate overview of the commonness of vulnerability types.

As shown in this section, security awareness in the BAS domain is missing and today's CAs need to be considered as insecure. Thousands of installations are being directly connected to the Internet, allowing an adversary to attack them.

3.2 Security Requirements

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. They are derived out of security research in e.g. industrial communication systems [Dz05], embedded systems [Ra04] or cyber-physical systems.

[3] `http://www.owasp.org/`, Last access: 2016/02/02
[4] `http://www.kb.cert.org/vuls`, Last access: 2010/04/03

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*:** Considering the execution of a CA on a SAC, the memory access must be controlled. On the one hand, a CA must not be allowed to access arbitrary memory locations to e.g. prohibit, that a malicious CA subverts any security mechanism. On the other hand, a secure storage of protected data must be possible. To put it differently, information such as configuration parameters or cryptographic keys invisible and unaccessible to the CA need to be stored on the SAC to provide the basis for a secure system. Vulnerabilities (e.g. memory corruption via buffer and format string overflows or code injection) caused by side effects have to be prevented.
***FR–low level functionality access*:** The same way it must be possible to limit the actions and allowed operations (e.g. access to low-level function calls) a CA can perform with respect to

- Access rights: Is a CA allowed to call a particular function or not? Note, that often a generic system software is deployed on SACs with far more capabilities and functions than a simple CA may need. Hence, it is desirable to limit the allowed operations for a SAC.
- Parameters: Likewise the parameters of a function call need to be limited so that e.g. the present value of a Datapoint (DP) does not exceed a critical value.
- Execution time: The point in time when a function is called is an additional constraint to monitor. Not only the actual instance, but also the invoking frequency is critical for some applications.
- Domain constraints: Dependencies between function calls, which can be seen as domain constraints, are a further critical issue. Consider e.g. an Heating, cooling, Ventilation, and Air Conditioning (HVAC) application, where it is not desirable to simultaneously switch on the heating and the cooling function.

***FR–protection of environment*:** CAs must neither destruct the hardware or waste resources intentionally nor due to programming flaws (e.g. wear out of a flash memory or exhaust battery power).
***FR–communication relationship*:** CAs have a defined (static) communication relationship. Being readily configured, it is known which CAs need to communicate and which CAs do not need to communicate. A simple light switch, for instance, must not be hacked and abused to open a security door. This communication relationship needs to be considered in security mechanisms.
***FR–availability*:** Availability needs to be guaranteed. Denial of Service (DoS) attacks need to be prevented or detected.

Organizational Requirements (ORs) cover the special environmental conditions required for developing secure CAs in BAS.

***OR–limited resources*:** Due to cost efficiency and form factor, SACs are normally embedded devices with limited system resources (e.g. memory, processing power) that rely on either bus or battery power. Security mechanisms are computationally intensive and must not exceed the available device's processing resources (processing gap) and power

resources (battery gap) [Ra04]. The overhead imposed by these mechanisms needs to be reasonably small. Therefore, a suitable balance between a required level of security and available resources has to be found (”good enough security”).

***OR–development*:** CA development has to be simple and secure by design so that even security unaware developers are able to design secure CAs. This is especially important for the BAS domain, since engineers are experts in the field of automation but not in the field of security. Therefore, a two level concept with a dedicated system software and a CA, as already present in KNX or LonWorks, needs to be supported.

***OR–high level language support*:** High-level programming languages (e.g. Java) need to be supported such that the desired control logic and behavior can be obtained more easily. CA development is also simplified, since the application programmer does not have to cope with details such as a hardware specific system software or the communication protocol.

***OR–long lifetime*:** BASs have to be kept operable for years or even decades. Due to this long lifetime, such systems obviously have to undergo maintenance during runtime in order to keep them operable. With the complexity of the CA software increasing, it also must be assumed that not all implementation flaws can be detected in the development phase. Since these may result in security vulnerabilities, a secure update mechanism is beneficial. Such a mechanism should allow the distribution of system software patches and secure download and replacement of CAs in an easy and secure manner.

***OR–scalability*:** Since BASs can consist of hundreds or even thousands of devices, appropriate scalability of security mechanisms is essential. For instance, key distribution schemes which routinely require physical access to the individual devices are not feasible in large networks. Therefore, services must be provided which assist in performing these tasks.

***OR–network technology*:** Security mechanisms need to be geared towards the different requirements in BANs regarding the used network technology. While in the IT world Internet Protocol (IP) based network protocols are dominant, the use of IP networks in BANs is reserved to the backbone level. At the field level, predominantly non-IP fieldbus are used. Besides, control data typically transmitted in BANs have a small volume (in the order of bytes) with perhaps soft real time requirements (e.g. reaction time in a lighting system).

***OR–compatibility*:** The integration of a security extension into an established BAS is preferable to create an entirely new system. Such an approach allows to leverage the existing base of available components for parts of the system where security is not (yet) a requirement. This allows a smooth transition until devices supporting the security extension become widely available. It also offers an economical upgrade path for existing installations.

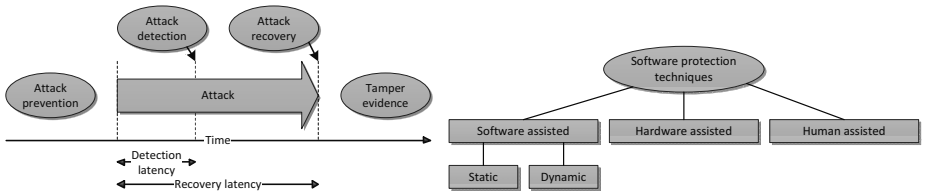
***OR–physical access*:** In BANs, devices often operate in untrusted environments where physical access (e.g. an intrusion alarm in a public building or a wireless sensor network [GH09]) is given. Therefore, it has to be assumed that a short time physical access to devices and networks cannot be avoided. Such attacks have to be detected by a security system.

***OR–usability*:** Usability of security measures has to be provided, when these systems are installed. On the one hand, this implies that it has to be possible to deploy them as easily as possible. In the best case, users do not even notice, that a security measure is enabled.

At least, education and guidelines (e.g. secure password guidelines) need to be provided for support. On the other hand, this requirement also covers protection against social engineering attacks.

As shown in this section manifold requirements need to be considered to be able to provide secure CAs.

4 Software Protection Techniques



(a) Attack Model (b) Software Protection Techniques

Fig. 4: Software Protection Techniques [Pr15]

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.

This section focuses on methods trying to prevent or at least detect attacks within a reasonable detection latency (cf. Figure 4a). Possible software protection techniques (cf. Figure 4b) are briefly discussed with respect to the security requirements formulated in the previous section and applicability to the device classes in BAS.

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

Static methods 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. However, they cannot detect all possible vulnerabilities without actually executing a piece of software. Besides, Static Code Analysis (SCA), Code-Signing (CS) and Proof-Carrying Code (PCC) have to be performed on every code change. Nevertheless, they are assumed to be easily applicable with respect to resources of the target system because they are only used at compile time. CS can be quite effective to prevent the installation of arbitrary CAs by simply refusing to execute unsigned or not properly signed code. Watermarking (WM) applied to software protects against illegitimate modifications and tampering by its users. The universal applicability of PCC to BAS is questionable since the generation and encoding of proofs for complex security policies are nontrivial tasks and have to be performed on every code change.

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

	method or BAS	*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	SCA	−	−	∼	−	−	+	∼	+	−	−	+	∼	−	−	∼	+	−
software	CS	−	−	−	−	−	+	∼	+	+	+	+	∼	−	−	+	+	+
methods	WM	−	−	−	−	−	+	−	+	−	+	+	∼	−	−	+	+	+
	PCC	−	−	∼	−	−	+	−	+	+	−	+	∼	−	−	−	∼	∼
	SIDS	−	−	*d*	*d*	−	∼	∼	+	−	−	+	+	−	−	+	+	∼
	AIDS	−	−	*d*	*d*	*d*	−	+	+	+	+	+	+	+	∼	+	+	∼
dynamic	SMT	−	−	∼	−	−	∼	−	+	−	−	+	∼	−	−	∼	+	−
software	SB	*p*	*p*	*p*	*p*	−	∼	+	+	+	+	+	∼	−	+	+	∼	+
methods	SCC	−	−	−	−	−	∼	−	+	−	−	∼	∼	−	−	−	+	−
	ASC	−	−	−	−	−	∼	−	+	−	−	+	∼	−	∼	∼	+	+
	OS	*p*	*p*	*p*	*p*	−	−	+	+	+	+	+	∼	−	+	−	−	+
	CP	−	∼	−	∼	−	+	−	+	−	−	+	∼	+	−	−	−	+
hardware	PP	*p*	*p*	−	−	−	+	+	+	−	−	+	∼	+	−	−	−	+
supported	HA	−	−	−	−	−	+	+	+	−	+	+	∼	−	+	+	+	+
	CPUEX	−	−	−	−	−	+	+	−	−	+	+	∼	−	+	+	+	+
human	IAC	∼	−	*p*	−	−	+	−	+	−	−	+	∼	−	−	−	∼	−
	FV	−	−	*p*	−	−	+	−	+	−	−	+	∼	−	−	−	∼	−

Tab. 1: Comparison of Software Protection Techniques with Respect to Security Requirements and Applicability to Sensors, Actuators and Controller Devices, Interconnection Devices, and Management Devices

Dynamic methods 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. Signature based Intrusion Detection Systems (SIDSs) offer a high attack detection accuracy. They, however, cannot detect new attacks and are vulnerable to attack variations such as worms, that alter their own code base on succeeding executions. SIDSs are not well suited as they depend on a usually large database and require constant updates, which would be difficult in case of SACs and ICDs. Anomaly based Intrusion Detection Systems (AIDSs) in contrast also allow to detect novel intrusions, which are not yet present in the database of an Intrusion Detection System (IDS). They are, however, not able to distinguish between natural changes of the monitored system and attacks. AIDS, Software Monitoring Techniques (SMT) as well as Self Checking Code (SCC) may be efficiently implemented and could therefore be quite appropriate. SMT at least requires hardware support for context switches. Attack Specific Countermeasures (ASCs) can also work well in many cases, but might not be applicable due to differing processor and memory architectures on SACs and ICDs. The applicabil-

ity of Sandboxes (SBes) strongly depends on the overhead imposed by their feature sets. While a reduced and lightweight SB could easily be deployed to SACs, an architecture like the full Java Virtual Machine (JVM) with its vast execution and security mechanisms imposes a big overhead. While in the IT world and thus also for MDs OSs typically limit what an application is allowed to do, the targeted MCUs being deployed to SACs and ICDs do not provide the necessary hardware support (e.g. lack of memory management units to separate the address spaces of different processes).

Hardware assisted methods use dedicated hardware for security checks (e.g. Co-Processor (CP), Physical Partitioning (PP), Harvard Architecture (HA), CPU EXtension (CPUEX)) and allow to lower the imposed performance overhead in contrast to pure software based methods. However, hardware supported methods requiring additional components cannot be cost effectively deployed to SACs and recent research also demonstrates that these methods may also be bypassed [Ro12]

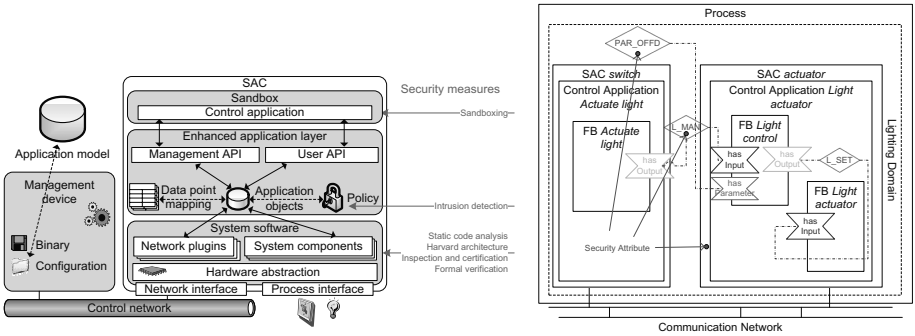
Human assisted methods rely on human expertise during CA development. Inspection And Certification (IAC) performed by humans may eliminate a lot of possible attacks, but requires extensive knowledge by the auditing person, is time consuming, expensive and error-prone. Thus, it does not scale at all and may limit flexibility, since it is only feasible to be applied to code, which does not change frequently. Formal Verification (FV) is the most tenable method for providing security constraints. If security attributes can be represented in a formal way, provers can guarantee that these attributes are met. However, applying FV to real life software or even an OS is a very hard task and only two approaches are known for now that allow Common Criteria EAL6+ certification [SAIC08, Kl09].

Hybrid methods 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. 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.

5 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.

The resulting secure CA architecture (cf. Figure 5a) consists of four parts: First, a generic application model is defined, being required to develop a secure system being used for the different BASs. It separates generic information of BASs from an installation dependent one. This is achieved by the definition of the 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.



(a) Architecture of Sensor Actuator and Controllers (b) Application Model and Security Policy for Use Case ⊗

Fig. 5: Secure Control Application Architecture

Second, a security policy based on security attributes allows a formal way to formulate security requirements. This global policy 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 of DPs, where security requirements for the conditions derived from the policy can be defined, formulated and finally evaluated. An instance of the application model and an example security policy with security attributes for use case ⊗ is shown in Figure 5b.

Third, a software environment to securely execute CAs and enforce the security policy is needed. As outlined in Figure 5a, it consists of three major components, each imposing an additional security barrier to the overall security and limiting possible security threats: A system software provides controlled access to system resources. It encapsulates hardware specific details, the network protocol stack, the process interface as well as any further system components and offers clean interfaces for the enhanced application layer. An enhanced application layer stores the application objects, their DP mappings as well as the security policy for the CA. A Sandbox executes the CA in a controlled way and is also designed to support its rapid development. It interfaces the system software via the enhanced application layer and provides a clear abstraction of the underlying hardware and software by providing an object-oriented access (using e.g. the Java programming language). The application designer can thus focus on the application development. Finally, methods to detect possible attacks (e.g. DoS) and violations of the security policy are needed.

6 Summary and Future Work

To summarize, the following hybrid software protection mechanisms are deployed to provide security of the presented architecture SCA, IAC and code reviews are performed on the system software, which provides an abstraction and layering to ease CA development. A combination of an AIDS and a SIDS based upon a security policy and a generic application model are used to detect and prevent attacks. Uncircumventable sandboxing of

CAs is performed to protect the system outside of the SB from attacks. WM could also be deployed to only execute signed CAs, if desired.

Details on the proposed architecture and a validation of its feasibility have recently been shown in [Pr15]. They will be published in future work: First, the process of how to implement secure CAs will be described and it will be shown, how the generic architecture can be instantiated, i.e. how the CA can be modeled and how the policies can be adapted. On the basis of use case $\otimes$, prototypes will be implemented for the different device classes to evaluate and test the stability with respect to memory consumption, performance, and security. To further evaluate the presented concept, a discussion on how it can be used to enable security in today's already existing BASs will follow. Attack detection and prevention become possible, if appropriate devices are installed. Concluding, an exhaustive discussion will evaluate, that all functional requirements are fulfilled by means of the concept. Some organizational requirements still need to be considered, when implementing real live installations. Thus, it will be proven, that the developed secure CA architecture can be implemented on the devices typically found in BAS. Besides, it fulfills the requirements for secure CAs and is able to prevent or at least detect attacks.

References

[Dz05] Dzung, D.; Naedele, M.; Von Hoff, T.P.; Crevatin, M.: Security for Industrial Communication Systems. 93(6):1152–1177, 2005.

[GH09] Gungor, V.; Hancke, G.: Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches. 56(10):4258–4265, 2009.

[GPK10] Granzer, Wolfgang; Praus, Friedrich; Kastner, Wolfgang: Security in Building Automation Systems. IEEE Transactions on Industrial Electronics, 57(11):3622–3630, Nov.'10.

[Gr10] Granzer, Wolfgang: Secure Communication in Home and Building Automation Systems. PhD thesis, Vienna University of Technology, February 2010.

[Kl09] Klein et al.: seL4: Formal Verification of an OS Kernel. In: SOSP '09: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. ACM, New York, NY, USA, pp. 207–220, 2009.

[KS04] Koeune, François; Standaert, François-Xavier: A Tutorial on Physical Security and Side-Channel Attacks. In: Foundations of Security Analysis and Design III, FOSAD 2004/2005 Tutorial Lectures. pp. 78–108, 2004.

[Pr15] Praus, Friedrich: Secure Control Applications in Smart Homes and Buildings. PhD thesis, Technische Universität Wien, November 2015.

[Ra04] Ravi, Srivaths; Raghunathan, Anand; Kocher, Paul; Hattangady, Sunil: Security in Embedded Systems: Design Challenges. Transactions on Embedded Computing Systems, 3(3):461–491, 2004.

[Ro12] Roemer, Ryan; Buchanan, Erik; Shacham, Hovav; Savage, Stefan: Return-Oriented Programming: Systems, Languages, and Applications. ACM Trans. Inf. Syst. Secur., 15(1):2:1–2:34, March 2012.

[SAIC08] Science Applications International Corporation, Common Criteria Testing Laboratory: , INTEGRITY-178B Separation Kernel Security Target Version 1.0, 2008.