FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Secure Control Applications in Smart Homes and Buildings

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Mag. Dipl.-Ing. Friedrich Praus
Registration Number 0025854

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner

The dissertation has been reviewed by:

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner	Prof. Dr. Peter Palensky

Vienna, 1st October, 2015

Mag. Dipl.-Ing. Friedrich Praus

Erklärung zur Verfassung der Arbeit

Mag. Dipl.-Ing. Friedrich Praus
Hallergasse 11/29, A-1110 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Oktober 2015

Friedrich Praus

Kurzfassung

Die zunehmende Integration von heterogenen Gebäudeautomationssystemen ermöglicht gesteigerten Komfort, Energieeffizienz, verbessertes Gebäudemanagement, Nachhaltigkeit sowie erweiterte Anwendungsgebiete, wie beispielsweise "Active Assisted Living" Szenarien. Diese Smart Homes und Gebäude sind heutzutage als dezentrale Systeme realisiert, in denen eingebettete Geräte Prozessdaten über ein Netzwerk austauschen.

Offensichtlich verändern sich dabei die Anforderungen an derlei Systeme, vor allem hinsichtlich der Informations- und Datensicherheit (Security). Dem Themengebiet sichere Kommunikation kommt dabei ein ähnlich wichtiger Stellenwert zu wie dem Aspekt der Softwaresicherheit. Während erstere Thematik bereits von Standardisierungsgremien und Herstellern aufgegriffen wurde, gibt es bis jetzt keine wissenschaftliche Aufarbeitung, wie das Problem der Softwaresicherheit in diesem Bereich systemweit realisiert werden kann. Kein generisches Angriffsmodell ist bekannt und es fehlt an Sicherheitsempfehlungen. Existierende Schutzmechanismen sind entweder zu zeit- und kostenintensiv oder können nicht einfach auf bestehende Technologien übertragen werden bzw. berücksichtigen nicht die besonderen Anforderungen. Der Entwurf und die Umsetzung von Sicherheitsmaßnahmen wird daher EntwicklerInnen überlassen, die oft aufgrund der Vielfältigkeit des Problems und der Sicherheitsanforderungen überfordert sind. Daraus resultiert, dass Steuerungs- und Regelungsanwendungen unsicher ausgeführt sind, und es Widersachern ermöglicht wird, Gebäudeautomationssysteme anzugreifen.

Diese Dissertation stellt eine Architektur für sichere und verteilte Steuerungs- und Regelungsanwendungen in Smart Homes und Gebäuden vor. Damit soll das Problem gelöst werden, wie diese Software sicher auf den unterschiedlichen oft eingebetteten Systemen ausgeführt werden kann. Die folgenden, bisher noch nicht wissenschaftlich aufgearbeiteten Themen, werden diskutiert: eine umfassende Identifikation der Sicherheitsanforderungen, ein Anwendungsmodell, das es ermöglicht Steuerungs- und Regelungsanwendungen formal zu spezifizieren, das Konzept von Sicherheitsattributen, die die Formulierung einer Sicherheitsrichtlinie erlauben und zu guter Letzt, eine Architektur, die die sichere Entwicklung und Ausführung von Steuerungs- und Regelungsanwendungen sowie die Einhaltung von Sicherheitsrichtlinien garantiert.

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 (recently) been addressed by standardization committees and manufacturers, until now 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 cost or time intensive to deploy, cannot be trivially applied to or do not cover specific demands and constraints of the building automation domain. Thus, deploying adequate control application security measures is left open to developers, who are overburdened with the manifold and often unknown security requirements. This yields to insecure control applications, which enable adversaries to attack building automation systems.

This dissertation 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, which – to the best knowledge of the author – have not been addressed in research, yet: a comprehensive identification of security requirements for control applications in smart homes and buildings, 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.

Acknowledgements

Ich möchte mich zuallererst bei meinem Doktorvater Wolfgang "k" Kastner bedanken. Danke k, dass du mir gezeigt hast wie schön die Welt der Wissenschaft ist und mir beigebracht hast, mich darin zurecht zu finden. Ohne deine Unterstützung wäre diese Arbeit nicht möglich gewesen. Danke, dass du mir auch als Freund seit mittlerweile über 11 Jahren beiseite stehst.

Ein großes Dankeschön auch an Peter Palensky. Danke Peter für deine Kommentare, die wesentlich geholfen haben, diese Dissertation zu verbessern.

Ich möchte mich bei meinen Studiums-, Arbeitskollegen und Freunden für die Unterstützung bedanken. Allen voran gilt mein großer Dank Wolfgang Granzer. Danke Woif, für die zahlreichen gemeinsamen Publikationen, die Motivation und die Unterstützung bei der Erstellung dieser Arbeit. Danke Christian, Felix, Lukas, Marion und Woif für das Korrekturlesen.

Ein besonderer Dank gilt meiner Familie. Danke für die jahrelange Unterstützung und Hilfe bei meiner Ausbildung.

This work was partly funded by the FWF (Österreichischer Fonds zur Förderung der Wissenschaftlichen Forschung; Austrian Science Foundation) under the project P19673 and by the City of Vienna, department MA23, under grant number MA23-Projekt 10-04.

Contents

List of Figures

List of Tables

Listings

Acronyms

AES Advanced Encryption Standard

AIDS Anomaly based Intrusion Detection System

API Application Programming Interface

ARC4 Alleged Ron's Code 4

ASC Attack Specific Countermeasure

APDU Application Protocol Data Unit

ASHRAE American Society of Heating, Refrigerating and Air-Conditioning Engineers

ASLR Address Space Layout Randomization

BAN Building Automation Network

BAS Building Automation System

BAU Bus Attachment Unit

BCU Bus Coupling Unit

BIM Bus Interface Module

CA Control Application

CBC Cipher-Block Chaining

CC	Common Criteria for Information Technology Security Evaluation
CCM	Counter with CBC-MAC
CLDC	Connected Limited Device Configuration
CMAC	Cipher Based Message Authentication Code
CP	Co-Processor
CPUEX	CPU EXtension
CRC	Cyclic Redundancy Check
CS	Code-Signing
DDC	Direct Digital Control
DEP	Data Execution Protection
DES	Data Encryption Standard
DIB	Description Information Block
DNS	Domain Name Service
DoS	Denial of Service
DP	Datapoint
DPT	Data Point Type
DRM	Digital Rights Management
EAL	Evaluation Assurance Level
ECB	Electronic Codebook Mode
EEP	EnOcean Equipment Profile
EHS	European Home System
EIA232	Electronic Industries Association standard RS-232-C

EIB	European Installation Bus
ES	Embedded System
ETS	Engineering Tool Software
FB	Function Block
FPGA	Field-Programmable Gate Array
FR	Functional Requirement
FV	Formal Verification
GCC	GNU Compiler Collection
HA	Harvard Architecture
HAL	Hardware Abstraction Layer
HMAC	Hash Message Authentication Code
HPAI	Host Protocol Address Information
HVAC	Heating, cooling, Ventilation, and Air Conditioning
IAC	Inspection And Certification
ICD	InterConnection Device
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IRAM	Industrial Risk Assessment Map
I/O	Input/Output
IP	Internet Protocol
IPv4	Internet Protocol version 4
IT	Information Technology

Java ME	Java Micro Edition
JVM	Java Virtual Machine
LNS	LonWorks Network Operating System
LON	Local Operating Network
MAC	Message Authentication Code
MCU	MicroController Unit
MD5	Message-Digest Algorithm 5
MD	Management Device
MDP	Management Datapoint
NV	Network Variable
NX	No eXecute
OR	Organizational Requirement
OSI	Open Systems Interconnection
OS	Operating System
OWL	Web Ontology Language
PCC	Proof-Carrying Code
PDP	Physical or Process Datapoint
PEI	Physical External Interface
PLC	Programmable Logic Controller
PP	Physical Partitioning
RDFS	Resource Description Language Schema
RF	Radio Frequency

RLC Rolling Code

RORG Radio Organizational Number

SAC Sensor Actuator and Controller

SB Sandbox

SCA Static Code Analysis

SCADA Supervisory Control and Data Acquisition

SCC Self Checking Code

SCPT Standard Configuration Property Type

SFPT Standard Functional Profile Template

SIDS Signature based Intrusion Detection System

SMT Software Monitoring Techniques

SNVT Standard Network Variable Type

TCP Transmission Control Protocol

TP-UART Twisted Pair Universal Asynchronous Transceiver

UDP User Datagram Protocol

UI User Interface

USB Universal Serial Bus

VAES Variable Advanced Encryption Standard

VM Virtual Machine

VTS Visual Test Shell

VPN Virtual Private Network

WAN Wide Area Network

WM Watermarking

XD eXecute Disable

1 Introduction

1.1 Motivation

A Building Automation System (BAS) aims at improving control and management of mechanical and electrical systems in buildings – more generally, interaction among all kinds of devices typically found in those smart homes and buildings. The core application area of BASs is environmental control handled by the traditional services Heating, cooling, Ventilation, and Air Conditioning (HVAC) and lighting/shading. Other application specific services are often implemented by separated systems. This is especially true for safety-critical (e.g. fire or social alarm systems) and security-critical (e.g. intrusion alarm or access control systems) services, which are typically provided by proprietary standalone systems. Today, if at all, only loose coupling is implemented with the core BAS for visualization, alarm management and trending [62].

The benefits of deploying BASs are manifold. While improved building management, sustainability and reduced operational costs are the dominant reasons for integrating modern BASs in buildings, increased comfort and e-health applications including telemonitoring and telecare are becoming possible in smart homes. Especially the latter can be seen as a new strategy to cope with the upcoming social and economic challenges in industrial nations towards an aged population and a decline in birth rates due to the demographic trend. Such Ambient Assistive Technologies – with smart homes being an integral part of them – can provide support to elderly or disabled persons, foster their autonomy and quality of life and lower the cost of medical care. The additional benefits concerning energy saving and energy efficiency apply to both applications areas – smart homes as well as buildings.

To activate all inherent synergies among these diverse systems and enable such new application areas, an integration – also of security and safety-critical systems – is a major topic. Consider, for example, the possibility of sharing the data originating from just one sensor in multiple application domains in parallel. This will reduce investment and maintenance costs and also facilitate management and in particular configuration of the integrated BAS as now a multitude of different applications can be substituted for a unified view and a single central configuration access point.

Obviously, the demands on a more sophisticated BAS controlling different subsystems increase. This is especially true for the integration of security-critical services. They depend on the underlying control system and its software to be reliable and robust against malicious manipulations to fulfill their purpose. Integration also makes way for further improvement of the traditional services. For example, although often not regarded as security-critical, incorporating security countermeasures for HVAC and lighting/shading services will prevent, among others, vandalism acts. The economic impact of a company-wide shutdown of the lighting system can be easily compared to a successful attack on the company Web server – the only difference being that, for the Web server, elaborate security measures are already common practice. With further standardization and increasing use of BASs (possibly with remote connections) adversaries can be expected to target unprotected building control systems. Thus, it is required to set up schemes for BASs providing protection as powerful as those already established in the Information Technology (IT) world.

However, directly applying those IT security concepts to BASs is impossible due to different requirements and resource limitations of the devices involved. To provide comparable protection, the development of new security concepts (although possibly based on a redesign of IT mechanisms) is strictly necessary. The ultimate goal is a secure system architecture for BASs applicable for the use in their core areas and tightly integrated environments [62].

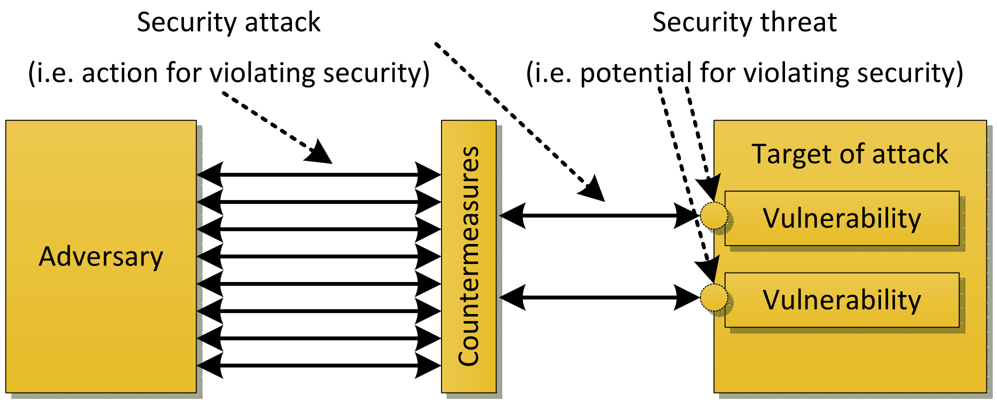


Figure 1.1: Security Attacks

1.2 Security in Building Automation Systems

To be able to integrate security-critical services, the implemented control functions, i.e., functions that control the building automation services, have to be protected against unauthorized access and malicious interference (security attack). A typical example of such a security attack is the manipulation of an access control system that opens and closes an entrance door. To perform security attacks, the malicious entity (adversary) has to identify vulnerabilities of a system that can be utilized to gain unauthorized access to the control functions. The existence of vulnerabilities leads to a security threat which can be regarded as the potential for violation of security that may or may not be utilized. Figure 1.1 shows the relation between these basic security terms.

On the one hand, the protection of a system against security attacks demands that the amount of vulnerabilities is minimized by incorporating security in every stage of the system's life cycle, especially already during design [90]. On the other hand, countermeasures that eliminate or prevent security threats and attacks in advance have to be implemented. For example, the encryption of transmitted data can be used to avoid a disclosure of confidential information. In cases where a prevention is not possible with reasonable effort, mechanisms have to be deployed trying to at least detect security attacks, report them, and minimize the resulting damage. Consider, for example, an Intrusion Detection System (IDS) that reveals abnormal system behavior and tries to prevent a propagation by isolating the source of the attack.

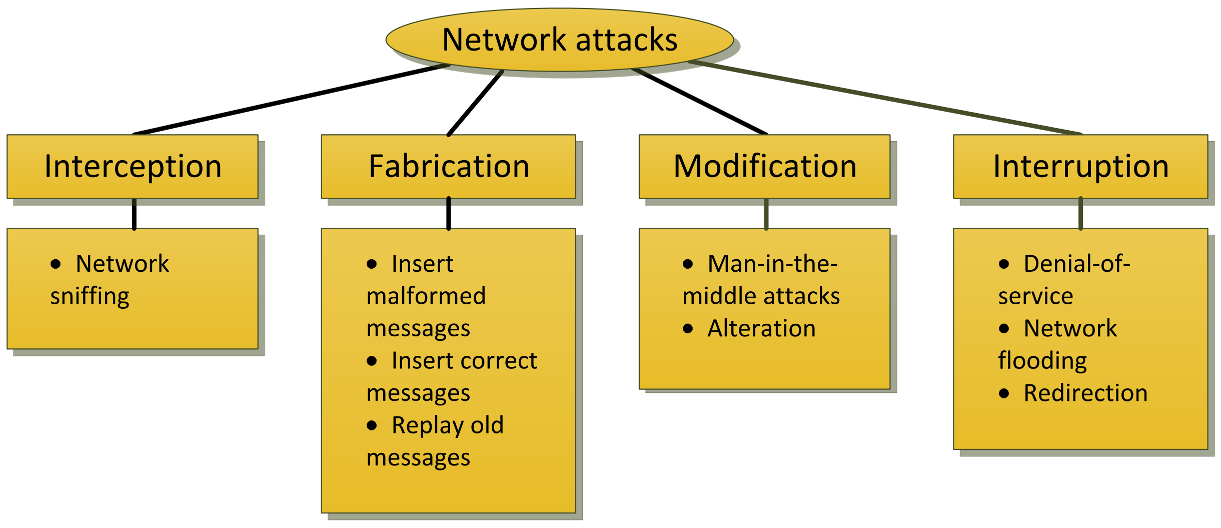


Figure 1.2: Network Attacks

In today's BASs, the control functions are distributed to Control Applications (CAs) being hosted on different devices interconnected by a common network. Unauthorized access by an adversary to control functions can be gained by, on the one hand, directly manipulating the devices (e.g. changing the control logic or modifying control data such as an output value) or, on the other hand, by indirectly changing control parameters or interfering with the data exchanged among the CAs. Therefore, the devices themselves as well as their communication means have to be secured [62].

1.2.1 Secure Communication

An adversary may attack the network medium to access the exchanged data and thus interfere with the data when they are transmitted (network attacks; cf. Figure 1.2). According to [126], an adversary may try to intercept, manipulate, fabricate, or interrupt the transmitted data. Access to the network medium can be achieved in two ways.

- Medium access: The adversary gains physical access to the network medium. This can be accomplished more easily when open communication technologies (e.g. Internet Protocol (IP), Radio Frequency (RF) or powerline networks) are used.
- Device access: The adversary can use the network interface of another compromised device (e.g. a Web gateway).

1.2.2 Secure Devices

An adversary may attack a device to access control functions (device attacks; cf. Figure 1.3). These attacks can be classified based on the means used to launch them [77], [136].

- Software attacks: An adversary may use regular communication channels to exploit weaknesses in a device's software.
- Side-channel attacks: An adversary may observe external (device) parameters which are measurable during operation to collect information about internals.
- Physical or invasive attacks: An adversary may use physical intrusion or manipulation to interfere with a device.

Physical and side-channel attacks can either be invasive, typically being hard to employ due to expensive hardware costs, or non-invasive targeting timing, power and electromagnetic analysis, and fault induction. Software attacks form a massive challenge for the security engineering effort. The list of possible issues is extremely broad, with the most common flaws being buffer overflows, memory corruption errors, code update processes, insecure algorithms, cryptographic flaws, insecure key management or operating system weaknesses. Additionally, the execution of basically trusted software, being hijacked by an adversary or containing undetected flaws (e.g. virus, trojan, worm) often causes security leaks or may result in Denial of Service (DoS) attacks.

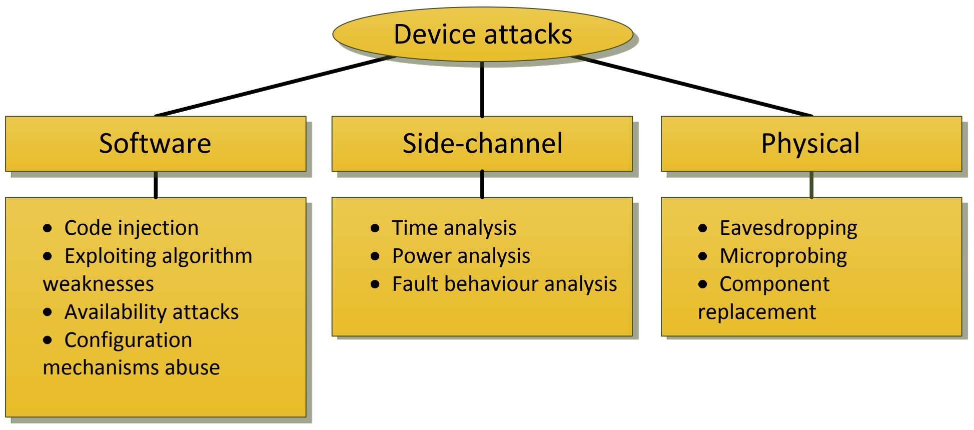


Figure 1.3: Device Attacks

1.3 Problem Statement and Hypothesis

In order to provide secure BASs, comprehensive measures need to cover communication as well as device security. Mechanisms tailored to the use in Building Automation Networks (BANs) that counteract network attacks are presented in [58]. 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 [104]. No scientific research is available that targets software attacks in the BAS domain. This thesis focuses on the research question, how control applications for smart homes and buildings can be secured. The following hypothesis will be discussed and proven throughout the work:

Hypothesis 1 *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 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.*

Figure 1.4 briefly describes the security process 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 building operator finally maintains the building. Basically, security is essential in all steps, for all stakeholders and during the complete lifetime. This dissertation describes the process of providing secure CAs (blue markings in Figure 1.4): Starting from secure CA development, committees are supported in standardizing mechanisms, frameworks and generic policies, while CA manufacturers are supported in creating CAs. System integrators and building operators are supported in adapting security policies during CA operation.

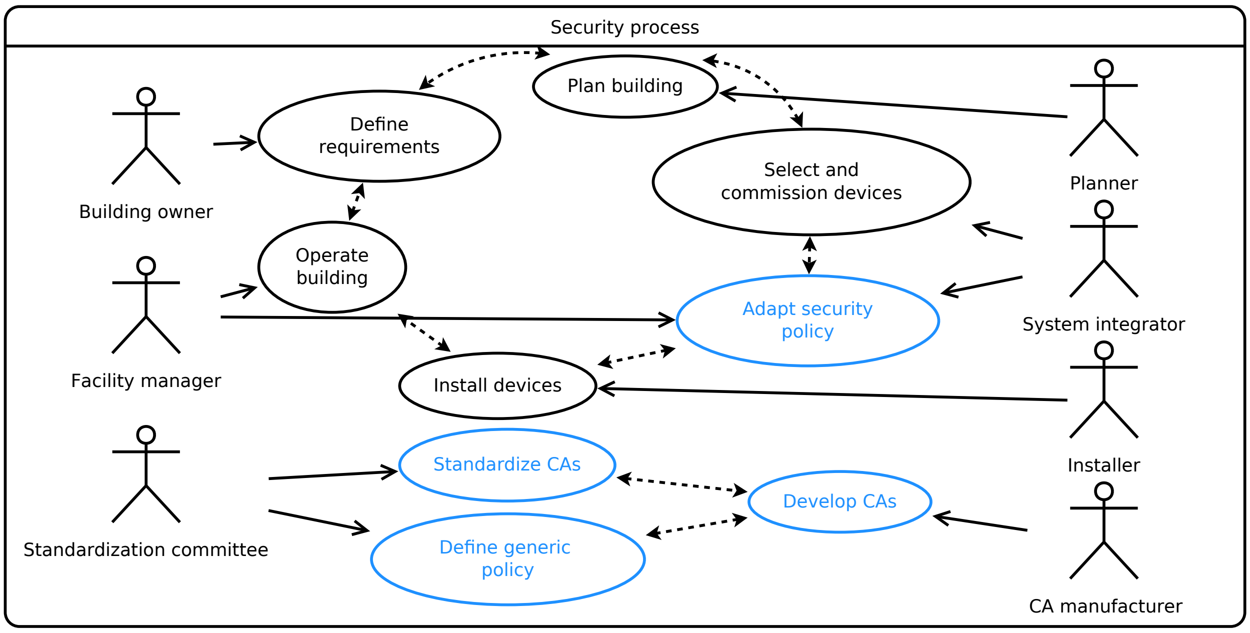


Figure 1.4: Security Process and Involved Stakeholders

1.4 Methodology and Organization

The following methodology has been applied during this work. To motivate the research question, first a security analysis of today's BAS technologies and installations has been performed. Then, functional and non-functional requirements for secure CAs have been defined and the state-of-the-art has been examined by literature search with respect to its applicability. Since the requirements can only be met partially, a secure CA architecture has been investigated. The research result is an approach to provide secure CAs. Validation could be done in various ways: First, it could be compared to other concepts and be analyzed with respect to improvements. Since neither concepts nor implementations for secure CA development are available in the BAS domain no benchmarking can be performed. Second, the generic technique could be implemented and be validated using standardized security tests (e.g. CIS-CAT and Other CIS Benchmark Assessment Tools[1]) or during security competitions (e.g. Competition on Software Verification (SV-COMP)[2]). However, no tests or competitions exist in the BAS domain and additionally such methods can only validate implementations and not generic concepts. Simulation

[1] `https://benchmarks.cisecurity.org/downloads/audit-tools/`, Last access: 2015/09/18

[2] `http://sv-comp.sosy-lab.org/2016/`, Last access: 2015/09/18

of the concept could validate its applicability, however it is not feasible for security requirements. Thus the presented approach has been validated using various orthogonal analyses. On the basis of a use case, prototypes have been implemented and performance benchmarks have been performed. An accompanying discussion shows how the security process is carried out and can be used 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.

The thesis is structured as follows. Each section starts with a hypothesis covering parts of secure CAs in smart homes and buildings, that is being discussed throughout the section. In a summary this hypothesis is being proved.

Section 2 analyzes domain constraints in distributed CAs and the structure of modern BASs, their devices and application models. Then the definition of CAs in related standards as well as their design, development and deployment in open BASs (i.e. BACnet, EnOcean, KNX, LonWorks, ZigBee) are investigated.

To demonstrate that software security is currently being strongly neglected, a deep analysis on CA security in current building automation standards and technologies is performed by researching security mechanisms being deployed in installations and by identifying vulnerabilities being present. An additional exemplary evaluation on installed CAs based on EnOcean shows the rather high probability of a successful attack. The problem of insecure CAs is spread in the whole smart home and building automation domain, irrespective of deployed technology or manufacturer. Based on a software security threat and risk analysis, security requirements are identified (cf. Section 3).

As a next step, a state-of-the-art analysis is performed. Existing software protection techniques are investigated and security in open BAS standards is analyzed. This related work is evaluated for BASs with respect to its applicability and implied security gain regarding the discussed threats and requirements (cf. Section 4).

The main contribution of this dissertation is a comprehensive concept and process for secure and distributed CAs, describing how to fulfill the demands for security-critical smart homes and buildings. This secure CA architecture consists of a model as well as a mapping to common technologies of an integrated BAS, being capable to describe secure CAs in a generic way. A formal method to define a security policy based on security attributes for a whole BAS, a secure software environment to allow the execution of CAs and the enforcement of the policy, and an attack prevention and detection system are investigated (cf. Section 5).

The evaluation is described in Section 6. A use case is defined and prototypes are implemented for the different device classes. Performance measurements and an accompanying discussion on how the security process is working demonstrate that the generic concept can be implemented on today's devices. Secondly, it is discussed how the presented approach can be used to enable security in current BASs. Attack detection and prevention become possible, if appropriate devices are installed. Concluding, an exhaustive discussion evaluates that all functional requirements are fulfilled by applying the developed framework. Some organizational requirements still need to be considered, when implementing real live installations.

Finally, recommendations for implementing secure CAs are formulated (cf. Section 7).

Parts of this dissertation were already published in the following peer-reviewed publications:

- W. Granzer, W. Kastner, G. Neugschwandtner, and F. Praus, "A Modular Architecture for Building Automation Systems", in *Proc. 6th IEEE International Workshop on Factory Communication Systems (WFCS '06)*, Jun. 2006, pp. 99–102
- W. Granzer, W. Kastner, G. Neugschwandtner, and F. Praus, "Security in Networked Building Automation Systems", in *Proc. 6th IEEE International Workshop on Factory Communication Systems (WFCS '06)*, Best Paper Award of WFCS '06, Jun. 2006, pp. 283–292
- F. Praus, T. Flanitzer, and W. Kastner, "Secure and Customizable Software Applications in Embedded Networks", in *Proc. 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08)*, Sep. 2008, pp. 1473–1480
- C. Reinisch, W. Granzer, F. Praus, and W. Kastner, "Integration of Heterogeneous Building Automation Systems Using Ontologies", in *Proceedings of 34th Annual Conference of the IEEE Industrial Electronics Society (IECON '08)*, Nov. 2008, pp. 2736–2741

- F. Praus and W. Kastner, "User Applications Development Using Embedded Java", in *Proc. KNX Scientific Conference*, Nov. 2008
- F. Praus, W. Granzer, and W. Kastner, "Enhanced Control Application Development in Building Automation", in *Proc. 7th IEEE International Conference on Industrial Informatics (INDIN'09)*, Jun. 2009, pp. 390–395
- F. Praus and W. Kastner, "Secure Control Applications in Building Automation Using Domain Knowledge", in *Proc. 8th IEEE International Conference on Industrial Informatics (INDIN'10)*, Jul. 2010, pp. 52–57
- W. Granzer, F. Praus, and W. Kastner, "Security in Building Automation Systems", *IEEE Transactions on Industrial Electronics*, vol. 57, no. 11, pp. 3622–3630, Nov. 2010
- F. Praus and W. Kastner, "Identifying Unsecured Building Automation Installations", in *Proc. 19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'14)*, Sep. 2014
- F. Praus and W. Kastner, "Spotting Unsecured KNX Installations", in *Proc. KNX Scientific Conference*, KNX Scientific Award 2014, Nov. 2014

Nevertheless, this dissertation is submitted as monograph, describing secure CAs in smart homes and buildings in a comprehensive manner.

2 Control Applications in Building Automation Systems

Hypothesis 2 *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.*

This section is dedicated to the definition of BASs and the classification of devices typically being found there. Besides, an identification of how to model distributed CAs in a general way and an analysis of application models in existing technologies are covered.

2.1 Device Classes

The system functionality of BASs can be divided into a three level model being ordered hierarchically [85]. At the field level, environmental data are measured and parameters of the environment are physically controlled. Automatic control is performed at the automation level whereas global configuration and managements tasks are realized at the management level. For years, the levels of the functional model have been mapped to separate networks when BAS had to be implemented. Sensors and actuators were interconnected via field networks. Controllers (e.g. Direct Digital Controls (DDCs)) responsible for dedicated process-oriented and time-dependent sequential control were combined via automation networks. Finally, servers and workstations hosting, for instance, applications for trend logging and visualization were linked by a management network. Of

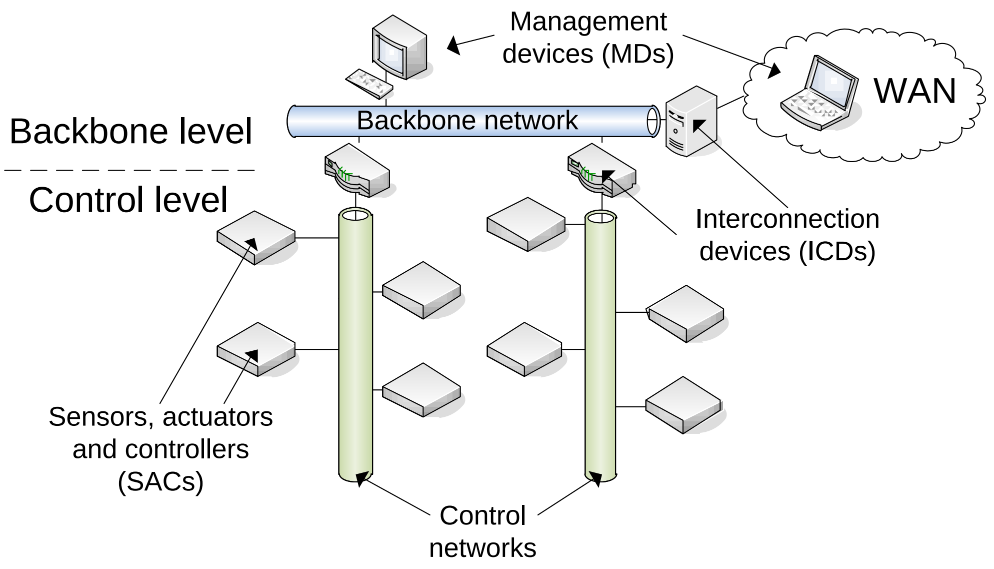


Figure 2.1: Building Automation Network

course, vertical access for data exchange from the lowest to the highest level had to be provided.

Nowadays, the standard three level functional hierarchy model can be implemented as a flatter, two-tier architecture (Figure 2.1) [96]. This is for two reasons. First, so called intelligent field devices incorporate more functionality than ordinary ones. Second, IT and its infrastructure became accepted not only at the management level, but also at the automation level. This is due to the fact that functions of the former automation level can be realized with cheap IT hardware since environmental conditions in the building automation domain are not that harsh compared to industrial automation. Consequently, functions of the former automation level are split, being reassigned either to field devices (e.g. implementing controller functionality) or management devices (e.g. realizing process data monitoring) [60].

As a result, the two-level architecture consists of a control network and a common backbone which together form the BAN. The control network is home to field devices and has a typical bandwidth in the order of a few KBits/s. Since the requirements of management devices still cannot be fulfilled by this control network (e.g. a global consistent view of the entire system needs higher data rates), control networks are interconnected via a high-bandwidth (typically MBits/s) backbone network. At the intersection point between the networks, interconnection devices are used. Based on this novel view on a BAN, BAS devices and their functionality have to be re-classified and the following demands have to be considered [60]:

- Resources: The required functionality of a BAS device significantly influences the demands on processing power and memory.
- Interfaces: Different interfaces are desirable:
  - *Network interfaces* to integrate the device into a BAN.
  - *Point-to-point interfaces* to perform configuration tasks.
  - *Process interfaces* for data acquisition and interacting with the physical environment.
  - *Human machine interfaces* for user interaction.
- Power consumption: If a device is driven by battery or by a common power supply (e.g. via link power), a low power consumption is needed.
- Environment: Since BASs shall be operable for many years, the used devices shall be insensitive to rough installation environments.
- Costs: Device installation as well as maintenance costs need to be considered to keep a BAS – probably consisting of thousands of devices – affordable.

Sensors, Actuators 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. Additionally, they may include controller functionality. The following requirements are crucial for SACs:

- Resources: As SACs have to perform more or less simple tasks, small and low cost MicroController Unit (MCU) based devices are sufficient. The memory requirements are relaxed since the amount of process data (in the order of bytes to a few KBytes) to be handled can be assumed to be small.

- Interfaces: In order to configure and maintain a SAC device (e.g. upload new firmware), a simple point-to-point interface (e.g. serial communication) has to be provided. To exchange process data, a network interface is required. Compared to industrial automation, the demands on the response time[1] are more moderate but they nevertheless exist (e.g. in the order of a few milliseconds; cf. [96]). Since wireless networks are gaining importance in BASs, such interfaces have to be taken into account as well. For SACs, a process interface for interaction with the physical environment is mandatory.

- Power consumption: SACs are often supplied via link power to avoid the need for an additional power cable. In some cases, SACs may be even driven by a battery (e.g. glass break sensors). Here, low power consumption is of major concern.

- Environment: As SACs are usually located in the field, they have to be robust and small.

InterConnection Devices (ICDs) provide an interconnection between network segments or remote access to foreign networks. They operate at different layers of the Open Systems Interconnection (OSI) Reference Model. To extend the maximum physical network cable length, repeaters and bridges can be used acting at the physical and data link layer, respectively. While a router operates on protocols at the network layer, a gateway to e.g., a Wide Area Network (WAN), ensures transparency to applications that run on top of the protocol stack. The demands on ICDs are the following:

- Resources: Compared to SAC devices, the requirements on ICDs are quite similar. Since routing is not a very complex task, MCUs are sufficient to fulfill the processing power requirements of ICDs. However, routers and gateways may need additional memory for storing routing tables or caching data.

- Interfaces: To configure and manage ICDs, some sort of local interface (e.g. Universal Serial Bus (USB)) is required. The main objective of ICDs is the interconnection of two or more network segments or networks. Therefore, ICDs need at least two (possibly different) network interfaces.

[1] The term response time is referred to as the time interval between action initiation (e.g. pressing a light switch) and action execution (e.g. light on).

- Power consumption: Since battery driven ICDs are uncommon, low power consumption is not as important as it is for SACs.

- Environment: ICDs will normally be located at central points in the building (e.g. in a switch cabinet). However, it is still important that ICDs are small and maintenance-free.

Finally, Management Devices (MDs) are used to configure and maintain a BAS. Typically, MDs execute configuration (e.g. set initial configuration parameters), maintenance (e.g. changing setpoints), and operator tasks (e.g. visualization, alarm monitoring, security alert monitoring). In most cases, representatives of the MD class will be located at the backbone level. Still, some representatives may be found at the control level (e.g. for on-site device configuration). Since MDs are often controlled by humans, they provide some kind of (graphical) User Interface (UI). Requirements on MDs are quite different compared to SACs and ICDs:

- Resources: MDs have to operate with data from the whole BAS and therefore are supposed to process higher data volumes (in the order of KBytes to MBytes). Additionally, management tasks require more processing power and (persistent) storage for software and data.

- Interfaces: To be able to fulfill configuration and management tasks, a connection to the BAN is mandatory. Such an interconnection can be achieved using so-called network adapters (e.g. SAC with a serial connection). The demands on the response time can be seen more relaxed since management tasks are not time critical. Additionally, interfaces for various UIs have to be provided.

- Power consumption: Since server, workstation and PC-based MDs are supplied via the power grid, power consumption is not of major concern.

- Environment: MDs with fully fledged UIs will be normally located in moderate environments (offices). Therefore, small size and robustness against rough environmental conditions are less important. For other MD representatives, relaxed environmental conditions can be expected as well.

Figure 2.2 shows, from an abstract point of view, the different building blocks which have to be assembled to provide the functionality for each device class. By selecting the corresponding soft- and hardware blocks, SACs, ICDs or even MCU-based MDs can be implemented.

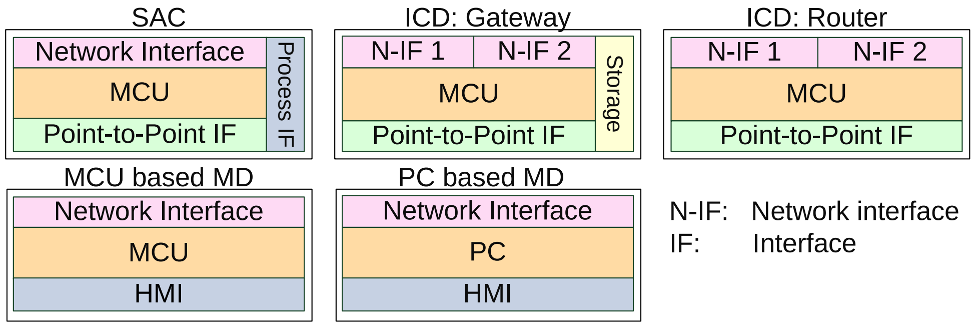


Figure 2.2: Building Automation System Hardware Building Blocks

2.2 Distributed Control Applications

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 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 [142]. Allowing CAs to be uploaded in the field has benefits for configuration and maintenance. Network adapters and application specific hardware can be procured separately, and software updates can be applied after installation.

Definition 1 *Control Application: A CA is a configurable software being executed on a SAC with the purpose to control a process at the control level.*

Definition 2 *Distributed Control Applications: 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) (cf. Figure 2.3). 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.

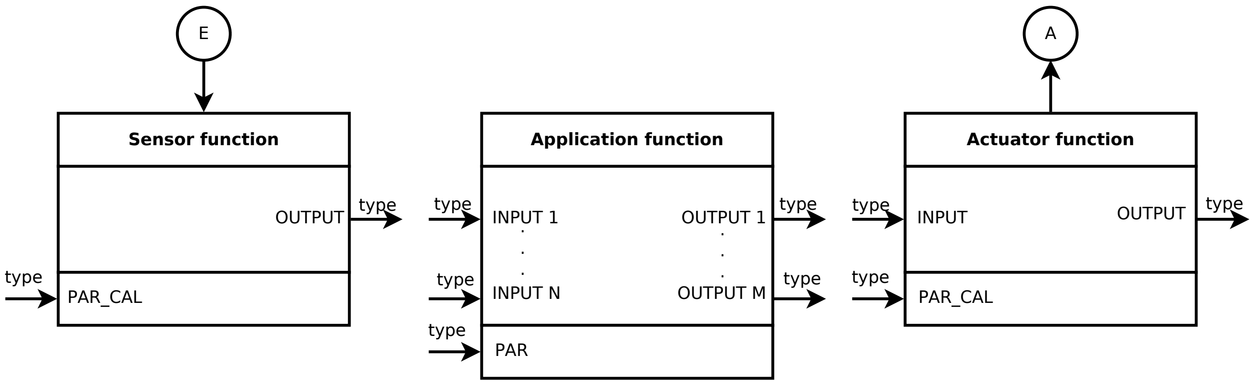


Figure 2.3: Function Blocks (based on [163])

The necessary vendor-independent compliance (i.e. interoperability) does not only have to be guaranteed for communication but also for these distributed CAs, so that devices of different vendors can be deployed in one system and broader acceptance of advanced trade neutral applications for e.g. energy optimizations is achieved. Interoperability can only be met by standardizing the application models, which state how data is represented (data format, encoding) and how the communication between CAs has to be realized (methods to manipulate data). Today, efforts are underway to lay down the functions of the building automation and required BAS domain knowledge within the following (international) standards [132].

Among defining terms and discussing basic considerations regarding application software, ISO 16484-3 [86] specifies the functions of building automation and control systems. It distinguishes between input/output functions (e.g. binary in/output, analog in/output), processing functions (e.g. monitoring, open loop controlling, closed loop controlling, calculating/optimizing), management functions (e.g. trending), and operator functions, and clearly describes *what* these functions ought to do. For description it relies on FBs. The method for programming functions and applications, however, is not part of ISO 16484-3.

Closely related to ISO 16484-3 is its predecessor VDI 3814 [164], which is partly obsolete, but additionally defines the requirements usually present in middle Europe including legal restrictions.

VDI 3813-1 [163] describes a subset of building automation – the fundamentals of room control. It defines the terms and functions of room automation independent of the underlying technology on the basis of use cases. It further provides a thorough system analysis and classification of room control functions into application functions (e.g. lighting control, temperature control), display/operator functions (e.g. light switch, temperature setpoint), service/diagnostic functions (e.g. self-diagnostics, software downloading), and management functions (e.g. trending, energy analysis).

VDI 3813-2 [165] extends Part 1 and specifies the methodology to describe functions of room automation. Using FBs and a textual description, it outlines the functionality (i.e. inputs, outputs, parameters, physical in/outputs) of an object.

Experiences with VDI 3813-1 and VDI 3813-2 revealed that an exact specification on how to combine those functions of room automation is missing. Thus, VDI 3813-3 [162] specifies room automation function macros and application examples of room types.

VDI 3805 [166] describes the open product data exchange in building services between different manufacturers or trades. Details such as product data, technical data and geometry data are covered and their description is standardized in a machine processable way. Although this standard is not directly related to CA development or CA security, important domain knowledge (e.g. optimal/minimal/maximal volume flow of a fan) can be extracted out of the technical data section.

With regard to system modeling, the process control domain is similar to BAS. Application structure and development has been covered in multiple standards.

The conceptual FB specification for process control is described in IEC 61804-2 [82]. The device model as well as specifications of FBs for measurement, actuation and processing are contained and can be mapped to specific communication systems and implementations. The IEC 61804-3 Electronic Device Description Language standard presents a generic language for the description of the properties of automation system components. Entities such as device parameters and their dependencies, device functions or interaction with control devices can be modeled.

Quite similar, IEC 61499-1 [83] defines a nomenclature and reference model and specifies the concept of FBs with the big difference that an IEC 61499-1 FB is an IEC 61804 FB with execution control, and event inputs and outputs being added.

IEC 61131-3 [84] defines the syntax and semantics of programming languages for programmable logic controllers. The software model, communication model as well as the programming model are covered.

To be able to describe the security concept presented in this dissertation in a better way, the following use case $\otimes$ is defined according to VDI 3813-3 and will be used throughout the remaining sections. Figure 2.4 shows a minimalistic light actuator with delayed on/off behavior being used for a stairwell light.

Use case $\otimes$ *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`.*

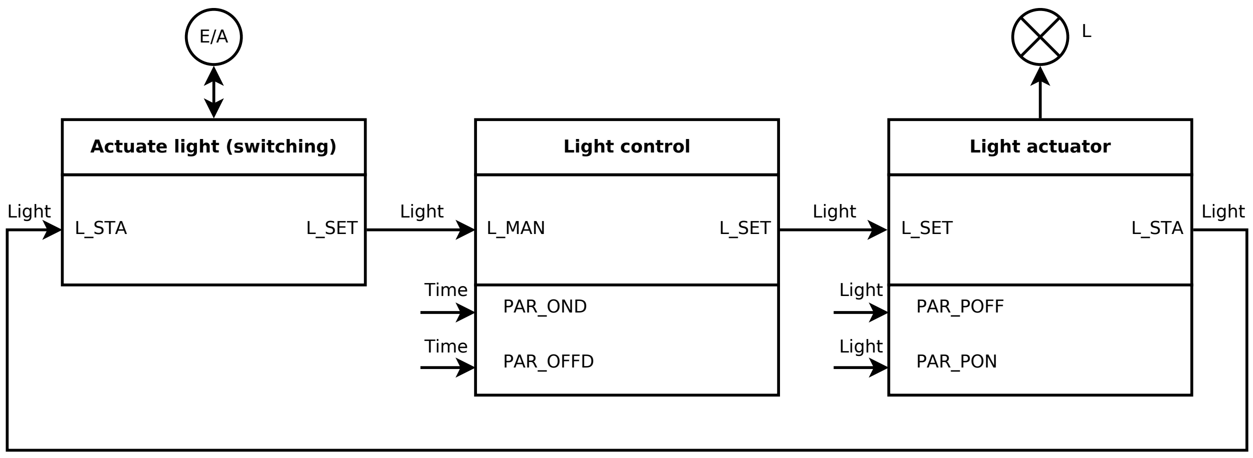


Figure 2.4: Use Case $\otimes$: Distributed Control Application: VDI 3813-3: D-1-2 Lighting Control Manual with Time-Controlled Switching Off (Stairwell Light)

2.3 Application Models

While the previous section considered CAs according to generic definitions, this section gives an overview of application models and CA development in current BAS technologies. Besides, application security aspects briefly including also communication are analyzed. Only the most important open BAS protocol standards BACnet, EnOcean, LonWorks, KNX and IEEE 802.15.4/ZigBee, which span more than one application domain are considered for the rest of this thesis. In fact, detailed information about proprietary standards such as Z-Wave [183] is not publicly available and BAS for special purposes (e.g. DALI for lighting [81], M-BUS for smart meters [25], and Modbus mainly used for accessing Programmable Logic Controller (PLC) memory [118]) do not cover all BAS application areas of interest.

2.3.1 BACnet

Building Automation and Control networking protocol (BACnet) was developed by a project committee established by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE). Its main objective is to provide a solution for BAS of all sizes and types. In 1995, BACnet was published as ANSI/ASHRAE 135 standard [7]. Meanwhile, BACnet has become the CEN and ISO standard 16484 [87], [91].

BACnet defines the network visible part called BACnet object of a single data element. The internal data structure is not covered and thus also the CA itself and its development are not specified. Each BACnet object has a dedicated object type and represents a collection of properties. Each property has a data type defining the size and encoding of the data element.

BACnet-2012 defines 54 different objects and more than 200 different property types as well as object access services. The most important services used to access and manipulate objects are `ReadProperty` (i.e. read the value of a property), and `WriteProperty` (i.e. set a new property value). Example objects are the generic binary and analog object types such as `BACnet Binary Output Object Type` and `BACnet Analog Input Object Type`. It is the responsibility of the CA to map the functionality of a simple light to a `BACnet Binary Output Object Type` or a room temperature value to a `BACnet Analog Input Object Type`. However, there are efforts underway to standardize more application-specific object types in BACnet.

The `Lighting-Output` Object Type, for instance, has been included in the latest standard (cf. Table 2.1). Issuing a `WriteProperty` request on the `Present-Value` of a device implementing the `Lighting-Output` Object Type with a BACnet compliant tool, a connected light can be switched on and off. If the parameter `Off-Delay` is set appropriately, a stairwell light can be realized.

Property Identifier	**Property Datatype**
Object-Identifier	BACnetObjectIdentifier
Object-Name	CharacterString
Object-Type	BACnetObjectType
Present-Value	REAL
Progress-Value	REAL
Lighting-Cmd	BACnetLightingCommand
Blink-Time	Unsigned
Off-Delay	REAL OPTIONAL
Blink-Prior.-Threshold	Unsigned OPTIONAL
Status-Flags	BACnetStatusFlags
Out-Of-Service	BOOLEAN
Priority-Array	BACnetPriorityArray
Relinquish-Default	REAL
Priority-Array	BACnetPriorityArray
Relinquish-Default	REAL
Lighting-Cmd-Prior.	Unsigned

Table 2.1: BACnet `Lighting-Output` Object Type

BACnet optionally uses the Advanced Encryption Standard (AES), the Keyed-Hash Message Authentication Code (HMAC) algorithm, and a message ID in combination with a time stamp to protect against unauthorized interception, modification, and fabrication of data exchanged via the communication medium. Furthermore, security concepts like a trusted key server being responsible for generating and distributing session keys and the use of different key types and key revisions are used.

2.3.2 ISO/IEC 14543-3-10/EnOcean

EnOcean is a wireless BAS standard for solutions with ultra-low power consumption and energy harvesting. It has been developed by the EnOcean GmbH, which has been founded in 2001 as a spin-off from Siemens AG. The company holds the patents for its energy harvesting technology and manufactures the communication modules. In 2012, the ISO/IEC 14543-3-10 standard was released, standardizing the physical, data link and network layer of EnOcean [88]. Interoperability of applications is pursued by the non-profit organization EnOcean Alliance. It defines the EnOcean Equipment Profiles (EEPs), providing the data format, descriptions and procedure of device pairing [38].

EnOcean defines the communication protocol (i.e. the telegram format) for data exchange in BASs. CAs, their development, internal structures or deployment are not standardized. Depending on the telegram type, different applications (e.g. Light and Blind Control, Temperature Sensor) can be realized.

Since no application model is defined by the EnOcean standard, a stairwell light can be implemented in various ways. Using a standard rocker switch, `EEP RPS F6-02: Rocker Switch, 2 Rocker` telegrams [38] are transmitted when buttons are pressed. Compatible actuators receive these telegrams and switch their outputs accordingly. Using a manufacturer dependent local configuration of the switching actuator (e.g. via DIP switches), a stairwell light can be realized. [38] defines the `EEP 4BS A5-38-08: Telegram-Central Command-Gateway` telegram. Using the command `0x01: Switching`, a delay value can be set to execute the switching command after a certain delay.

Until 2012, EnOcean did not provide any security mechanisms. In March 2012, version 1.0 of the Security of EnOcean Radio Networks specification [43] was released, which covers security features targeting the very-low energy requirements of self-powered devices. The current specification version 1.9 provides secure communication. The algorithm AES-Cipher Based Message Authentication Code (CMAC) is used for authentication, AES-Cipher-Block Chaining (CBC) and Variable Advanced Encryption Standard (VAES) for en/decryption.

In September 2013, three application notes have been released. [39] describes how to implement security in CAs and the required security tasks for secure communication in transmitters and receivers. [40] covers security in line-powered devices (i.e. gateways, actuators). Secure CA development and a secure architecture with the use of EnOcean Link [42] a middleware for the EnOcean protocol stack are described. The EnOcean Link documentation contains a security example for secure communication and a security watcher example, which can be used to detect possible security issues. [41] describes how to include security into self-sustaining applications (i.e. sensors).

2.3.3 KNX

The KNX standard [103] – being maintained by the KNX Association – describes an extensive open system concept for distributed home and BASs. KNX covers the full scope of related applications, including lighting, shading, shutters and blinds, HVAC and remote meter reading. The KNX specification results from a formal merger of three technologies dedicated to this area: European Installation Bus (EIB), Batibus, and European Home System (EHS). Significant parts of the KNX specification are published as European standards EN13321 and EN50090 [74], [123], [124] but KNX technology is also covered by an international standard ISO/IEC 14542-3 [89] as well as GB/Z 20965 [23] published by the Standardization Administration of China.

KNX is more than just a simple network protocol specification. Besides merely defining how data is transferred, the standard includes rules and definitions of how a KNX system is managed, how devices can be implemented by different vendors and how they have to behave to achieve interworking. The central hardware concept are customized application modules being connected via the Physical External Interface (PEI) to standardized Bus Attachment Units (BAUs). Certainly, the manufacturer dependent CAs are the corresponding software components within a KNX system. KNX supports various standardized hardware interfaces. The first time creation of CAs for Bus Coupling Unit (BCU) 1, Bus Interface Module (BIM) M111, BIM M115, BCU 2 and BIM M113 based devices, however, remains quite complex since the hardware does not provide enough memory for advanced CAs. No toolchain is available that offers high level programming languages (e.g. C, C++) and the developer has to consider MCU specific issues (e.g. RAM flags). The new generation of BIMs (NEC 78K0/Kx2 based) overcomes the hardware limitations and allows development in the C programming language and debugging using the IAR Embedded Workbench. For development of CAs on a separate MCU, a Twisted Pair Universal Asynchronous Transceiver (TP-UART) chip can be used, which handles the physical and most of the data link layer tasks. Besides, various companies provide KNX compliant hard- and software such as evaluation boards, devices, modules, stacks and software (e.g. TAPKO Technologies GmbH[2], Weinzierl Engineering GmbH[3]). The central management software ETS then allows customization and adaptation of most KNX compliant CAs to a specific installation by changing parameters.

KNX defines application models within the different application domains (e.g. HVAC, lighting). Their functionality is distributed across functional blocks which are described by a well-known behavior and consist of one or more datapoints. A functional block is implemented by a single device, and each device can host multiple functional blocks. A single datapoint (i.e. input, output, parameter that influences the behavior of the functional block) represents a single data of the application and has a defined Data Point Type (DPT) which states its format (e.g. bit length), encoding, range (e.g. upper and lower bound of the value) and engineering unit (e.g. percent). Each datapoint can be accessed using a specific KNX communication service as well as a particular address (e.g. group object datapoints use group addresses and are accessible through group communication services).

The KNX function block `Light Actuator` defines the following datapoints [103, Part 7/20] (cf. Table 2.2). A push button can be used to issue an `A_GroupValue_Write` on the datapoint `OnOff`. The light actuator then switches off the light after the time configured by parameter `Timed Duration`. Feedback to sensors is given using the output `InfoOnOff`.

KNX did not offer mechanisms to guarantee confidentiality, integrity or freshness for process data being exchanged via the network until 2013. Neither did it support a dedicated authentication service. It only provided a basic access control scheme, which can

[2] `http://www.tapko.de/`
[3] `http://weinzierl.de/`

Data Point	**Data Point Type**
Output: InfoOnOff	DPT_Switch
Input: OnOff	DPT_Switch
Parameter: Timed Duration	DPT_TimePeriodSec

Table 2.2: KNX `Light Actuator` Function Block

be used to limit the management access to devices. Up to 255 different access levels can be defined, each of them associated with a different set of privileges. For each of these access levels, a 4 byte password can be specified. The security extension KNXnet/IP Secure [102] provides data integrity, freshness, confidentiality and mutual authentication for KNXnet/IP. Moreover, KNX Data Security [101] supports secure communication for all existing KNX communication media.

2.3.4 LonWorks

LonWorks or Local Operating Network (LON) was originally developed by Echelon Corporation as a generic open control network to support a wide range of distributed applications in various domains such as building automation, industrial automation, transport automation or street lighting. It consists of the LonTalk communication protocol, a controller (Neuron Chip), transceivers for bus access and management and development tools. In 1999, LonTalk was standardized as ANSI/EIA-709 and ANSI/CEA-709 standard [26]. It is also available as European standard EN14908 [122] and as international standard ISO/IEC-14908 [92]. These standards are accompanied by ANSI/EIA-852 [160], which describes tunneling of LonTalk over IP networks.

LonWorks nodes are usually based on a network controller (e.g. Neuron series, LoyTec LC3020), which executes the network protocol stack and the CA. CAs define so called Network Variables (NVs), which are basic communication objects and are shared by devices over the network. The CA may be implemented in Neuron C (a programming language similar to ANSI C), which represents NVs as standard C variables with the unique property that a data packet is automatically created and transmitted whenever the value of the NV changes or is automatically updated whenever a data packet has been received from the network.

For interoperability reasons, the LonMark Association[4] standardizes the NVs through the definition of Standard Network Variable Types (SNVTs). A SNVT is an exactly defined (e.g. encoding, physical unit), calibrated, filtered and linearized engineering value which allows its doubtless interpretation. Besides the SNVTs, also so called Standard Configuration Property Types (SCPTs) are defined and used to access configuration functions within a device (e.g. changing parameters). To satisfy the demands of special domains such as building automation, the LonMark Interoperability Association defines so called Standard Functional Profile Templates (SFPTs) for interoperability. These SFPTs are application-specific and include NVs and configuration properties, default values, and power-up behaviors.

To implement a light with delayed on/off behavior, three SFPTs are needed [112]. The `scene panel` selects a defined scene (cf. Table 2.3). The `scene controller` delays the switch off command (cf. Table 2.4). Finally the `lamp actuator` controls the light (cf. Table 2.5).

Network Variable	**SNVT Type**
Output: nvoScene	SNVT_scene

Table 2.3: LonWorks `Scene Panel`

Network Variable	**SNVT Type**
Input: nviScene	SNVT_scene
Input: nvoSwitch	SNVT_switch
SCPTdelayTime: nciDelayTime	SNVT_time_sec

Table 2.4: LonWorks `Scene Controller`

Network Variable	**SNVT Type**
Input: nviLampValue	SNVT_switch
Output: nvoLampValueFb	SNVT_switch

Table 2.5: LonWorks `Lamp Actuator`

LonWorks provides a four step challenge-response mechanism to counteract modification and fabrication attacks of network messages. A sender which intends to authenticate

[4] `http://www.lonmark.org/`

a transmission asserts the authentication bit of its message. Receivers reply with a 64 bit random number. The sender returns a 64 bit hash value calculated over the content of the message and the random number using a shared key. The receiver performs the same calculation and compares the results.

In addition to the basic LonWorks authentication mechanism, LonWorks/IP defines its own security mechanism, which uses Message-Digest Algorithm 5 (MD5) together with a shared secret to protect the data against modification and fabrication.

2.3.5 IEEE 802.15.4/ZigBee

IEEE 802.15.4 specifies a physical and a data link layer for communication in wireless personal area networks as an open standard [78]. ZigBee is based on IEEE 802.15.4 and standardizes a network and an application layer [180]. It aims at providing a simple, low-rate, low-power and cost effective protocol for RF applications.

ZigBee CAs are implemented by application objects that are distributed across the ZigBee devices, with one ZigBee device hosting a maximum of 240 application objects. Each application object hereby implements a specific functionality of the distributed application. Within the application object, the functionality is represented by so called clusters. A cluster is a collection of commands and attributes. While a single attribute of a cluster represents a single data of the process to be controlled (e.g. the state of a light), commands are used to manipulate these attributes as well as to initiate actions within the device. Therefore, clusters act as interfaces to the application objects.

The exact structure of the application objects and their associated clusters (including the specification of the clusters' attributes and commands) is not defined by the core specification. However, to enable interoperability between ZigBee devices, so called application profiles are defined. Application profiles target a specific application domain. They contain a set of logical device descriptions that define the functionality to be implemented. This functionality is represented by clusters, whose implementations can either be mandatory or optional. An application object is thus an implementation of a logical device description (or at least of all its mandatory clusters) within a physical ZigBee device, e.g. an `On/Off Light`.

An example of such an application profile is the ZigBee Home Automation Profile [179] targeting applications typically found in residential or light commercial environments. The `Dimmable Light` device can be used to realize a stairwell light, although devices may disregard the `Transmission time` field if not being able to move at a variable rate (cf. Table 2.6).

Cluster	**Command**
Basic	
Identify	
On/Off	
Level Control	Move to Level (with On/Off) Cmd.
Scenes	
Groups	

Table 2.6: ZigBee `Dimmable Light`

IEEE 802.15.4 offers security services for data exchange at the link layer that use Counter with CBC-MAC (CCM)* as combined encryption and authentication block cipher mode. It can be used in environments requiring authentication only (against fabrication and modification), encryption only (against interception), or a combination of both.

ZigBee utilizes the IEEE 802.15.4:2003 transmission services of the data link layer. However, ZigBee does not use the security mechanisms provided by IEEE 802.15.4:2003 – they are completely replaced by a more advanced security concept. This concept supports the use of different key types and provides advanced key management services. Again, CCM* is used as cryptographic algorithm.

2.4 Summary

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 this section, 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 [137], 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. 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

Hypothesis 3 *Today's CAs are insecure. Irrespective of the used technology, no sound protection considering BAS specific constraints is deployed, thus enabling adversaries to attack installed devices.*

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 a burglar trying to hack a door switch to illegally access a building or to neutralize a window break sensor by crashing its CA. Such attacks can either be performed remotely via the network or locally, exploiting threats in a device's interface. Based on the abstract BAS model presented in Section 2.1, different attack scenarios on such software can be identified.

- SAC: 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). On the one hand, such attacks can be performed exploiting the threats discussed in Section 3.3. On the other hand, the two level concept of downloadable CAs (as it is available in e.g. KNX and LonWorks), which allows rapid innovation and implementation, may impose security risks and another way of compromising a BAS. Malicious, erroneous or compromised CAs may be uploaded long after device deployment and may tamper with the device software to attack a BAS. In case a system integrator installs such a malicious CA, neither network nor device protection help since the application seems to be trusted (trojan horse threat). A maliciously modified CA could obtain access to arbitrary memory regions where, for instance, secret keys are stored.

- ICD: 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.
- MD: 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.

This section is structured as follows: Section 3.1 is dedicated to a security analysis of current BAS installations based on BACnet and KNX technology. The goal is to investigate, if and how many BAS are being connected to the Internet and to find all available SACs. Section 3.2 then covers research on how to attack the CAs on these SACs. In this context, the EnOcean technology serves as an example for the investigations. Section 3.3 analyzes the threats to software running on BAS devices. Finally Section 3.4 defines the requirements for secure CAs.

3.1 Security in Current Installations

Similar to the Industry 4.0 initiative in the industrial automation or the establishment of cyber-physical systems, Ethernet and IP-based interconnection in open and well established BAS technologies such as BACnet, EnOcean, KNX, LonWorks, Modbus using specific ICDs (e.g. routers, gateways) are getting increasingly important. An integration and connection to the management level is achieved in a more convenient way. Remote access paves the way for energy management systems dedicated for functional buildings and ambient assisted living applications tailored to our homes of which future solutions may even reside within the cloud.

The Internet itself, however, is an open medium, which is used by adversaries all over the world to attack connected devices – including automation technologies. In the industrial automation, such attacks already have been performed (e.g. Stuxnet [95]). Security awareness among integrators, developers and end-users, however, is still missing as recent research and experiments have shown (e.g. Industrial Risk Assessment Map (IRAM)[1]). Thousands of Supervisory Control and Data Acquisition (SCADA) and industrial control systems are directly connected to the Internet exposing them to various attacks.

Even worse, often security vulnerabilities are present in those SACs. Early 2013, a software bug in a block heat and power plant has been discovered, which allowed unauthorized remote control. Meanwhile, the software has been fixed and a Virtual Private Network (VPN) box is available for secure data exchange [69]. Beginning of May 2013, a software bug in a widespread industrial control system has been discovered, which also allowed unauthorized remote control. 500 installations in Germany were affected [68]. It lasted till August 2013 until the manufacturer released an update for up to 200.000 world-wide installations [113].

Research and analyses targeted industrial automation mainly based on the fact, that a web server has been running on the default Transmission Control Protocol (TCP) port 80 of affected SACs, which has been exposed to the Internet search engine Shodan[2]. Up to now, no extensive research is available, which deeply analyzes BASs being connected to the Internet.

Therefore, this section is dedicated to security of existing (and installed) BASs. In Section 3.1.1, BACnet and KNX and the basic technology required to connect them to IP-based networks are briefly described. It is also outlined how discovery is standardized. Section 3.1.2 describes a scanning architecture to detect BASs being connected to the Internet and presents the results of worldwide Internet Protocol version 4 (IPv4) scans [131], [133].

[1] `http://www.scadacs.org/iram.html`, Last access: 2015/08/03
[2] `http://www.shodanhq.com/`, Last access: 2013/10/05

3.1.1 BACnet/IP and KNXnet/IP

BACnet provides the network option BACnet/IP, which permits BACnet devices to use standard Internet Protocols (User Datagram Protocol (UDP) and IP) as virtual data link layer.

BACnet defines the network visible part called BACnet object of a single data element. The internal data structure is not covered. Each BACnet object has a dedicated object type and represents a collection of properties. Each property has a data type defining the size and encoding of the data element. An object in a network is referenced by its system-wide unique `Object_Identifier` property, which is usually assigned during configuration. This provides a mechanism for accessing every object in the BAN via defined object access services.

The left part of Figure 3.1 shows an example communication. To search for BACnet devices in a network, the `Who-Is` broadcast service can be used by BACnet clients. Each receiving BACnet device shall respond with a broadcast `I-Am` request containing the `IAmDeviceIdentifier` (object type, device instance number) and some further properties. The most important services used to access and manipulate objects are `ReadProperty` (to read the value of a property), and `WriteProperty` (to set a new property value), which are sent via unicast communication using the object and property identifier.

BACnet integrates protocol security extensions for quite some time (since Addendum g in 2008), which should protect the exchanged data against interception, modification, and fabrication. Furthermore, advanced security concepts like the use of different key types and key revisions have been introduced.

KNXnet/IP describes transportation of KNX telegrams on top of IP networks with main purpose to expand building control beyond the local KNX bus. KNXnet/IP supports discovery and self-description of a KNXnet/IP server using one well known discovery endpoint. A server should at least support one control endpoint and one data endpoint (UDP or TCP on arbitrary ports) per KNX connection for additional communication (cf. Figure 3.2).

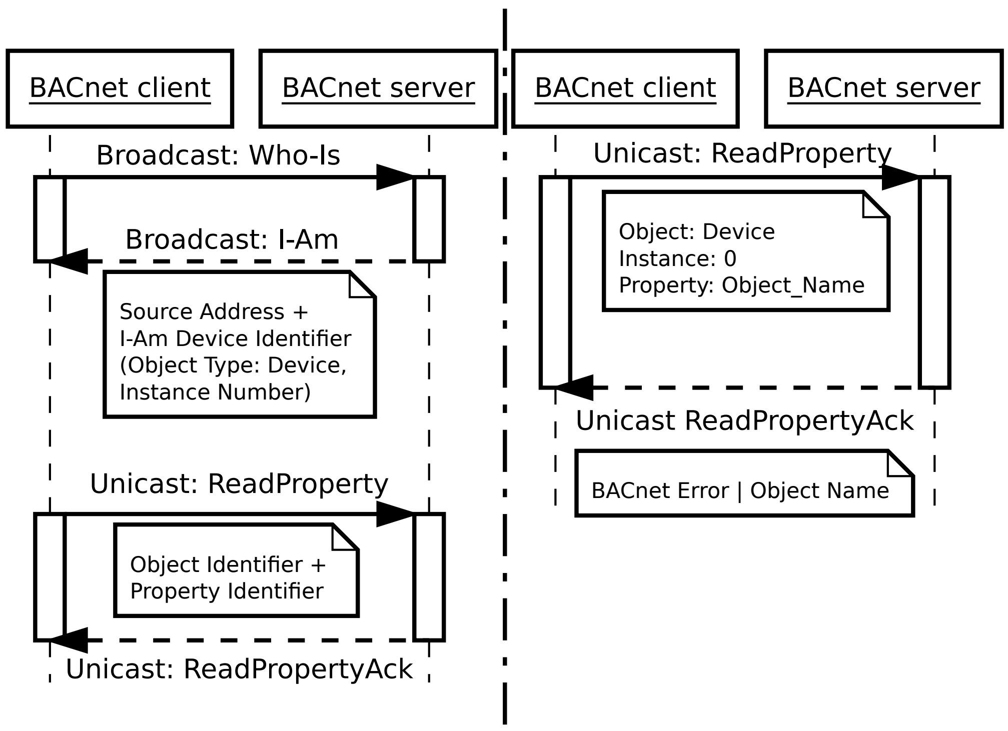


Figure 3.1: BACnet Communication

The left part of Figure 3.3 shows an example communication. For discovering a KNXnet/IP server, the client sends a `SEARCH_REQUEST` to the discovery endpoint (system default multicast address 224.0.23.12, UDP port 3671). Every server receiving the request should respond immediately with a `SEARCH_RESPONSE` frame for each of its service containers containing the Host Protocol Address Information (HPAI) (IPv4: IP address and port number) of the control endpoint. Afterwards, the client typically sends a `DESCRIPTION_REQUEST` to all received control endpoints using unicast telegrams and the information contained in the HPAI. Servers respond with a `DESCRIPTION_RESPONSE`, containing Description Information Blocks (DIBs) with supported protocol, capabilities, state information and an optional user friendly name. To connect to the control endpoints, a unicast `CONNECT_REQUEST` can be used.

The KNX Association standardized KNXnet/IP Secure in 2014. Until now, no devices implementing this standard are available.

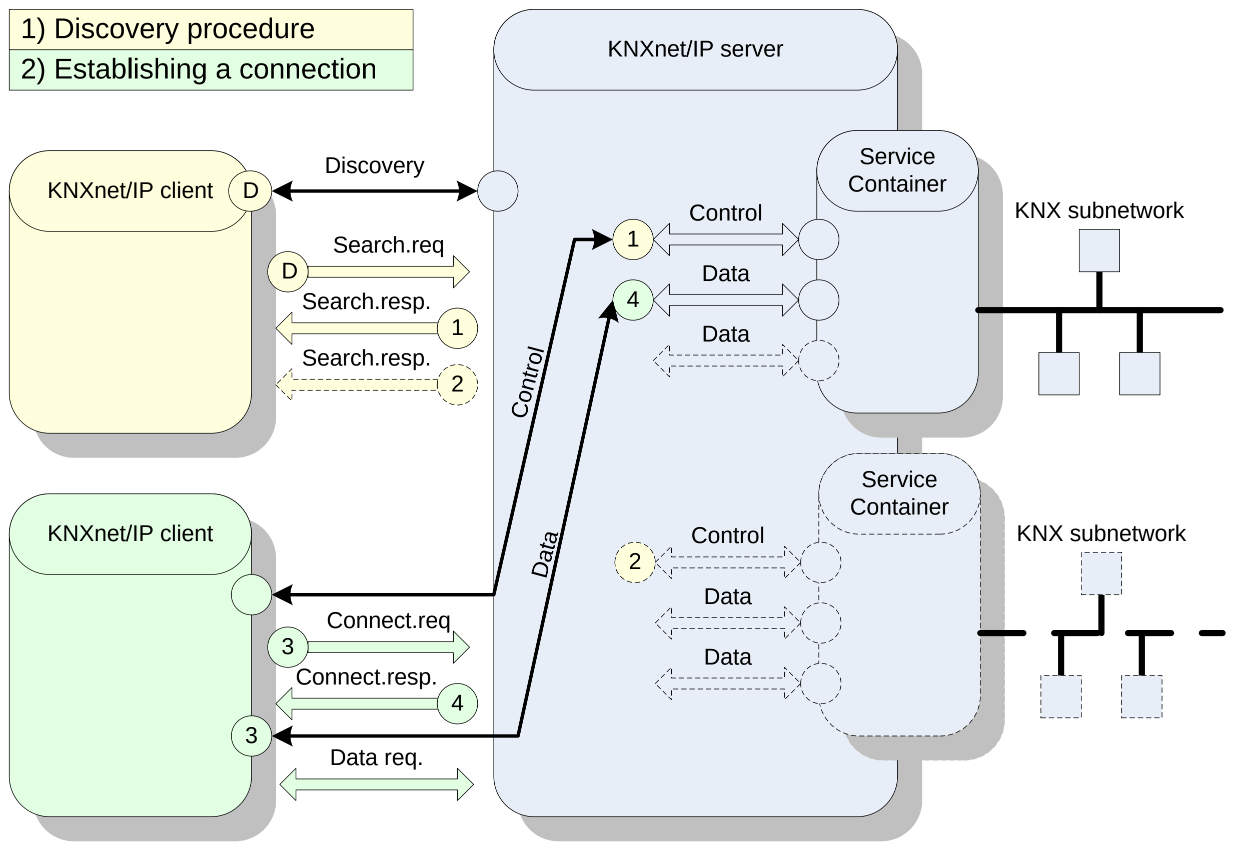


Figure 3.2: KNXnet/IP Discovery

3.1.2 A Survey on Worldwide Installations

3.1.2.1 Attack Vector

In order to research whether and how many BASs based on BACnet and KNX are openly connected to the Internet and what security measures are currently implemented, the following assumptions are made to find such sites:

- BACnet devices are connected to the Internet using their BACnet/IP network interface or using BACnet/IP ICDs (i.e. BACnet IP routers/gateways).
- KNX installations are connected to the Internet using KNXnet/IP ICDs. KNX devices directly connected to an IP backbone ("native" KNX IP devices) are not considered, since such devices hardly exist.
- IPv4 is used, no distinction between dynamic or static IP address ranges is made.

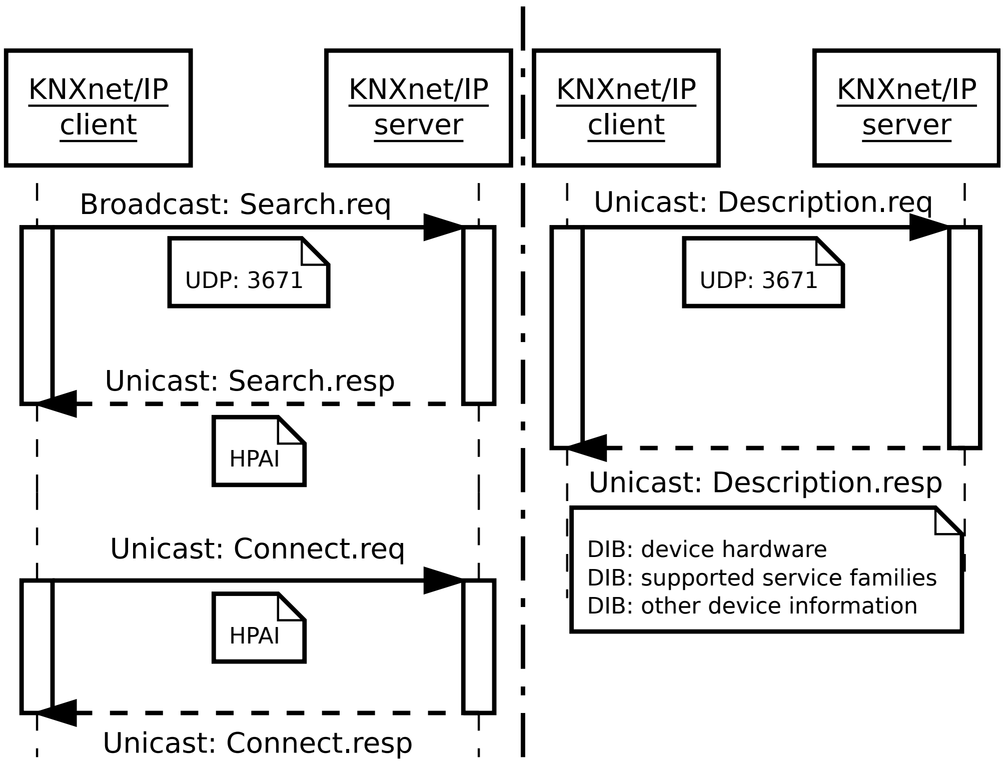


Figure 3.3: KNXnet/IP Communication

- The installations are standard compliant as described in Section 3.1.1 and are reachable via the default IP ports. Devices are either being directly connected to the Internet using a public IP address or reachable using port forwarding to a private IP address. These ports are not filtered using a firewall.

Based on these assumptions, the following attack vector can be used to analyze BASs being openly connected to the Internet.

Iterate through all (worldwide) IPv4 addresses and try to discover BACnet or KNX services. Simple port scans using common tools (e.g. nmap[3]) cannot reveal BAS specific details (e.g. human readable names, manufacturers) of connected installations and false positives might occur if non BAS protocols rely on this port. A false positive is an error, where a test indicates that a condition has been met although it has not been fulfilled. Also such port scans without deeper protocol knowledge might result in false negatives, since a connected device simply might ignore such scans. A false negative is an error, where a test indicates that a condition failed, while it actually was successful.

[3] `http://nmap.org/`, Last access: 2013/10/05

As shown in Section 3.1.1, the discovery mechanisms of BACnet and KNX rely on broadcasts/multicasts. Within the Internet, however, UDP/IPv4 multicast and broadcast telegrams and TCP/IPv4 broadcast telegrams are not routed, and TCP/IPv4 multicast telegrams are not supported. Thus, discovery does not work for non-local networks. Trying to perform a (slightly not standard compliant) discovery using unicast telegrams from the client to the server might work in this direction, but according to the specifications servers might reply using multicast/broadcast telegrams which will not arrive at the client.

To discover BACnet/IP based installations, a request as shown in the right part of Figure 3.1 can be used. This is handled by issuing a well formed and standard compliant unicast (UDP/IP, port 0xBAC0) `ReadProperty` on property `Object_Name` to a probably existing `Device Object, Instance 0` and evaluating the unicast `ReadPropertyAck`. If a BACnet server is connected, it either will reply with a BACnet error if for example the object is not found, or with the proper `Object_Name`.

The only well known default port in KNXnet/IP is the control endpoint UDP 3671, which can be used to get information about the data endpoint. The KNX standard defines, that devices may use the same port numbers for both endpoints but may also assign different port numbers for data exchange. Out of the box experiments with ICDs of the major KNX manufacturers revealed that control and data exchange is implemented using equal ports. Hence, to discover KNXnet/IP based installations, a request as shown in the right part of Figure 3.3 can be used. A well formed and standard compliant unicast `DESCRIPTION_REQUEST` is sent to the data endpoint, which is assumed to be located on UDP port 3671. If a KNXnet/IP ICD is connected, it replies with a unicast `DESCRIPTION_RESPONSE` containing the DIBs. If another device is connected, a non KNXnet/IP standard compliant answer or no answer will be received. If no device is listening to this port at all, no response will be received.

If the IP address of a BAS installation is found, further investigations can be done:

- Perform a detailed port scan on all ports. Often services, such as web servers, visualizations or web-cams are also reachable via the same IP address on probably non-standard ports. Additional information regarding the BAS installation (e.g. human readable installation name or abbreviation, manufacturer of the connected device) can easily be gained by simply accessing these services. If authentication is requested, supplying no password, default usernames and passwords gained out of the user manual of the specific manufacturer or trying guest accounts might give access.

- Information such as country, city, organization, Internet service provider, latitude and longitude can simply be gained performing geolocation and "whois" Domain Name Service (DNS) lookups of the IP address. Thus, it might be possible to clearly identify a BAS installation.
- If a BACnet/IP installation is found, `Read` and `Write Property` requests on different object types or object identifiers can be tried.
- If a KNXnet/IP installation is discovered, connecting to the installation via a KNXnet/IP tunneling request can be tried. It is then possible to read and write group addresses or receive all KNX data of the BAS.

3.1.2.2 Scanning Architecture

A simple but modular scanning architecture, which allows to deeply analyze BASs being connected to the Internet has been developed. A multi-threaded C-program initializes logging facilities, handles inter-process communication and synchronization using semaphores and allows to limit the amount of parallel IP connections. Pluggable protocol stacks (BACnet[4], KNX[5]) provide the communication services.

The test system has been connected to the Internet using a consumer service provider with bandwidth 150Mbit/s download and 15Mbit/s upload. System specifications are an Intel Atom CPU 330 (2 cores, 1,6GHz) and 3GB RAM. The IP addresses [1-9].*.*.*, [11-126].*.*.*, [128-223].*.*.* have been scanned. Concurrent connections have been limited to 2048/second. The timeout per connection has been set to 3 seconds. The average CPU load was around 19%, average memory usage about 400MB, average incoming traffic 50kBit/s and average outgoing traffic 532kBit/s. After scanning, IP geolocation informa-

[4] `http://sf.net/projects/bacnet/`, version 0.8.2
[5] `http://www.auto.tuwien.ac.at/~mkoegler/index.php/bcusdk`, version 0.0.5

Country	BACnet
US, United States	8989
CA, Canada	2296
FI, Finland	282
AU, Australia	271
ES, Spain	231
FR, France	148
SE, Sweden	138
GB, United Kingdom	131
DE, Germany	118
KR, Korea, Republic of	110
NO, Norway	103
IT, Italy	101
CZ, Czech Republic	98
TW, Taiwan	97
NL, Netherlands	89
NZ, New Zealand	47
HK, Hong Kong	45
JP, Japan	44
AT, Austria	42
CH, Switzerland	39
worldwide	13964

Country	KNX
DE, Germany	627
NL, Netherlands	522
ES, Spain	332
FR, France	244
AT, Austria	220
CH, Switzerland	204
IT, Italy	173
NO, Norway	129
SE, Sweden	120
BE, Belgium	119
IL, Israel	109
PL, Poland	67
GB, United Kingdom	56
GR, Greece	42
CZ, Czech Republic	30
RU, Russian Federation	24
VN, Vietnam	23
TR, Turkey	21
LT, Lithuania	20
PT, Portugal	20
worldwide	3295

Table 3.1: *Scan 1* Results (Top 20 Countries)

tion has been gathered using the Maxmind GeoLite Country and GeoLite City database[6]. The search for additional open ports has been performed using nmap and TCP SYN scans. The final visualization is based on Google Earth.

3.1.2.3 Scan Results

Three scans have been carried out in 2014.

Scan 1 started on 6th January 2014 and lasted till 9th May 2014. Table 3.1 shows the scan results grouped by technology and summed up per country. A total of 17.259 BAS installations has been detected. 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.

[6] `http://dev.maxmind.com/geoip/legacy/geolite/`, Last access: 2015/04/10

Figure 3.4 shows the geolocations of the installations in Europe, whereas Figure 3.5 shows the geolocations of the installations in the US. Finally Figure 3.6 shows the geolocations of KNX installations in East Asia.

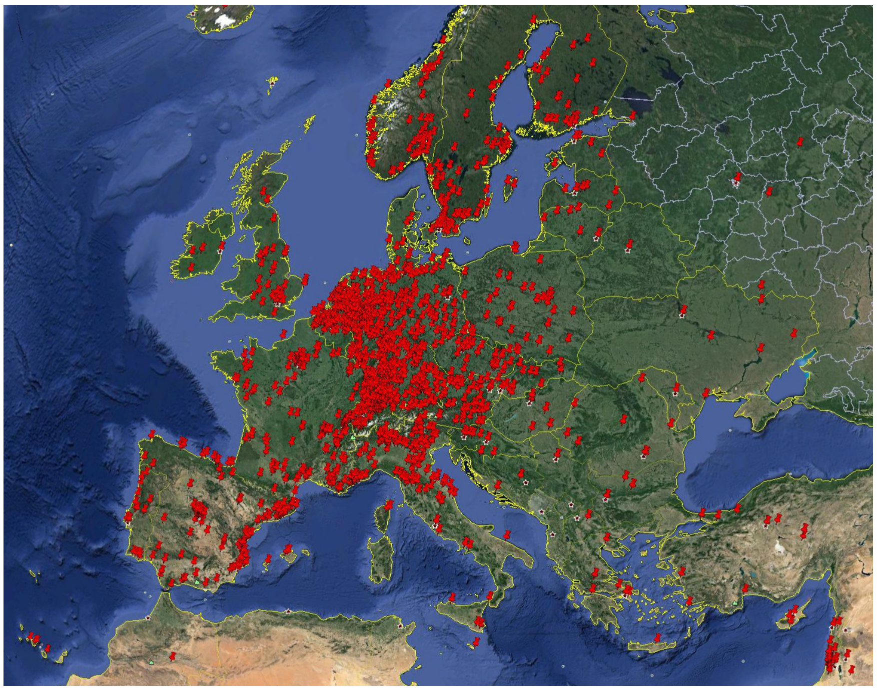


Figure 3.4: Unsecured Building Automation System Installations in Europe

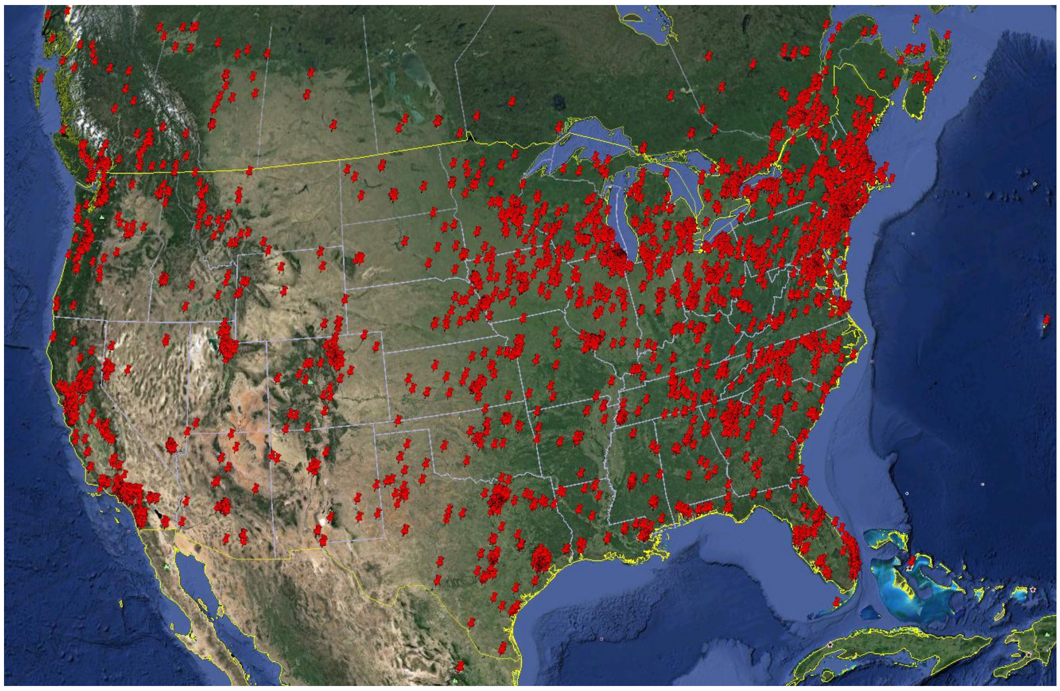


Figure 3.5: Unsecured Building Automation System Installations in US

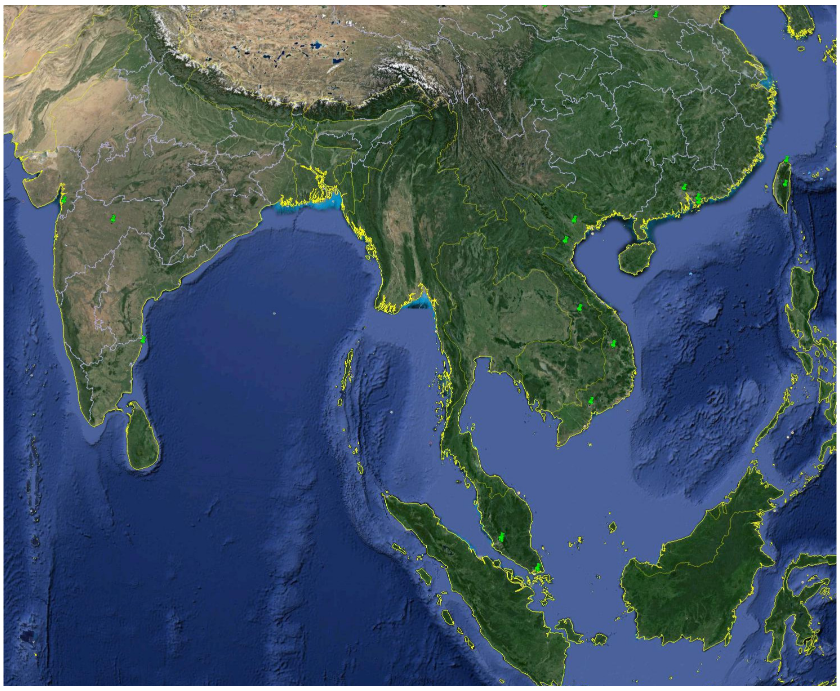


Figure 3.6: Unsecured KNX Installations in East Asia

A deeper analysis of BACnet installations is shown in Table 3.2. Most of the responses correspond to installations where no `Device Object` with `Instance` 0 is found. Since more detailed scans (with e.g. different instance numbers) on these installations can be considered as illegal, they have been left out of scope. A possible real adversary, however, would not stop at this point. In 250 cases, it was possible to read out the object name. Only in 3 cases (2x `services: service request denied`, 1x `object: service request denied`) the request has been denied which shows, that BACnet security as standardized since 2008 is seldom enabled in real systems.

Table 3.3 shows the additional top 15 open TCP ports grouped by port number. Typically, a web server is also available (especially in BACnet based installations) and authentication is required. Since either default or guest passwords often permitted a login, or a direct connection using the BACnet/IP or KNXnet/IP protocol is allowed anyway, severe security attacks cannot be prevented.

A total of 3.295 KNX installations have been detected. The MAC addresses have been extracted out of the DIBs and its `Organizationally Unique Identifiers` have been used to to find out the vendors of the devices. Devices per vendor have been summed up. Figure 3.7 shows this anonymized analysis. Under the assumption that the

Return value (E)rror, (R)eject, (A)bort	Count
E: object: unknown-object	13297
Empty	333
Success	250
E: device: unknown-object	31
R: Unrecognized Service	28
E: device: other	5
device	3
E: object: unsupported-object-type	3
E: property: unknown-object	3
E: services: service-request-denied	2
A: Buffer Overflow	2
E: device: configuration-in-progress	2
E: object: other	2
A: Other	1
A: Preempted by Higher Priority Task	1
E: object: service-request-denied	1

Table 3.2: BACnet Responses

Port	Count
80/http	7846
443/https	3472
135/msrpc	3302
139/netbios-ssn	3268
445/microsoft-ds	3261
8080/http-proxy	2504
21/ftp	2375
3389/ms-wbt-server	1983
23/telnet	1874
3011/trusted-web	1861
5960/unknown	1524
22/ssh	1451
25/smtp	1297
1723/pptp	1163
50001/unknown	1086

Table 3.3: Open Ports

security awareness of people installing KNX based systems is independent of the devices they deploy, the following estimation holds: The percentage of total installations to directly connected BAS can be estimated if the number of sold devices of one manufacturer is known. Investigations revealed, that at least 1-5% of all KNX installations are being insecurely connected to the Internet.

A rescan of the previously found installations (*Scan 2*) has been performed on 19th August 2014. Table 3.4 shows the *Scan 2* results summed up per country. 1.662 out of the 3.295 installations (about 50%) have still been reachable under the same IP address. This does not imply, however, that the other 50% of installations can now be considered to be secure. Most of the (private) installations use a dynamic IP address, obtained from the service provider.

A short third scan (*Scan 3*) with a small subset of IP addresses from countries in Europe lasted from 19th August to 25th August 2014. 375 installations (more than 10%) have been re-detected (identified by their MAC address), which were reachable with a different IP address.

Country	KNX
NL, Netherlands	427
AT, Austria	142
FR, France	133
ES, Spain	131
DE, Germany	115
CH, Switzerland	92
NO, Norway	87
IT, Italy	83
IL, Israel	73
SE, Sweden	61
BE, Belgium	58
PL, Poland	42
CZ, Czech Republic	27
RU, Russian Federation	19
GB, United Kingdom	19
LT, Lithuania	16
TR, Turkey	15
RO, Romania	14
SK, Slovakia	13
FI, Finland	12
worldwide	1662

Table 3.4: *Scan 2* Results (Top 20 Countries)

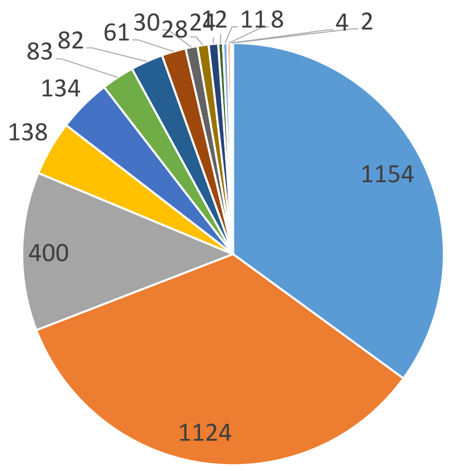


Figure 3.7: KNX Device Manufacturers

3.2 Attacking Control Applications based on EnOcean

While the previous section described an attack vector on how to find BAS, this section is dedicated to attacking the found CAs. The EnOcean technology is taken as an example to demonstrate attacks targeting the monitoring of process data exchanged between SACs. Due to requirements regarding low power communication, EnOcean's security concept is a trade-off between security and energy efficiency. It provides authentication, integrity, confidentiality and freshness. Several weaknesses within EnOcean's security have been revealed and are still present [2], [8], which are listed here for completeness since no extensive summary is available in related work:

- An insecure cryptographic algorithm (Alleged Ron's Code 4 (ARC4)) has been used until version 1.3 of [43].

- For authentication, EnOcean also relied on the unique production given sender ID until September 2013. [39] clarifies, that this concept is not secure.
- No encrypted key exchange mechanism is provided.
- Due to energy saving issues, the CMAC field is limited to 3 or 4 bytes. [36] recommends at least 8 bytes to avoid guessing attacks. Additionally, the rolling code window size leads to a reduced guessing time for a valid CMAC (approximately 65 seconds [8]).
- The usage of the rolling code is not mandatory. Thus, no protection against a replay attack is given.
- Authentication and encryption are not mandatory. In [43], use cases are given, that show the single use of the two security features. Without authentication, replay attacks can be launched; without data encryption, no confidentiality can be guaranteed.
- For many devices, private keys are not changeable. Therefore, the key lifetime cycle cannot be limited.
- Usually AES-CBC is used for input data longer than one block. Due to the very limited number of input data bytes in EnOcean (longest telegrams are 14 bytes [38]) and a fixed initial vector of `0`, this mode is reduced to Electronic Codebook Mode (ECB). Besides, 16 byte blocks are required for AES-128. The required padding sequence is fixed in EnOcean, which may lead to attacks against the encryption if known-plaintext weaknesses come up.
- The VAES algorithm (cf. Figure 3.10) has been invented by EnOcean. It does not seem to be mathematically proven or publicly reviewed. The algorithm seems to be weak with respect to known-plaintext and guessing attacks and its security seems to be dependent on the rolling code size.
- No profound, open discussion on the security of EnOcean is available. Besides, the document creation process, review policy and release policy need to be enhanced.

[43], for instance, did not receive updates for 7.5 months until version 1.5. Updates 1.6 - 1.9 occurred within less than one month. Besides, version numbers are not always clear and consistent within the document.

The following subsections are based on [2] and show, how EnOcean based CAs can be attacked via the wireless network using three practical scenarios based on side-channel and algorithm attacks.

3.2.1 Eavesdropping Control Application Communication

The first attack scenario is to eavesdrop the communication between two CAs.

- Similar to BACnet/IP or KNXnet/IP security, EnOcean's security introduced in 2013 is not enabled in practice. Most of today's devices communicate unencrypted until now, making eavesdropping trivial, e.g. by using standard hardware such as the USB 300 [44].
- Eavesdropping of teach-in telegrams (cf. Figure 3.8), which are required to bind an EnOcean receiver and an EnOcean transmitter, is possible. These teach-in telegrams are transmitted in plain text, thus an adversary can reveal the contained private information of the transmitting device. Information such as profile data (contains Radio Organizational Number (RORG)/telegram type, device type and function), private keys, security level (defining encryption and authentication level) and Rolling Code (RLC) (i.e. a synchronized shared value to prevent replay attacks) can be extracted. Using this information, future, even encrypted communication can be eavesdropped.

Two approaches seem feasible to prevent such attacks. First, an alternative teach-in procedure could be used to provide a secure way of the pairing process. Using a wired communication interface (e.g. a simple 2-wire connection probably even providing power supply) the necessary information could be exchanged securely between the devices. If, additionally a central MD is used, secret information such as private keys or RLCs can be generated and downloaded securely into the SACs, even at the installation side. In such way, no confidential information needs to pass the wireless medium in an unencrypted

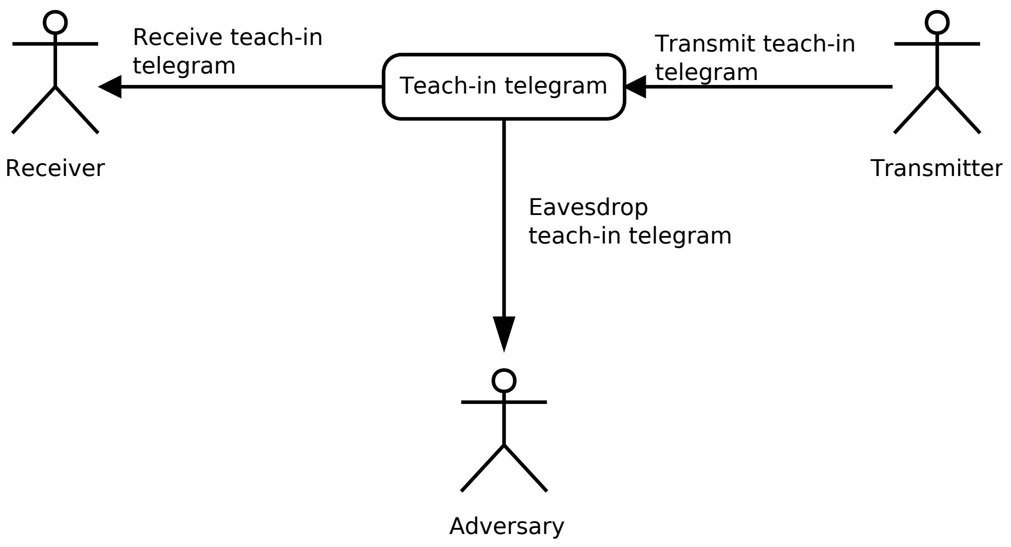

Receive teach-in telegram
Teach-in telegram
Transmit teach-in telegram
Receiver
Transmitter
Eavesdrop teach-in telegram
Adversary

Figure 3.8: Eavesdropping EnOcean Control Application Communication (based on [2])

way. Second, RF shielding or RF level limiting could be deployed, to reduce the risk of eavesdropping a teach-in telegram. This is, however, difficult for already installed SACs.

3.2.2 Interfering Control Application Communication

The second attack scenario are interference operations issued by an adversary. Thereby, the adversary tries to disturb normal communication by manipulating (parts of) a telegram, being sent by a transmitter. Thus, the telegram gets corrupted and the receiving CA will ignore it. On the one hand, adversaries might interfere all or only specially targeted communication to just generate annoyance.

On the other hand, more sophisticated attacks might target the pairing process of EnOcean: SACs use an RLC to provide protection against replay attacks. According to the specification, such an RLC needs to be synchronized between communicating CAs within a window value of 128 maximum difference to be able to en-/decrypt and authenticate a telegram. If the RLC differs more than 128, the devices have to be resynchronized using the pairing process. An adversary being able to interfere telegrams, will also be able to eavesdrop these plain text teach-in telegrams enabling the attack scenario as described in the previous subsection.

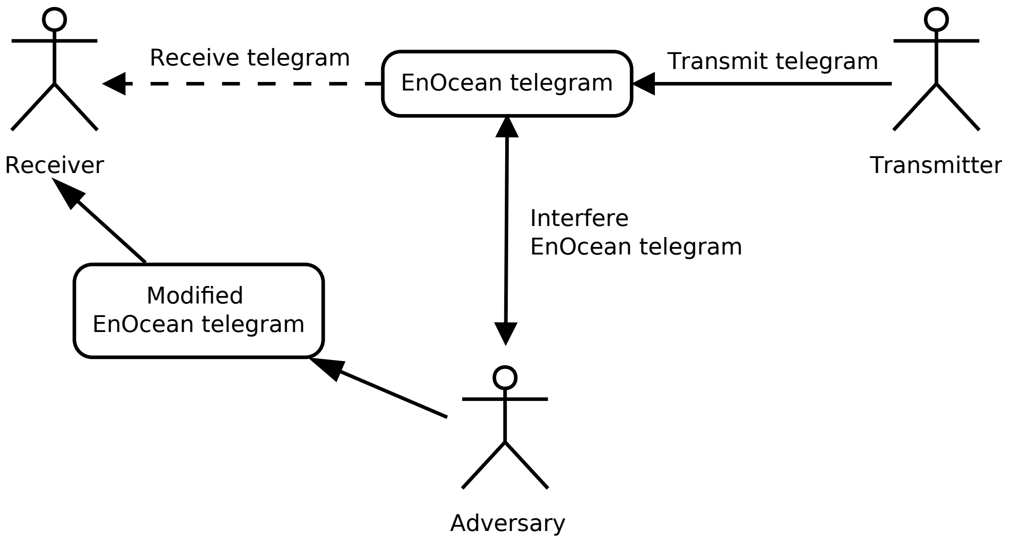


Figure 3.9: Man-in-the-Middle Attack on EnOcean Control Application Communication (based on [2])

The following procedure allows an adversary to perform a man-in-the-middle attack (cf. Figure 3.9):

1. Interfere EnOcean telegrams transmitted between devices (e.g. jam the Cyclic Redundancy Check (CRC) or hash value at the end of a telegram), until the RLC is out of synchronization in order to provoke a teach-in. In practice, it can be assumed that users will teach-in the corresponding devices, as soon as their malfunctioning is reproducibly noticed. Thus, it is not necessary to interfere all 128 telegrams to get the devices out of synchronization.
2. Eavesdrop and save the teach-in telegram and its parameters.
3. Interfere and save every encrypted and authenticated telegram from the original transmitter. Thus, the receiver will drop this telegram.
4. Send a newly generated telegram, encrypt and authenticate it with a higher rolling code (at least 128 steps difference to the original one).

As a result, the receiver will be synchronized to the adversary and will only accept the adversary's telegrams. Telegrams from the original transmitter will be dismissed.

This attack scenario does not work on standard EnOcean hardware, since it does not allow the required precise interfering or to freely change the Sender ID. A proof-of-concept attacking device allowing this man-in-the-middle attack is presented in [2].

3.2.3 Attacking Encrypted Control Application Communication

While the two previous subsections discussed side-channel attacks, this part describes a theoretical attack on encrypted EnOcean CA communication, in particular an attack on its VAES encryption.

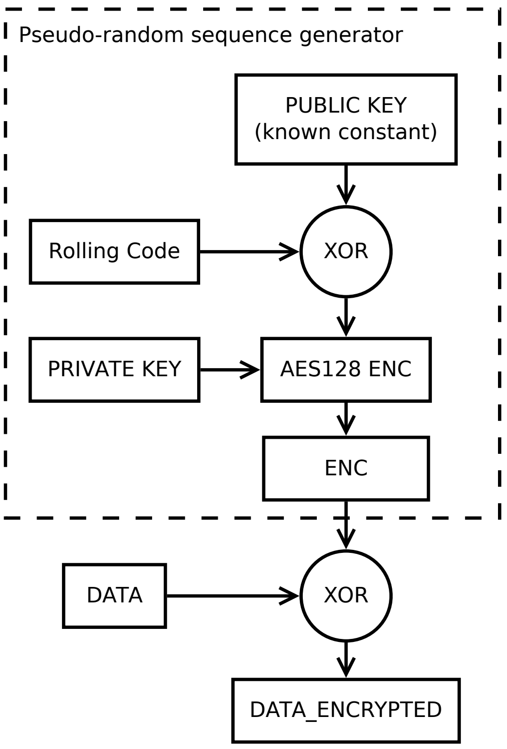


Figure 3.10: EnOcean Variable Advanced Encryption Standard Encryption (based on [43])

Using a state of the art block cipher in combination with a commonly used mode of operation (e.g. AES-CBC), a block size of a multiple of 16 bytes is required, which may produce a large overhead especially for energy harvesting networks, where typically information less than 5 bytes needs to be transmitted. Hence, the VAES encryption (cf. Figure 3.10) has been specified, which allows a smaller encrypted data length. The basic idea is that transmitter and receiver share a synchronized random value, that changes after each telegram. This value is cut to the same length as the plaintext data and XORed with it to generate the encrypted data. For decryption, the receiver XORs the encrypted telegram with its (=the same) cut random value to recover the plaintext. A pseudo-random sequence generator has been specified to provide this synchronized random value. A known constant (EnOcean imprecisely refers to it as public key) is XORed with the RLC and serves as an input to AES-CBC encryption using the private key. The encrypted value has a variance depending on the RLC size (16 or 24 bits). Obviously the RLC needs to be the same value for the transmitter and the receiver, which can be guaranteed by including the RLC within the transmitted telegram or by checking the Message Authentication Code (MAC).

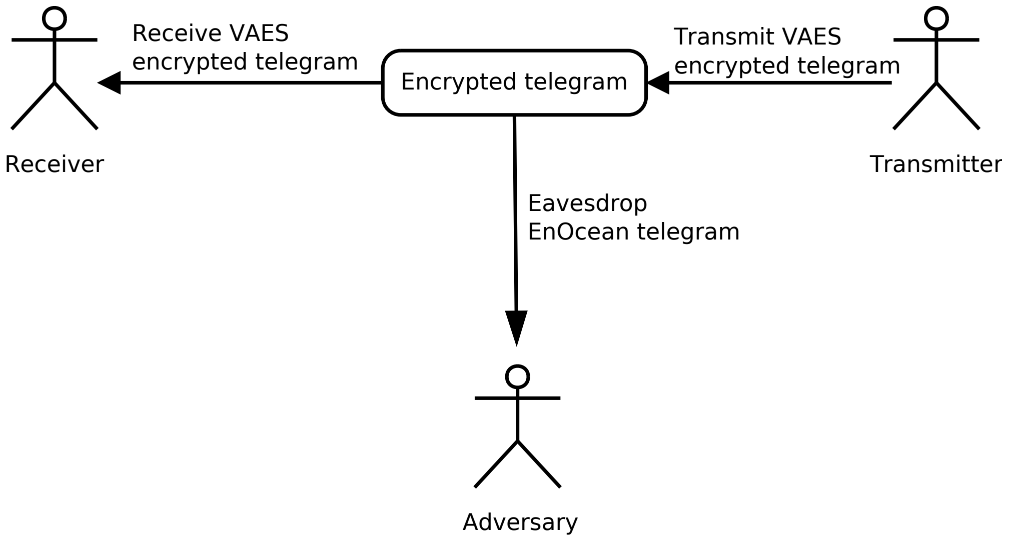

Receive VAES encrypted telegram
Encrypted telegram
Transmit VAES encrypted telegram
Receiver
Transmitter
Eavesdrop EnOcean telegram
Adversary

Figure 3.11: Attacking Encrypted EnOcean Control Application Communication (based on [2])

An attack (cf. Figure 3.11) on the VAES encryption targets the pseudo-random sequence generator. If an adversary knows this value, it can simply eavesdrop the transmitted telegrams. In fact, the pseudo-random sequence generator is periodic with a wrap around after 2^{16} or 2^{24} telegrams. The following attack model based on known plaintext values can be used to get access to the pseudo-random sequences:

1. Find a CA communication relying on the VAES encryption. An adversary has to eavesdrop RORG telegrams of type $0x30$ or $0x31$, which specify that encryption using AES-CBC or VAES is being used. Since AES-CBC always has a 16 byte block alignment, all telegrams with other alignments need to be based on VAES.

2. Eavesdrop encrypted data fields and save them using the RLC or MAC as index. This can be achieved using standard EnOcean devices and is invisible to the transmitter and receiver since no active interference takes place.

3. Associate encrypted data bytes with known plaintext data bytes. The probability to guess the correct plaintext data bytes depends on the CA of the transmitter. The easiest case is a simple switch, being either pressed or released. Obviously the device type needs to be known by the adversary.

4. Recover and save the pseudo-random sequence by XORing the encrypted and plaintext data bytes. At least 2^{16} or 2^{24} telegrams have to be eavesdropped, to get access to all possible pseudo-random values. The actual RLC width/period has to be known. It can be guessed by detecting fixed data bits within the telegrams.

5. Use the pseudo-random value to decrypt all previously saved or future telegrams. Note that it is not possible to recover the private key of a device, but actually this is not necessary to get access to the plaintext data.

To increase the probability of a successful attack, the previously described procedure can be combined with further measures. In combination with e.g. interference, additionally transmitted telegrams can be provoked.

In the following, an attack on three EnOcean CAs – a PTM energy harvesting switch, a temperature and a light sensor – is described in more detail. A summary is shown in Table 3.5.

The PTM switch transmits a telegram containing 11 different possible plaintext values when being pressed and a fixed release telegram when being released. Thus, at least every second telegram is definitively known to an adversary giving a plaintext probability greater than 0.5. Since only one byte is being transmitted and the typical data pattern can be recognized, a PTM switch can easily be identified. Assuming that the switch is pressed ten times a day, the 16 bit RLC overflows in about 8.9 years and a 24 bit RLC in about 2356 years. A successful attack thus is lasting longer than 17.8 years. This time might be reduced, if a switch is pressed more often or the data variation of the press telegram can be reduced (e.g. a switch being known to only have 1 button).

Device EEP	PTM switch D2-03-00 [38, p. 106]	Temperature sensor A5-02-04 [38, p. 25]	Light sensor A5-06-01 [38, p. 32]
Data type	4 bit (11 button configurations)	8 bit (-10° C to 30° C)	8 bit (300lx to 30000lx or 600lx to 60000lx)
Data pattern	1 byte (only 4 bits used) Fixed release telegram	1 variable data byte 1 bit teach-in	3 variable data bytes 1 bit teach-in
Possible plaintext values	11 + 1 release	at room temperature: 15° C to 25° C $\approx$ 64	256
Plaintext probability	> 0.5	≈ 0.016	≈ 0.0039
Transmission pattern	Press / release	Usually periodic every 1s-100s	Usually periodic every 1s-100s
RLC overflow 16 bit 24 bit	10 switch actions/day $\approx$ 8.9 years $\approx$ 2356 years	1s transmission cycle $\approx$ 18.2h $\approx$ 194 days	1s transmission cycle $\approx$ 18.2h $\approx$ 194 days
Attack duration 16 bit RLC 24 bit RLC	 $\approx$ 17.8 years $\approx$ 4712 years	 $\approx$ 47 days $\approx$ 34 years	 $\approx$ 194 days $\approx$ 140 years

Table 3.5: Variable Advanced Encryption Standard Encryption Attack

The temperature sensor transmits a telegram containing an 8 bit encoded temperature value between -10° C to 30° C in a configurable interval. To enhance the plaintext guessing probability, the data variance can be reduced to 64 different values if the temperature interval is reduced to 15° C to 25° C for a room temperature sensor. Likewise it can be reduced, if the sensor is used as outside temperature sensor and the adversary can guess the correct temperature using other channels (e.g. weather information or measuring the temperature on its own). Assuming a transmission interval of 1 second, the 16 bit RLC overflows in 18.2 hours and the 24 bit RLC in 194 days. A successful attack is lasting 47 days respective 24 years.

A light sensor encodes the illumination between 300lx to 30000lx or 600lx to 60000lx into an 8 bit value, thus giving 256 possible plaintext values. Assuming a transmission interval of 1 second, an attack is lasting 194 days for a 16 bit RLC and 140 years for a 24 bit RLC.

Although the duration of a successful attack might seem quite long in many cases, it has to be noted that no deep mathematical analysis of the VAES algorithm has been performed. It can be assumed, that the duration can be shortened by a well-trained adversary.

3.3 Threat Analysis

To be able to provide secure CAs in BASs, it is first necessary to identify the threats to CA software and analyze possible vulnerabilities. Based on the Open Web Application Security Project[7] and [76], the following categorization can be established:

- *Authentication vulnerability*: Authentication is the process of verifying the identity and ownership of a user or a BAS node. An adversary may exploit vulnerabilities such as authentication bypass via assumed-immutable data (i.e. an authentication decision is handled on the client side and is thus subject to adversary modification) and empty string or hard-coded passwords being deployed to nodes by security unaware users.
- *Authorization vulnerability*: Authorization is the process of verifying the access rights of an authenticated process or user. An adversary may bypass authorization mechanisms by exploiting vulnerabilities like weak privilege management (i.e. CAs being executed with higher privileges than necessary).
- *Code quality vulnerability*: On the one hand, poor code quality may lead to poor usability. On the other hand, it enables an adversary to stress a system in unexpected ways. Examples are double frees (occurs in memory management when `free()` is called more than once on the same memory address), leftover debug code, memory leaks (occurs in memory management when unneeded memory is not freed), null dereferences (dereferencing a null pointer), uninitialized variables (using the value of an uninitialized variable), using freed memory (referencing memory after it has been freed).
- *Cryptographic vulnerability*: Adversaries may exploit vulnerabilities in cryptographic modules due to

[7] `http://www.owasp.org/`, Last access: 2010/08/03

  - Algorithmic problems: use of insecure algorithms such as Data Encryption Standard (DES), MD5; choosing the wrong algorithm (e.g. encryption algorithm for hashing); inappropriate use of an algorithm (e.g. insecure encryption methods or non random initial vector); implementation errors.
  - Key management problems: weak keys (too short or simple), key disclosure (keys transmitted in clear text), key updates (reuse of existing keys).
  - Random number generator problems: poor random number generators (c: `rand()`, Java: `java.util.Random()`), no seed to the random number generator, use of the same seed for the random number generator every time.

- *Error handling vulnerability*: Adversaries may try to generate errors and then exploit vulnerabilities in error handling such as empty catch blocks, improper cleanup on thrown exceptions, missing error handling, uncaught exceptions.

- *General logic error vulnerability*: This category includes vulnerabilities such as assigning instead of comparing or comparing instead of assigning, omitted break statements or use of `sizeof()` on a pointer type.

- *Input validation vulnerability*: If (user) input to CAs is not checked for proper constraints, an adversary might stress a BAS by intentional malformed inputs. This type of vulnerability includes flaws due to buffer overflows (e.g. [121]), format strings [120], improper data validations, string termination errors or validations performed at client side.

- *Protocol errors*: This category includes vulnerabilities such as the failure to add integrity check values, the failure to check for certificate revocations, the failure to encrypt data, key exchange without entity authentication or trusting self-reported DNS names.

- *Range and type error vulnerability*: Improper handling of the type, size and signedness of data may be exploited by an adversary. Typical vulnerabilities are comparing classes by name, integer overflows, sign extension errors, signed $\rightleftarrows$ unsigned conversion errors or truncation errors.

- *Sensitive data protection vulnerability*: This category contains vulnerabilities that lead to insecure protection – i.e. failure to maintain confidentiality and integrity – of sensitive data due to e.g. heap inspection, information leakage, privacy violation. Simple attacks by a malicious CA would allow to read the whole memory – including firmware, secret password or any other confidential information – and to send the contents over the network interface.

- *Session management vulnerability*: A session refers to a lasting interaction between CAs and/or users. Vulnerabilities in session management due to e.g. cross site scripting, insufficient session-ID length enable adversaries to hijack such interactions.

- *Synchronization and timing vulnerability*: This category covers vulnerabilities, which occur due to improper or unhandled timing dependencies in CAs or in communication between CAs. Examples are capture-replay vulnerabilities (i.e. the replay of network traffic), race conditions [9] or time-of-check/time-of-use race conditions as shown in the Listing 3.1 of a simple program, which accesses a file:

```
1 if(!access(filename,W_OK)) { // check permissions
2   f = fopen(filename,"w+"); // open file
3   operate(f); // operate on file
4   ...
5 }
6 else {
7   fprintf(stderr,"Unable to open file %s.\n",filename);
8 }
```

Listing 3.1: Synchronization and Timing Vulnerability

If an adversary is able to manipulate the file with the name `filename` after its access rights have been checked in line 1 (e.g. by suspending a program and creating a link to another file), all subsequent operations on the new file are not checked for access permissions.

- *Mobile code*: Mobile code is code being transmitted across a network and being executed on a remote machine. Special guidelines are necessary to protect such code from manipulation by an adversary. Common examples of mobile code are Java applets or ActiveX elements.

- *Use of dangerous Application Programming Interfaces (APIs)*: Certain APIs in various programming languages are considered as being insecure and their usage may impose security vulnerabilities in CAs. Examples are dangerous functions (e.g. `C`-functions `gets(), strcpy(), strcat(), printf()`) that do not check for the size of destination buffers), insecure temporary files or the use of obsolete methods.

To be able to create secure CAs, it is essential to know how often attacks on the presented vulnerabilities occur and which vulnerabilities need to be dealt with. No statistical data is, however, available for BASs or more generally for Embedded Systems (ESs) typically being deployed in BASs. Therefore, an analysis based on the introduced categorization has been performed [129]. Figure 3.12 shows a breakdown of vulnerabilities being openly available at the US-CERT Vulnerability Notes Database[8]). All entries of the years 2007 to March 2010 (632 in total) have been analyzed, categorized and counted. The numbers give an adequate overview of the commonness of vulnerability types. Although this analysis basically covers vulnerabilities of traditional IT systems (i.e. OS and middleware software), it shows the broad range of threats also possibly applicable to software in BASs. Moreover, it reveals that since years, input validation vulnerabilities are the most common challenge to provide secure CAs, but the other vulnerabilities must not be neglected.

3.4 Requirements for Secure Control Applications

Due to the extreme broadness of threats and vulnerabilities to CAs, an attack model needs to be defined:

Definition 3 *Software attack model: 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.*

[8] `http://www.kb.cert.org/vuls`, Last access: 2015/08/03

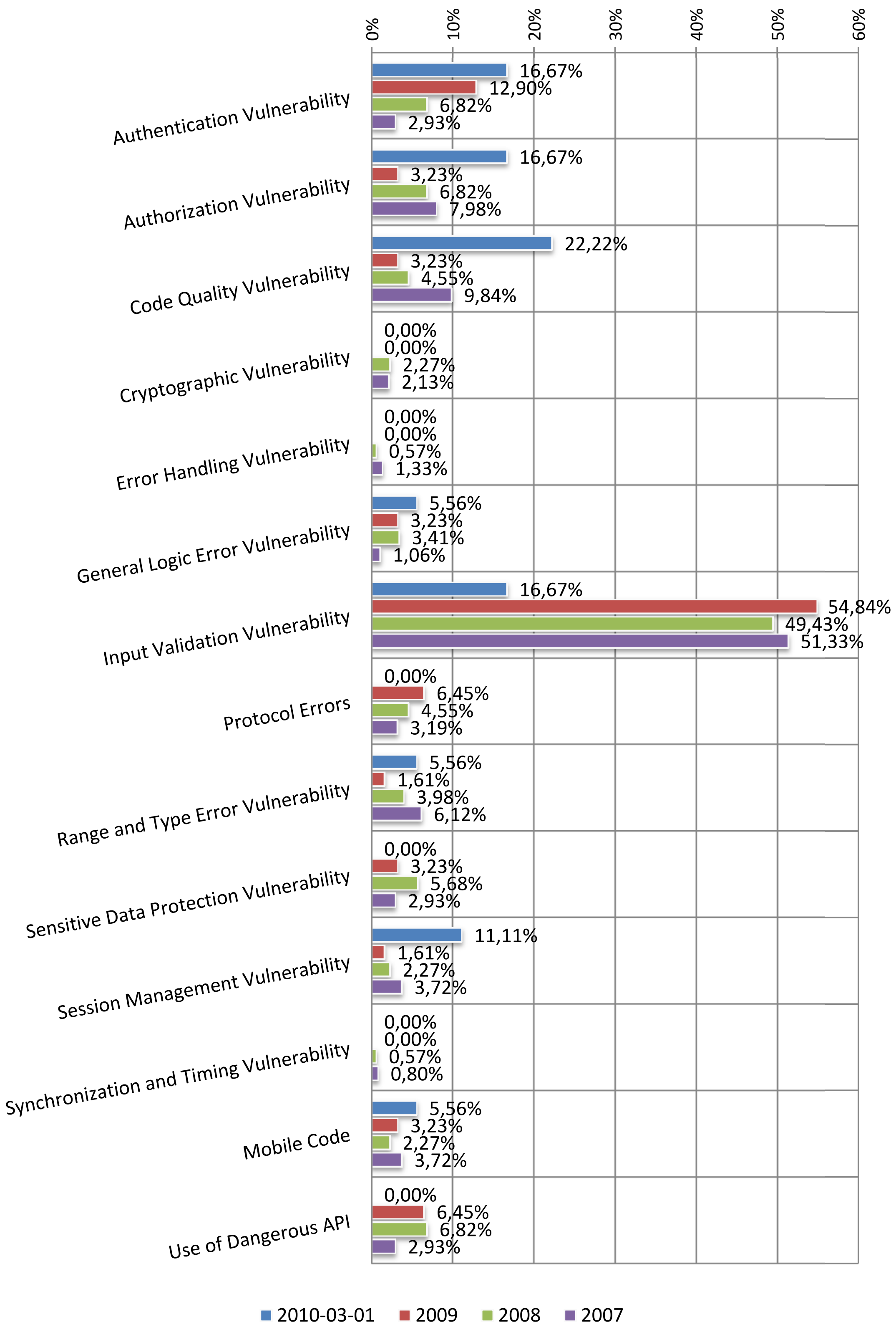


Figure 3.12: Vulnerabilities Breakdown 2010-03 – 2007

Thus, it is not enough to provide yet another new method to prevent single attack types like buffer overflows, but to provide a solid and reasonable approach allowing secure CA development. Therefore, security mechanisms need to be included that ascertain the operational correctness of protected code and data before and at runtime, enforce that application content can remain secret and protect against probing.

Definition 4 *Secure Control Application: A secure CA is a CA, which additionally provides mechanisms to prevent and detect software attacks.*

The overhead imposed by security mechanisms needs to be reasonable small and a suitable balance between required level of security and available resources for a specific domain has to be found. The resulting implications to BAS nodes are described in the next two sections.

3.4.1 Functional Requirements

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 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, i.e. DoS attacks, need to be prevented or detected.

3.4.2 Organizational Requirements

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 bus- or battery-power. Security mechanisms (especially cryptographic algorithms) are computationally intensive and must not exceed the available device's processing resources (processing gap) and power resources (battery gap) [136]. 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. This way, also portability of CAs can be achieved due to their separation from the system software. Clearly, this may impose security risks, which a security concept has to deal with: While malicious, erroneous or compromised CAs may be uploaded long after device deployment, they shall not interfere with the concerning device software and thus violate the device's security.

***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. It can also be used to add required functionalities not anticipated during development. Since such an update mechanism also offers an additional attack point it has to be protected against unauthorized use.

***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. For many services in the IT domain the amount of devices that communicate with each other is relatively small, thus allowing the client/server model to be used in most cases where only the communication between the clients and the server has to be secured. BANs, on other hand, usually consist of only a few MDs, some ICDs with defined applications, and thousands of manifold SACs. Communication between SACs occurs peer-to-peer based without a central instance. Thus, scalability of the integrated security mechanisms is of major concern.

***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 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). In the IT/office domain, the data volume to be transferred is commonly high (in the order of mega- or gigabytes) with usually no real time requirements.

***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. Downward compatibility will influence the acceptance of a security extension significantly. Such a compatible extension shall not make existing standard system components obsolete. It shall be possible to use them simultaneously with new secure devices, without mutual interference. However, security must not be compromised.

***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 [64]) 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. It has to be prevented, that credulous users deactivate or bypass security measures unintentionally and enable an adversary to attack a BAS.

3.5 Summary

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 BACnet and KNX based installations are being directly connected to the Internet, allowing an adversary to attack them. Besides, exemplary attacks on the EnOcean technology demonstrate, that BAS specific constraints have not been covered in research. Manifold vulnerabilities and requirements need to be considered to be able to provide secure CAs.

4 Software Protection Techniques

Hypothesis 4 *Existing software protection techniques from the IT domain cannot be used to develop secure CAs. They are insufficient with respect to the functional and organizational requirements and not applicable to BAS.*

A broad range of commonly used state of the art approaches aims at improving application level security in the presence of programming flaws or untrusted code. Considering the attack model as depicted in Figure 4.1, they can be distinguished into techniques preventing attacks, detecting attacks, recovering after attacks and providing tamper evidence.

This section focuses on related work which addresses attack prevention and attack detection. Attack recovery and tamper evidence are out of scope of this dissertation. Software protection techniques present in the IT world are likewise covered as techniques

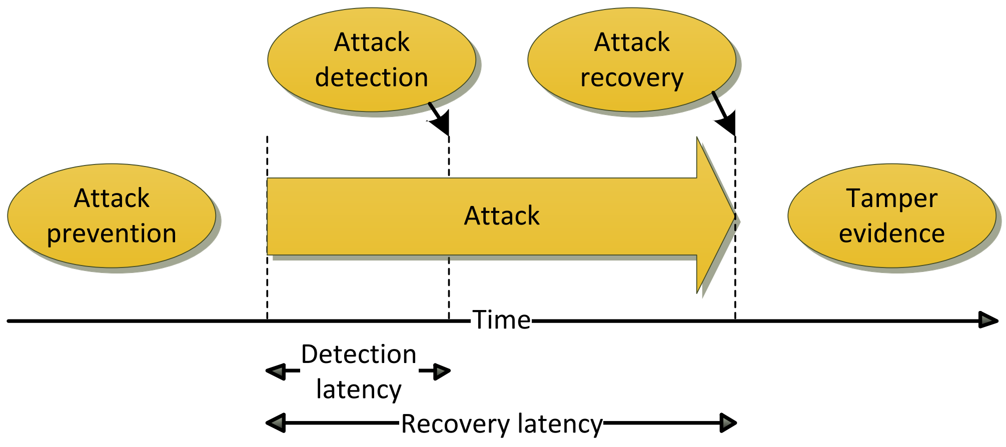


Figure 4.1: Attack Model

specifically tailored to ESs. In theory, it is preferable to fully prevent attacks. This may, however, not always be possible or feasible with respect to a system's resources. Thus, to prevent a successful exploit of an attack, its detection latency has to be smaller than the time it takes to reach an undesired software state. In the following, a short description of software protection techniques (cf. Figure 4.2) is given and a categorization into static software methods (cf. Section 4.1.1), dynamic software methods (cf. Section 4.1.2), hardware assisted methods (cf. Section 4.2), human assisted methods (cf. Section 4.3) as well as hybrid methods (cf. Section 4.4) is performed. Then, security in open BASs is analyzed. Finally, the presented techniques and technologies are evaluated with respect to their applicability to protection against vulnerabilities, security requirements and device classes in BASs (cf. Section 4.5).

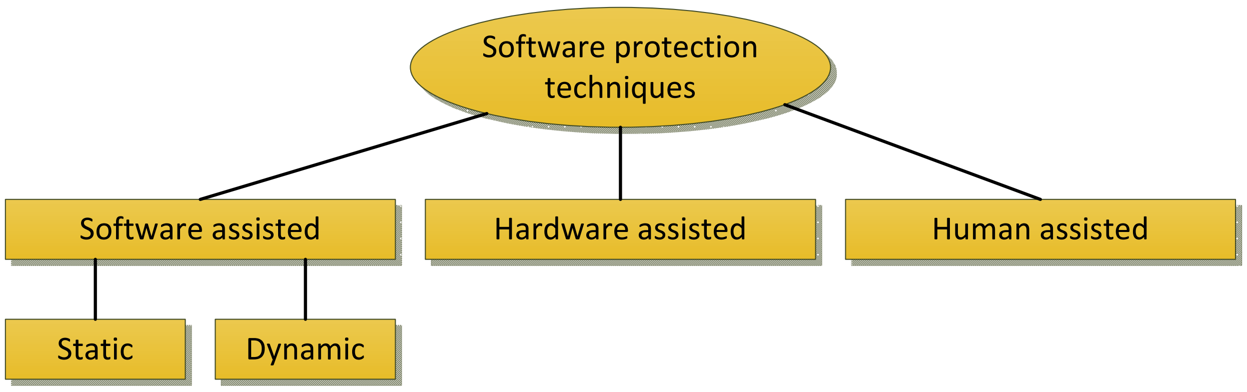


Figure 4.2: Software Protection Techniques

4.1 Software Assisted Methods

Approaches falling into this category try to improve security by applying additional software assisted methods. A security policy or at least some security rules need to be defined at system level, which then can be checked for violation. Software assisted methods can be divided into static, usually performed at compile time or before execution, and dynamic methods. The latter generally try to hamper software attacks by applying security mechanisms at runtime, either by preventing attacks a priori, or by detecting malicious or undesirable actions and reacting to them according to a given policy.

4.1.1 Static Methods

Static Code Analysis (SCA) generally refers to manual as well as automated tool supported analysis of program code to detect certain properties of a program without executing it [22]. In security, it is used to detect programming flaws that result in vulnerabilities and provide valuable feedback in a human readable form to developers. Automated tools usually use pattern matching to detect common flaws. More sophisticated tools combine techniques like annotations, heuristics and modeling the execution state.

On the one hand, SCA can be carried out on program source code. [15], for instance, describes a compile-time analyzer which is capable of detecting dynamic errors. An analyzer traces the execution paths, models memory and reports inconsistencies. Besides, automatically generated models abstract the behavior of individual functions and thus allow inter-procedural errors to be detected. [110] presents `Splint`, an approach, which tries to mitigate buffer overflow vulnerabilities by lightweight and efficient static analysis of C-code. Their approach is based on the static analysis tool `LCLint` [46]. With minimal effort a detection of software flaws such as unused declarations, type inconsistencies, use before definition, unreachable code, ignored return values, execution paths with no return, likely infinite loops, and fall through cases can be achieved. By adding annotations to the source code, which provide additional information about its intended behavior, stronger checking can be performed. The commercially available static analyzer `ASTRÉE`[1] intends to prove the absence of runtime errors such as float rounding errors, overflow errors in C-programs. By abstract interpretation of program code, all possible errors are reported (i.e. `ASTRÉE` is always sound), however occasionally errors are signaled, that cannot really happen (i.e. false alarms on spurious executions). `ASTRÉE` does not support dynamic memory usage or recursion has, however, been deployed in the A340 and A380 aircraft and Jules Vernes Automated Transfer Vehicle.

On the other hand, SCA can be carried out on binary code. [27], [108] describe an approach to statically analyze x86 binaries using symbolic execution. [161] uses SCA based on abstract interpretation on embedded executable assembly code to determine

[1] `http://www.astree.ens.fr/`, Last access: 2015/08/03

whether coding standards have been followed during application development or not. However, only the presence of unwanted hard coded pointer variables is discussed.

Using *Code-Signing (CS)* program code is signed by its producer to confirm its origin and non-modification [126]. Although CS does not provide any security measures itself, the user can at least decide about the trustworthiness of a program with respect to its origin. Most of the current OSs (e.g. Linux, Apple Mac OS X, Microsoft Windows) utilize CS to secure their update services and prevent maliciously distributed code via their patch system. Typically a trusted third party is required to establish the trust relationship between code producer and consumer, but also approaches exist, where this is not required [143].

Somewhat similar to CS, but with different purposes, is *Watermarking (WM)* [24]. It is used to embed additional, secret and non-removable information into a piece of data, usually to assure that the rights of the creator are not hurt and digital information is disseminated in a controlled way [125]. A typical example is media WM where information about owner and copyright is embedded into a picture, music or a movie (Digital Rights Management (DRM)). WM applied to software protects against illegitimate modifications and tampering by its users [17] before actually executing it.

Proof-Carrying Code (PCC) [119] is a technique where a code producer provides a proof along with a program allowing to check with certainty, that the code is secure to execute (e.g. does not to contain buffer overflows). On the one hand, the user specifies a set of security rules. The code producer, on the other hand, creates a formal security proof which proves adherence to these rules. Correctness is checked using a simple and fast proof validation. PCC may be used in environments such as web browsers or ESs, where the execution of downloaded code is allowed. A trust relationship between code producer and user is not required, since all information needed to determine that the code is secure to execute is contained within the code and the proof. If modifications have been applied to the code, PCC guarantees that either the proof is no longer valid and the user can reject the execution of the program, or the proof is valid and does not correspond to the program anymore or that the proof is still valid.

4.1.2 Dynamic Methods

IDSs observe the behavior of a system by e.g. tracing system calls, file system operations or network traffic and use the collected information to detect malicious modes or actions [111], [117]. Advanced approaches exist, which rely on a distributed system to allow better scalability [107].

Signature based Intrusion Detection Systems (SIDSs) detect malicious actions by comparing observed information to a collection of signatures describing known attacks. Most modern anti virus software use SID to detect malware.

Anomaly based Intrusion Detection Systems (AIDSs) use representations of the trained, normal behavior of a system to detect abnormal activities which are assumed to be caused by attacks. Here, mechanisms from simple statistical analyses to neural networks and other artificial intelligence techniques are used. As an example, [71] is able to detect intrusions at the level of privileged processes since they usually implicate anomalous sequences of system calls. In fact, short sequences of system calls executed by running processes are a good discriminator between normal and abnormal behavior. An enhanced training and learning sequence approach based on finite state automata is introduced by [148]. [47] describes an AIDS, which allows anomaly detection by dynamically extracting information from the call stack of a program execution. The return addresses are being used to generate an abstract execution path between two program execution points. Besides, approaches exist which describe an AIDS that targets mobile ad-hoc networks and considers requirements such as the limited processing power and battery gap [32].

Software Monitoring Techniquess (SMTs) observe the execution of specific programs. By identifying and reacting to certain security relevant events they can check if programs behave according to a given (human specified or automatically generated) security policy. On the detection of malicious behavior, actions can be taken to prevent it or to stop the program. Monitoring usually takes place at instruction level by checking some or all instructions which are to be executed by a program. [54], [167] describe a secure environment for untrusted helper applications, which protect the hosting application by intercepting and filtering dangerous system calls and thus restricting a program's access to the OS. Pre-existing legacy code can likewise be protected as newly developed software. Control-Flow integrity is a safety property being able to hinder adversaries from arbitrarily controlling a program behavior [1]. Its enforcement is simple, practical and can be performed efficiently. Quite similar [93] defines local security checks to implement a global security property by monitoring the control flow of software.

A *Sandbox (SB)* is a technique where possibly untrusted and malicious programs are executed in a controlled way, often with restricted permissions [159]. The essential benefit is that the executing host is protected from direct attacks of the software running in the SB. Additionally, the behavior of the program can be monitored and controlled. The first approaches used SBes to isolate software faults by logically separating an application's address space [168]. Adobe Acrobat Reader 10.0 and later, for instance, use a SB to implement a so called protected mode prohibiting write accesses of code, which has been infiltrated by malicious `pdf` documents[2].

An extension to SBes is the concept of Virtual Machines (VMs) [153]. Process VMs provide virtualization mechanisms between OSs and processes. Such techniques are present in most modern OSs and allow multiple processes to coexist on the same hardware and have the illusion to possess the whole system for itself. System VMs apply virtualization techniques between the OS and hardware. In such way, several OSs may be executed on same hardware without interfering with each other. A lot of different VMs have been developed, which target (mobile) ESs and consider their requirements and resources. [157] is a simple, fast and robust VM, which allows to run code compiled for a common instruction set architecture, independently from the underlying hardware. To allow efficient on-the-fly compilation, the basic instruction set is matched to popular processor architectures. Additional instructions for inter-device communication, power management and error recovery are provided.

Self Checking Code (SCC) describes techniques where programs check themselves for modifications at runtime, assuming that modifications are undesirable and probably malicious [6]. Simple methods generate hash values over parts of the program and compare them to stored values, usually determined at compile time. Note, however, that such approaches can easily be distinguished from normal code and thus could be bypassed by an

[2] `http://www.heise.de/ct/meldung/Adobe-Reader-X-mit-Sandbox-fuer-Windows-verfuegbar-1139095.html`, Last access: 2015/08/03

adversary. Measures such as watermarking, obfuscation and anti-debugging techniques have to be taken, that hamper the discovery and modification of SCC code. Other methods apply public key cryptography to generate signatures of code and critical data which are added to the program. This way not only the program's integrity, but also its source and trustworthiness can be checked. [75] describes a SCC mechanism to improve tamper resistance of large programs with the aim of securely executing them on potentially hostile hosts. The concept of testers is introduced, which redundantly test for changes in the executable code as it is running and report modifications. Addressing the challenge of obfuscating the checking mechanisms, oblivious hashing [21] continuously calculates hash values based on the dynamic execution trace of a piece of code. This way operational correctness can be checked at any point in the program and SCC can be blended seamlessly into the application code.

There is also a number of *Attack Specific Countermeasures (ASCs)* which target only specific types of vulnerabilities. In such cases, the attack requirements are narrowed to a small set of conditions, and assumptions are made on how they could be prevented. Especially for the still dominant stack based buffer overflow attacks a lot of approaches exist which try to hinder them [172]. A popular example is StackGuard [28], which has been released in 1997 for the GNU Compiler Collection (GCC). It applied several techniques such as storing return addresses for comparison or placing known values (canaries) between buffer and control data for detecting overwrites. ProPolice [45], being part of the GCC by default, additionally protects all registers saved in a function's prologue (e.g. frame pointer) and also sorts array variables to highest part of the stack frame. A compiler addition (i.e. `/GS` switch) for Microsoft Visual Studio [12] inserts a cookie (random data) between stack data and function return address and checks it when a function exits. Besides, buffers are placed in higher memory locations than non buffers (e.g. function pointers). Quite similar heap based buffer overflow attacks can be performed (e.g. [4], [115]). However, detection mechanisms are deployed which make such rather simple attacks harder. A random cookie can e.g. be added to each heap block. If it has been tampered due to an overflow the application can then be aborted. Besides, heap integrity checks are common when blocks are being freed and data structures are checked for validity. PointGuard [30] describes a compiler technique to defend against buffer overflow attacks by encrypting pointers while they are stored in memory and only decrypting them when being loaded into registers. Address Space Layout Randomization (ASLR) is another technique being deployed against buffer overflow attacks [151]. By randomly arranging the position of key data (e.g. base of the executable, stack and heap in a process's address space, position of libraries), the predictability of the exact memory locations required for attacks is not given anymore. In this way, ASLR does not prevent such attacks, but it makes them harder to perform. Since the Vista version of Microsoft Windows, for example, stack randomization is active and the stack base address varies by 0–31 pages and a random offset in 4 KB pages is chosen. Similar to heap randomization, the heap offset is also randomized between 0–4 MB. To protect against exploits attempting to call shared libraries (e.g. `return-to-libc` exploit [154]), function entry points are moved in memory between 256 different locations. To counter format string attacks FormatGuard [29] provides a replacement of `printf()` function with more secure implementation and additionally performs function argument counting. Race conditions are addressed in e.g. RaceGuard [31], a kernel enhancement which circumvents temporary file race vulnerabilities.

Embedded *OSs* provide hardware abstraction and offer an interface to execute (multiple) CAs. A kernel handles tasks such as interrupts, memory management, Input/Output (I/O) or network access and limits what an application is allowed to do. Until now, only two OSs are available, that target security critical applications and have achieved a satisfying Common Criteria for Information Technology Security Evaluation (CC) level [90]. Green Hills Software Integrity operating system[3] is one of the most secure commercial OS. Its integrity kernel has been certified according to CC Evaluation Assurance Level (EAL) 6+ level [80]. The Secure Embedded L4 microkernel [100] is a complete, general-purpose OS kernel. Its functional correctness has formally been proven in such a way that the implementation strictly follows a high-level abstract specification of the kernel behavior.

[3] `http://www.ghs.com/`, Last access: 2015/08/03

4.2 Hardware Assisted Methods

Dynamic software mechanisms typically face two common problems: First, they are executed on the same underlying processing hardware which exposes them to security attacks themselves. Second, they implicate a performance overhead. Hardware assisted methods may pose an extra barrier that is not easily bypassed with conventional techniques and speed up security processing.

A common approach is to use a *Co-Processor (CP)* which performs security checks at runtime. This often involves static analyses of the programs to generate some kind of execution profile that is used for monitoring. The CP is usually closely attached to the executing processor to enable extensive low-level monitoring. [5] describes such a concept, which is able to monitor the inter-procedural control flow, intra-procedural control flow, and the instruction stream integrity. Upon a security violation, the CP disrupts the executing application. [114] provides an extension which allows detection within one instruction cycle. [177] presents an approach, where a secure CP is used to execute an IDS.

The increasing use of multicore architectures even on ESs allows their usage for security mechanisms. *Physical Partitioning (PP)* [70] may be used to improve a system's reliability as well as its resistance against security attacks by separation of execution domains.

Processors implementing the *Harvard Architecture (HA)* provide resistance against code injection attacks by design. The separation of instruction and data memory makes it impossible to execute injected code. [139] proposes a change to the memory architecture of modern processors imitating the HA on von Neumann processors. The memory is split virtually into code and data memory. Therefore, a processor would never be able to fetch injected code execution.

CPU EXtensions (CPUEXs) providing security features are common in the IT domain. The No eXecute (NX) bit was first introduced by AMD[4]. It is present in most modern processor architectures, only with different names, e.g. Intel's eXecute Disable (XD) bit. Traditional buffer overflows enter a system as data and then are executed. Thus, these

[4] `http://www.amd.com/`, Last access: 2015/08/03

CPUEXs prevent the execution of code from data segments. They allow memory regions to be designated as being non executable meaning that they may only be used to store data making such traditional code injection attacks impossible. CPUEXs are supported since Microsoft Windows XP SP2 (i.e. Data Execution Protection (DEP)), Linux 2.6.8 and Mac OS X 10.6. Dynamic information flow tracking [158] is an architectural support which tracks I/O inputs and monitors their use. Any unintended use is detected by a processor and handled by special software. ARM TrustZone® technology[5] is a system-wide approach to security, which is tightly integrated into processors and targets high performance platforms with applications such as secure payment, DRM and web based services. [79] is an approach – consisting of a hardware architecture and a software counterpart – to provide a processor virtualization architecture to be used on mobile terminals for pre-installed applications as well as downloaded applications. [176] addresses software protection based on a Field-Programmable Gate Array (FPGA) hardware. A secure hardware component is used to recognize and certify strings of keys hidden in regular unencrypted instructions in order to guarantee the non-modification of the executable.

4.3 Human Assisted Methods

Manual *Inspection And Certification (IAC)* performed by humans can be applied during the software design and development process to strengthen security. In such way, conformance to standard coding rules (e.g. [73]) can be checked and code reviews can reveal software bugs. IAC of a product's functionality is already common for ensuring interoperability between devices from different manufacturers.

In the context of software protection techniques, *Formal Verification (FV)* is a mathematical proof that a program (i.e. an implementation) fulfills its specification. For automated verification it often models the problem using formal methods such as finite state automata or petri nets. Model checking explores all states and transitions within a model and, if no violation of the specification is found, the model is proven to be correct. [20] describes model checking programs for security properties, a formal approach for finding

[5] `http://www.arm.com/products/processors/technologies/trustzone.php`, Last access: 2015/08/03

bugs. Rules for safe programming practices are identified, encoded as safety properties and then verified, if they are obeyed. [97], [98] present a flexible method to detect malicious code patterns in executables by model checking.

4.4 Hybrid Methods

Hybrid methods combine different approaches with the goal to enhance overall security. Only a few approaches are listed here as examples:

Program shepherding [99] is a SMT approach which relies on three techniques to enforce a security policy. First, execution privileges of instructions can be restricted on base of code origins. For that purpose, the three categories (1) code from original image on disk, (2) dynamically generated but unmodified code, and (3) modified code are distinguished. Second, control transfer can be restricted by e.g. checking each direct or indirect call, return or jump and forbidding the execution of shared library code except through declared entry points. Third, all transfers of control are monitored by un-circumventable sandboxing.

A hybrid approach combining static PCC and dynamic SMT is presented in [149]. Model-carrying code executes untrusted applications by allowing the user to select a security policy and enforcing it during runtime.

The Java Virtual Machine (JVM) uses a hybrid approach. Within the Java runtime system, a SB forms an essential part of the security architecture. [33] identifies four basic security mechanisms of the SB in the Java Micro Edition (Java ME)[6]: (1) Java class files are verified for correctness before execution, (2) only a restricted API is available for untrusted code, (3) class loading mechanisms cannot be bypassed, and (4) the set of functions accessible to the SB is closed. The SB model has undergone many improvements to provide better security [55], [56]. A security manager class, for instance, has been deployed, which checks and restricts actions performed by untrusted code. Besides, an advanced class loader ensures, that such code cannot interfere with the operation of other Java programs. [169]–[171] discuss, how to further strengthen the Java security con-

[6] `http://www.oracle.com/technetwork/java/embedded/javame/index.html`, Last access: 2015/08/03

cept for mobile code. [65], [66] describes a Java secure execution framework providing a highly configurable security environment.

There are also a number of JVMs for ESs available, which offer a subset of the full Java functionality. Oracle provides the official Java ME for mobile and other ESs such as mobile phones, personal digital assistants, TV set-top boxes and printers. Although being strictly reduced in size and overhead, it still offers powerful programming interfaces, robust security and built-in network protocols. The minimum system requirements specify around 160-512 KB of total memory and a 16 bit processor being clocked at at least 16 MHz. Oracle's Java Card[7] technology offers a secure environment for applications that run on smart cards and other devices with very limited memory (1 KB RAM, 32-48 KB ROM) and processing capabilities. [152] presents an advanced JVM architecture based on the Connected Limited Device Configuration (CLDC) standard, which is targeted to run on next generation smart cards. [11] describes a JVM for small ESs with a special focus on home automation. The JVMs LEJOS[8] and NanoVM[9] allow the execution of standard Java byte code, at least after preparation with an automated tool. They target very low profile ESs. LEJOS has been developed for for the Lego Mindstorms RCX programmable bricks with memory requirements of about 10 KB and support for multi-threading, exceptions and synchronization but e.g. missing floating point support. The NanoVM was written for the Atmel AT-Mega8 micro controller, e.g. lacks multi-threading support but has a small memory footprint of approximately 7 KB size. Besides, the JVM JamaicaVM[10] with a memory footprint of about 256 KB provides a real-time environment for ESs. Also dedicated processors implementing the JVM exist (e.g. [145]).

[7] `http://www.oracle.com/technetwork/java/embedded/javacard/overview/index.html`, Last access: 2015/08/03

[8] `http://tinyvm.sourceforge.net/`, Last access: 2015/08/03

[9] `http://www.harbaum.org/till/nanovm/index.shtml`, Last access: 2015/08/03

[10] `http://www.aicas.com/`, Last access: 2015/08/03

4.5 Applicability Analysis for Building Automation Systems

This section evaluates the potential of available software protection techniques to provide an overall software security for CAs. Thus, their applicability to SACs, ICDs as well as MDs and their protection against the presented threats (cf. Section 3.3) and the attack model will be discussed in the next paragraphs. A summary can be found in Table 4.1. Table 4.2 evaluates the applicability with respect to the functional and organizational security requirements described in Section 3.4.

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. These requirements are, however, contradictory and cannot be applied all at the same time.

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. Applications successfully checked for security vulnerabilities could be "malicious by intent" and e.g. distribute sensitive data such as encryption keys. Besides, SCA, CS and 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.

Applying SCA to binaries is difficult because of decoding the instructions. Especially in the field of BAS and the ESs being deployed there, different instruction set architectures hamper the use of general binary code SCA tools. Besides, most SCA techniques target `x86` binary code only, which are rarely used in this domain. SCA on source code can be applied more easily, as long as the code is portable across architectures and standards for the programming language are respected. A problem of SCA is that for typical programming languages some fundamental questions (consider, e.g. the halting problem) are undecidable or uncomputable [109]: The CA could be frozen, or be locked in an endless loop. Even when the allowable instruction set (language) for CAs is restricted, it cannot be guaranteed that insecure behavior can be detected a priori – at least while the remaining subset is still powerful enough to do anything useful. Therefore, the results of SCA tools can never be sound and complete and it will always be uncertain if some of the detected flaws are false positives, or some real flaws remain undetected. [182] shows that SCA does not detect some errors in real word programs and claims, that support of automatic analysis tools is needed to be able to cope with high rate of false positives. [178] shows that SCA is affordable with respect to costs and [16] demonstrates that it can be quite effective.

CS can be quite effective to prevent the installation of arbitrary CAs by simply refusing to execute unsigned or not properly signed code. Note, that a trust relationship to the code signer and possibly a trusted certification authority are needed. However, CS does not prevent malicious programs or accidental flaws. Besides, mechanisms for sensitive data protection are needed, since the data required for maintaining the trust relationship must not be overwritten by an adversary.

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. Besides, proofs do have a significant size, which can be a magnitude larger than the actual code size. Although automated proof generators could be developed, they will not be able to solve every difficulty for all types of programs and complex security policies. [141] even states, that "there are properties related to information flow and confidentiality, that can never be proven this way". A hybrid combination of the static PCC approach with dynamic execution monitoring, however, seems to be practically feasible [149].

Obviously 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. Besides, applying appropriate countermeasures during runtime is a difficult task, since quite often there is no clear way on how to prevent damage and hinder a successful attack.

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.

AIDSs in contrast also allow to detect novel intrusions, which are not yet present in the database of an IDS. They are, however, not able to distinguish between natural changes of the monitored system and attacks. Thus, false positives could be reported, if e.g. nodes are added or removed from a BAS.

AIDS, SMT as well as SCC may be efficiently implemented and could therefore be quite appropriate. SMT at least requires hardware support for context switches.

ASCs can also work well in many cases, but might not be applicable due to differing processor and memory architectures on SACs and ICDs. [172] even states, that preventing buffer overflows using ASCs only works in 50% of all test cases. Besides, the inner working of such methods is often publicly available and therefore adversaries might find a way to bypass them more easily. Memory randomization (e.g. stack or heap randomization) is often applied to hamper successful exploits of buffer overflows, but does not really protect a BAS [151].

The applicability of 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 JVM with its vast execution and security mechanisms imposes a big overhead. In any case, careful design is necessary to mind pitfalls and provide security [53].

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). Traditional OSs for such lean ESs thus cannot provide comparable protection or are not even designed to provide security measures. In general, trusted software such as an OS, cannot be guaranteed to contain no flaws. For completeness, it has to be noted that possibilities exist to overcome the lack of memory management units: [63] describes an approach providing virtual memory and OS protection.

Using dedicated hardware for security checks allows 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: [150] presents a technique, that allows a return-to-libc attack to be performed on x86 executables and calls no functions at all. [52] describes a code injection attack for the HA based Atmel AVR microcontrollers. [140] introduces "return-oriented programming", which can be used to introduce arbitrary behavior in a program code whose control flow has been diverted without injecting any code and is not prevented by CPUEXs methods. Finally, [19] demonstrates an attack to voting machines. Despite their consequent HA, arbitrary programs can be constructed out of existing code chunks from the stack and tricky placement of return calls and thus elections can be manipulated.

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.

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 [80], [100]. A simpler form of FV, which provides a trade off between a full mathematical proof and reduced input set nevertheless seems feasible [10].

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 (cf. [141]). 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. Therefore, hybrid methods cannot be discussed in the following tables.

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

	method or BAS	*Authentication vulnerability*	*Authorization vulnerability*	*Code quality vulnerability*	*Cryptographic vulnerability*	*Error handling vulnerability*	*General logic error vulnerability*	*Input validation vulnerability*	*Protocol errors*	*Range and type error vulnerability*	*Sensitive data protection vulnerability*	*Session management vulnerability*	*Synchronization and timing vulnerability*	*Mobile code*	*Use of dangerous APIs*
static software methods	SCA	−	−	*p*	*p*	*p*	*p*	∼	∼	∼	−	−	−	−	*p*
	CS	−	−	∼	−	−	−	∼	−	−	−	−	−	∼	−
	WM	−	−	−	−	−	−	−	−	−	−	−	−	−	−
	PCC	−	−	*p*	−	−	*p*	∼	∼	∼	−	−	−	∼	−
dynamic software methods	SIDS	∼	*d*	*d*	*d*	*d*	*d*	*d*	∼	*d*	−	−	−	∼	−
	AIDS	∼	*d*	*d*	*d*	*d*	*d*	*d*	∼	*d*	−	*d*	*d*	*d*	−
	SMT	∼	*d*	*d*	−	−	*d*	*d*	−	*d*	−	−	*d*	−	−
	SB	∼	*d*	*d*	*d*	*d*	*d*	*d*	∼	*d*	*p*	*p*	*d*	*d*	−
	SCC	−	−	−	−	−	*d*	*d*	−	*d*	−	−	−	−	−
	ASC	−	−	−	−	−	−	*p*	−	−	−	−	−	−	−
	OS	∼	*d*	∼	*d*	−	*d*	∼	∼	*d*	∼	∼	∼	∼	∼
hardware supported	CP	−	−	*d*	*d*	*d*	*d*	∼	−	∼	−	−	−	−	*d*
	PP	−	*d*	*d*	*d*	*d*	−	−	−	*d*	*d*	−	−	∼	−
	HA	−	−	−	−	−	−	*p*	−	−	−	−	−	−	−
	CPUEX	−	−	−	−	−	−	*d*	−	−	−	−	−	−	−
human	IAC	*p*	*p*	*p*	*p*	*p*	*p*	*p*	*p*	*p*	*p*	*p*	*p*	−	*p*
	FV	−	−	*p*	−	*p*	*p*	*p*	−	*p*	*p*	−	*p*	−	*p*
open BAS	BACnet	−	−	−	−	−	−	−	−	−	−	−	−	−	−
	EnOcean	−	−	−	∼	−	−	−	∼	−	−	−	−	−	∼
	KNX	∼	∼	−	−	−	−	−	−	−	−	−	−	−	−
	LonWorks	−	−	−	−	−	−	−	−	−	−	−	−	−	−
	ZigBee	−	−	−	−	−	−	−	−	−	−	−	−	−	−

Table 4.1: Comparison of Software Protection Techniques and Software Security in Open Building Automation Systems: Protection Against Vulnerabilities

Open BAS standards cover – if at all – only security for communication aspects. However, several flaws exist which shall be listed here briefly.

BACnet added security with Addendum g in 2008. It has been integrated into the standard in 2012. Before, several threats have been presented [72], [146], [175]. To support all kinds of applications, the use of different communication models shall be possible. BACnet only provides support for the client/server model – exchanging process data within groups as it is possible in LonWorks and KNX is not supported. Attacking EnOcean based CAs is described in Section 3.2 in more detail. Before 2013 (and in fact, in all of today's installations), attacks to KNX based CAs could be launched in several ways [59], [61]. The current KNXnet/IP specification [103] describes some attacks and also countermeasures. [94] analyzes the security extension KNXnet/IP Secure [103] and shows some limitations concerning the provided level of security. [146] shows, that the security mechanisms of LonWorks are not sufficient to fulfill the requirements on BASs integrating security subsystems. IEEE 802.15.4/ZigBee provides a solid base. [181] however, highlights some weaknesses in the standard. Mechanisms to protect against interruption attacks (e.g. DoS attacks) are not supported by any of these standards. A key problem which has not been solved by any of these five systems is the generation and distribution of the required initial secrets. Even if the architecture of the system itself is secure, a mechanism must be available to distribute the initial secrets in a secure manner.

Mechanisms to counteract device attacks and provide software security are not covered by any of these standards. In fact, only the specifications of the standards can be considered in the following. Local implementations or security checks cannot be considered, since they are not open to the public.

Since BACnet, EnOcean and IEEE 802.15.4/ZigBee only specify the communication protocol and the application model to be used, details about the device implementation and methods to manage CAs (e.g. management tools to configure and upload CAs) are not covered. Therefore, appropriate security mechanisms that handle device attacks are missing, too. A variety of engineering tools is available for LonWorks, most of them based on the LonWorks Network Operating System (LNS) management middleware. However, no security countermeasures are incorporated into LNS. For KNX, a single management tool called ETS is available which provides mechanisms to configure and upload the CAs.

−: 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*
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 4.2: Comparison of Software Protection Techniques: Security Requirements

However, the only way to avoid unauthorized use of these management services is to use an insecure access control mechanism. The new security extensions will address the access to CAs, but they are not final at the time of writing this dissertation. A test for malicious behavior of the uploaded CA is neither provided in LonWorks nor in KNX. Thus, no protection against e.g. trojan horses or security concerns regarding resource access, storage of sensitive information, content security and availability exist.

Table 4.3 summarizes the applicability of software protection techniques to SACs, ICDs, and MDs. The software of a MD typically consists of an OS and the management software. The security of the OS has to be provided by the system administrator. The

−: not applicable ∼: applicable with restrictions +: applicable

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

Table 4.3: Comparison of Software Protection Techniques: Applicability to Sensors, Actuators and Controller Devices, Interconnection Devices, and Management Devices

security of the management software itself can be established using SBes and hardware supported mechanisms. Since the software of ICDs is rather fixed, SCA, SMT, SCC or ASC may be used. Besides, ICD software is less susceptible to tampering since it is reused over a broad range of devices. Modifications are thus less likely to go unnoticed. Also the effort (and cost) required for implementing the mechanisms mentioned above can be divided among a larger number of devices. Since usually a large number of SACs is deployed, their functionalities depend on the application and CAs are likely to be developed by various manufacturers, static or human assisted methods seem too expensive and ineffective to deploy. A reasonable combination, however, seems appropriate and is presented in the next section.

4.6 Summary

As can be seen, 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 threats discussed in Section 3. 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.
- Third, even if perfectly applied, none of these techniques can offer a protection against the widespread social engineering attacks, where users are somehow fooled into revealing something they should not reveal. The only way to deal with those attacks is user education or so restricted security permissions, that effective use of the underlying system is not possible.

5 Secure Control Application Architecture

As presented in the previous sections, today's BAS do not provide sufficient protection against device attacks to satisfy the demands of security critical applications. Secure CAs have only been considered as a side issue at best in open standards. Traditional IT software methods to improve application security do not provide sufficient protection or cannot be easily applied to the BAS domain due to e.g. limited system resources. Hence, it is necessary to establish a security concept, that uses the domain knowledge being implicitly present in the discussed standards to provide overall protection for CAs.

A secure BAS architecture rests on three different building blocks:

1. Secure BAN: Mechanisms for secure data exchange shall be provided.
2. Secure CAs: It shall be guaranteed that malicious CAs can not compromise the security of the system.
3. Attack detection: Mechanisms to detect attacks such as DoS within a reasonable detection latency are required.

A secure BAN and detection of BAN DoS attacks are described in [58]. The goal of this section is not to provide yet another new method to prevent single attack types like buffer overflows, but to provide a solid and reasonable approach for secure CAs with respect to the requirements presented in Section 3.

Thus, a secure architecture being adaptable to all common BAS standards has to be established. This architecture needs to cover BAS specific constraints, provide a security policy and a secure software environment. Besides, possible attacks need to be detected.

Hypothesis 5 *Only hybrid software protection mechanisms can provide an overall CA security.*

To ease the development of secure CAs hosted on SACs, a software environment being able to prevent or detect the discussed attacks is required. Basically, protection against software attacks shall be provided, but with other attack scenarios such as physical or side channel attacks in mind.

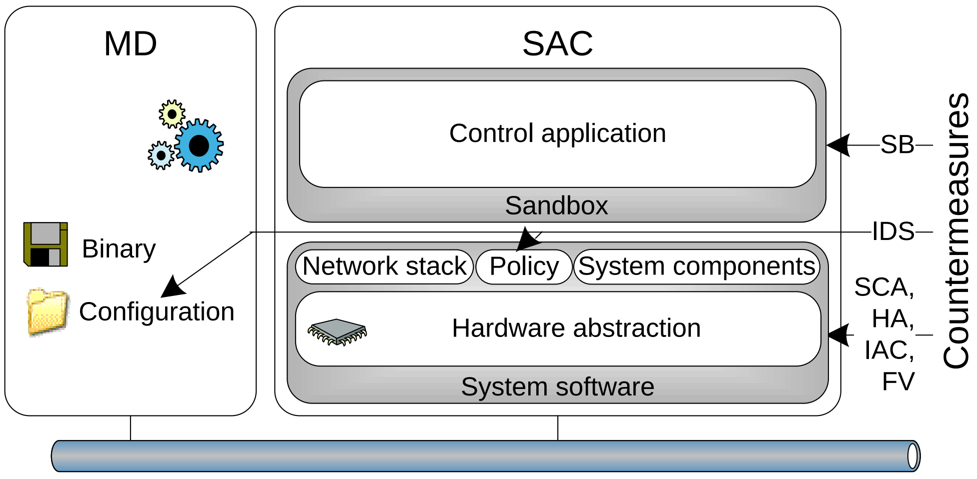


Figure 5.1: Secure Control Applications

As outlined in Figure 5.1, the software of a SAC consists of three major components, each imposing an additional security barrier to the overall security and limiting possible security threats:

- A tight and secure system software provides controlled access to system resources. Its security is analyzed using human being based IAC, SCA using automated tools as well as FV.
- A SB restricts the execution of customizable CAs. Basically, this uploadable code shall be allowed to perform any desired action. However, to prevent security flaws it is under continuous control of the system software during runtime, determining whether it is allowed to execute a desired function or not. The SB is also designed to support the rapid development of CAs. It provides a clear abstraction of the underlying hardware and offers interfaces to the system software. The developer is thus relieved of any hardware or device specific details and can focus on the

application development. This allows portability of applications between devices offering the same SB.

- A configuration has to be provided to the system software during upload of a CA that defines its basic policy (i.e. normal behavior). Any abnormal behavior can then be detected by the system software using an AIDS. Thus, limits to e.g. network or processing resources may be defined, which are enforced at runtime.

To further limit possible attack scenarios, the use of a HA based hardware is recommended.

The following sections describe the security architecture in more detail. Section 5.1 defines a generic application model, required to develop a secure system being used for the different BASs. Section 5.2 describes how to define a security policy using security attributes, which allow a formal way to formulate security requirements. Section 5.3 outlines the software environment being needed to securely execute CAs and enforce the security policy. Finally, Section 5.4 outlines the methods to detect possible attacks.

5.1 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. A generic application model is required that can accommodate common functionalities found in BASs. The employment of this model promises on the one hand a security architecture being compatible with all open BASs and on the other hand also several benefits such as a central point for configuration and system access [137].

One possible way to represent such a generic application model are ontologies. "(An ontology is) a shared and common understanding of a domain that can be communicated between people and heterogeneous and distributed systems [48]." Thus, an ontology is basically a way to store, organize and represent knowledge. More specific, concepts belonging to a particular domain and their relations can be defined in an abstract way, thus forming a model of this domain. Additionally, ontologies allow to reason on the stored data, so that an automatic generation of new information is possible. Ontologies are commonly designed and developed with tool support and are often based on the Resource Description Language Schema (RDFS) [13] and the Web Ontology Language (OWL) [116] as description languages. Probably the most prominent tool for this purpose is *Protege*[1] which was developed at Stanford University. Ontologies are used in various scientific fields, ranging from biology to linguistics. Also, technical systems are considered to be augmented with their help. In the industrial automation field, ontologies allow to access fieldbus device information (device descriptions) using Semantic Web technologies and thus easier creation, processing and management of this information becomes possible (cf. [67]). In [155], an ontology called DomoML representing data of household appliances and their environment (e.g. rooms, furniture) is defined. The main target is human home interaction with the goal to improve pervasiveness and interoperability of domestic devices. [127] describes BASont, a modular, adaptive BAS ontology that addresses use cases from design, to commissioning, to operation and refurbishment. Within the ThinkHome project[2], a multi-agent system, a knowledge base and ontology, control strategies and a simulation environment are being researched with the goal to utilize artificial intelligence to improve control of home automation functions provided by dedicated automation systems. [105] describes the developed ontology. The Secure and Semantic Web of Automation project[3] addresses interoperability issues that arise due to the plethora of commonly used home and building automation systems by realizing the idea of universally interconnected things through the use of Semantic Web technologies. [49] gives a case study for interoperability on management level of BACnet and OPC UA. A similar approach as DomoML but for industrial and building automation systems is outlined in [18]. The main aim is to establish interoperability among heterogeneous automation networks at the level of web services. [57] presents first experiences on the concept of integration ontologies, targeting the interoperation of different ontologies. Ontologies also allow the representation of application specific knowledge in a structured way. However, no ontology describing CA security is available and also a generic application model is missing. Nevertheless, for providing security in BASs, ontologies seem promising regarding the

[1] `http://protege.stanford.edu/`, Last access: 2015/08/16

[2] `https://www.auto.tuwien.ac.at/projectsites/thinkhome/overview.html`, Last access: 2015/08/16

[3] `https://www.auto.tuwien.ac.at/projects/viewBlog/40/`, Last access: 2015/08/16

heterogeneous network and device infrastructure often found there. The remainder of this section focuses on a generic application model, which is being described in a formal way enriched with illustrative listings.

As described in Section 2.3, the application models in open standards are different. They do not only use different communication services at the application layer but also different data structures to store application data. To provide a unified view and CA security, a mapping between the different application models is necessary. In a specific installation, this means that the network-visible data structures of the application data, i.e. the DPs, need to be mapped from one protocol to the other (DP mapping). Likewise, also the services used to access them have to be translated. Additionally, it has to be defined which DPs shall be used for data exchange. This step is referred to as binding. In fact, binding and configuration of thousands of devices in larger installations is time consuming and error prone. [34], [35] describe an automated way to enhance the situation.

The basic idea of such a generic application model for BAS is to separate generic information 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.

An application model thus can be used to abstract an existing BAS installation and represent this particular installation in a generic form. 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. Considering the integration of heterogeneous networks, the employment of a generic application model holds major benefits (cf. Figure 5.2).

It is no longer necessary to define multiple security policies and mapping rules covering each protocol with each protocol (pairwise for each two protocols), but it is sufficient to map each protocol to the abstract representation (the generic application model), which acts as a common base for all protocols. Furthermore, if a new technology is considered for integration, again just a single mapping needs to be defined (i.e. from the technology

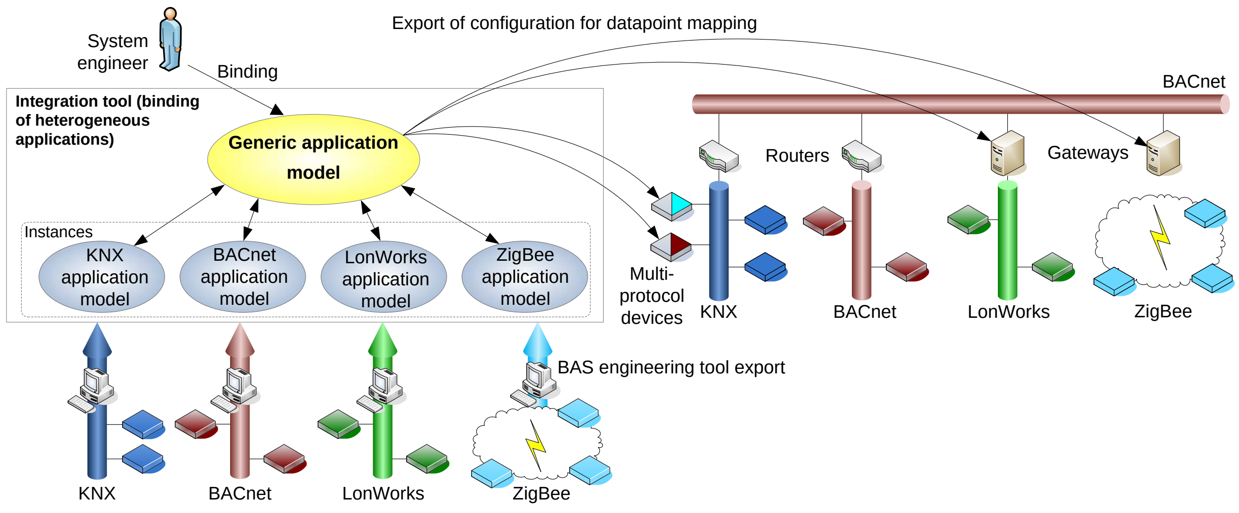


Figure 5.2: Building Automation System Integration and Advanced Use Cases

to the generic application model), while an approach without common representation would require additional mappings. Therefore, future extensions are highly facilitated.

Besides, integrated BAS can be configured centrally by accessing and modifying the application model only. For all protocols that are employed, an (automatic) translation of the (centrally managed) configuration data into the technology-specific form (i.e. the instances representing the used BAS) becomes possible. This central management approach facilitates BAS management and guarantees system consistency. Additionally, also the gateway configuration can be derived automatically. Consider, for example, a ZigBee light switch that shall be integrated into a KNX lighting system. To achieve this, the engineer now binds the generic DPs of these two functions. After having finished the binding of all desired functions, it is possible to automatically generate and export this binding in form of configuration data. This configuration data can then be loaded into gateways or multi-protocol devices respectively. [147] presents such a reliable and flexible gateway between KNX and ZigBee.

This generic application model [132] (cf. Figure 5.3) is expressive enough to be able to model all different types of distributed CAs typically found in the building automation domain. It allows an easy mapping between the generic CA model and the standard-specific CA models of the most popular open BAS standards (cf. Section 2.3). At the same time, the model is generic enough to accommodate to the CA models of future BAS

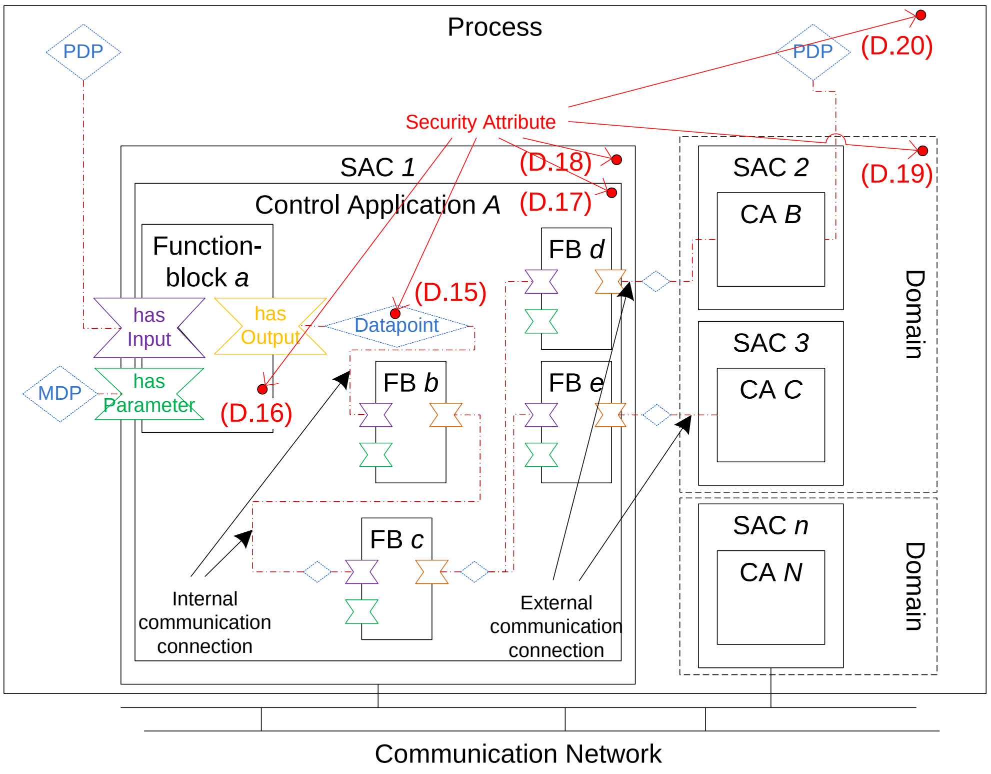


Figure 5.3: Generic Application Model

standards. In this model, the nomenclature is based on the one used in IEC 61499 [83], but modifications were made with respect to the building automation domain vocabulary. It is specified using a formal way accompanied by a textual representation.

$$\mathrm{DP}: \quad < p, \ldots > \tag{D.1}$$

$$\mathrm{FB}: \quad \{dp_i | dp_i \in \mathrm{DP}, i \in \mathbb{N}\} \tag{D.2}$$

$$\mathrm{CA}: \quad \{fb_i | fb_i \in \mathrm{FB}, i \in \mathbb{N}\} \tag{D.3}$$

$$\mathrm{SAC}: \quad \{\mathrm{CA}\} \tag{D.4}$$

$$DOMAIN: \quad \{sac_i | sac_i \in \mathrm{SAC}, i \in \mathbb{N}\} \tag{D.5}$$

$$PROCESS: \quad \{domain_i | domain_i \in DOMAIN, i \in \mathbb{N}\} \tag{D.6}$$

The application model 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)).

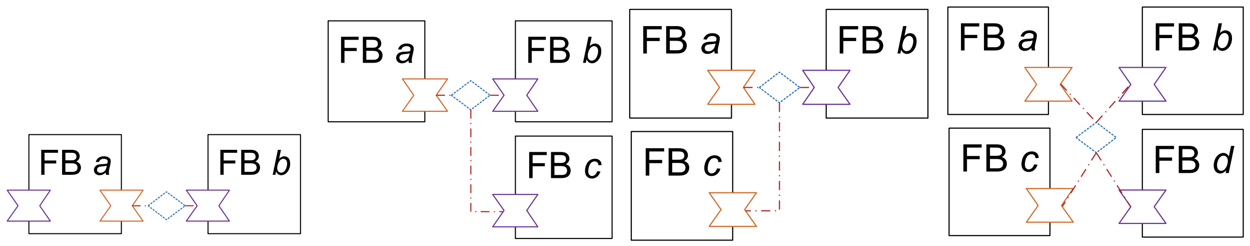
A control domain can logically be seen as a grouping of distributed BAS nodes (i.e. SACs), which realize a common functionality (e.g. HVAC, lighting). It consists of at least one SAC, which itself may arbitrarily belong to multiple other domains and which interacts with the environment (cf. Definition (D.5)).

Each SAC hosts exactly one CA[4] (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 5.3. In such a way, the cardinality of the relation between FBs is not limited and so the definitions of the well known *1:1*, *1:n*, *n:1*, and *m:n* relations are possible (cf. Figure 5.4a–5.4d).



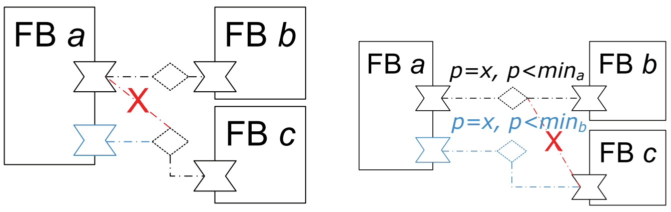
(a) 1:1 (b) 1:n (c) n:1 (d) m:n

Figure 5.4: Communication Connections

Note, that a binary relation is a relation between exactly one DP and one FB. Thus, a setup as shown in Figure 5.5a where a single relation links two different DPs (i.e. the relation with red cross) is not possible. It has to be realized by adding an additional *hasOutput* relation to *FB a*. These two DPs are either equal and thus only a single communication connection is required, or unequal and have to be treated differently by the FB anyway. Communication connections are called internal if their corresponding FBs are located within the same CA or device, respectively. They are called external if the FBs do

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

not share the same CA and communication between SACs via the network occurs. A DP refers to a single datum (i.e. tuple) of the CA whose structure and semantics (e.g. present value p, encoding, unit, minimum value, maximum value) are fully specified (cf. Definition (D.1) and [14]). A DP represents the same datum with all the same values for every involved relation. A setup (i.e. the relation with red cross) as shown in Figure 5.5b is not allowed. Another DP would have to be added, perhaps with the same present value but other differences to allow varying input constraints for *FB b* and *FB c*. The Physical or Process Datapoint (PDP) refers to a DP located within the process (e.g. temperature value t in room x). Therefore, special hardware is required to access it. A Management Datapoint (MDP) corresponds to a configuration parameter of a FB. PDPs and MDPs can only be connected to a single FB.



(a) Case I (b) Case II

Figure 5.5: Impossible Communication Connections

In Figure 5.3, PDPs, 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.

The generic application model is defined in the following machine readable form (cf. Listing 5.1):

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ApplicationModel xmlns:xsi="http://<!-- [...] -->">
  <!-- BACnet application model -->
  <BACnet>
    <Object Name="Lighting-Output" Object_Type="LIGHTING_OUTPUT">
      <Property id="bacnetdp_1" Name="Object_Identifier" Datatype="
          BACnetObjectIdentifier" />
      <Property id="bacnetdp_2" Name="Present_Value" Datatype="REAL" />
      <Property id="bacnetdp_3" Name="Off-Delay" Datatype="REAL" />
  <!-- [...] -->
  </BACnet>
```

```

<!-- KNX application model -->
<KNX>
  <DataTypes>
    <DataType id="1" name="Boolean" format="1" unit="-">
      <Range>
        <Value id="0">0</Value>
        <Value id="1">1</Value>
      </Range>
    </DataType>
    <!-- [...] -->
  </DataTypes>
  <DatapointTypes>
    <DPT id="1.001" Name="DPT_Switch" DataTyperef="1">
      <Encoding valueref="0" description="Off"/>
      <Encoding valueref="1" description="On"/>
    </DPT>
  <!-- [...] -->
  </DatapointTypes>
  <!-- [...] -->
  <FunctionalBlock ObjectType="417" Name="FB_Light_Actuator" Title="Light
      Actuator">
    <Functional_specification>
      Switch the light according to the received value on the OnOff
          datapoint
    </Functional_specification>
    <DataPoints>
      <Input>
        <DP id="knxdp_1" Name="OnOff" Abbr="OO"
          Mandatory="true"
          Description="To switch the light On or Off"
          DatapointTypeRef="1.001" />
        <!-- [...] -->
      </Input>
      <Parameter>
        <DP id="knxdp_2" Name="Timed Duration" Abbr="P1"
          Mandatory="false" Description="Duration to switch..."
          DatapointTypeRef="7.005"
          Default="0" />
        <!-- [...] -->
      </Parameter>
    </DataPoints>
  <!-- [...] -->
  </FunctionalBlock>
</KNX>

<!-- EnOcean application model -->
<EnOcean> </EnOcean>

<!-- LonWorks application model -->
<LonWorks> </LonWorks>

<!-- ZigBee application model -->
<ZigBee> </ZigBee>

<!-- Generic aplication model -->
<Generic>
  <!-- Stairwell light -->
  <Application Name="Stairwell light" id="3813-3_D-1-2">
    <FunctionBlock Name="Actuate light" />
    <FunctionBlock Name="Light actuator" id="3813-2_6_4_2"/>
    <FunctionBlock Name="Light control" id="3813-2_6_5-6">
```

```
71       <!-- Associate technology specific datapoints with generic datapoint -->
72         <ProcessDataPoint id="L_SET" Name="Light control setpoint" Type="Light
               ">
73           <BACnetDataPoint id="bacnetdp_2" />
74           <EnOceanDataPoint id="enoceandp_1" />
75           <KNXDataPoint id="knxdp_1" />
76           <LONDataPoint id="londp_1" />
77           <ZigBeeDataPoint id="zigbeedp_1" />
78         </ProcessDataPoint>
79         <ManagmentDataPoint id="PAR_OFFD" Name="Parameter off delay" Type="
               Time">
80           <BACnetDataPoint id="bacnetdp_3" />
81           <EnOceanDataPoint id="enoceandp_2" />
82           <KNXDataPoint id="knxdp_2" />
83           <LONDataPoint id="londp_2" />
84           <ZigBeeDataPoint id="zigbeedp_2" />
85         </ManagmentDataPoint>
86       </FunctionBlock>
87 <!-- [...] -->
```

Listing 5.1: Extract of Application Model in XML Format

In the listing, first the `Lighting-Output` object of BACnet is being modeled. The two BACnet DPs `Present_Value` and `Off-Delay` are declared (cf. lines 4–10). Then, the KNX application model is specified. Lines 14–29 specify the KNX DPT `DPT_Switch`, lines 31–52 define the FB `FB_Light_Actuator` containing the DPs `OnOff` and `Timed Duration`. The EnOcean, LonWorks and ZigBee application model is contained in lines 55–62. Line 67 declares the generic application `VDI 3813-3: D-1-2 Lighting control manual with time-controlled switching off (stairwell light)` containing the FBs `Actuate light`, `Light actuator` and `Light control`. The latter contains the PDP `L_SET` and MDP `PAR_OFFD`. The mapping of technology specific DPs to `L_SET` is shown in lines 72–78. The mapping of technology specific parameters to `PAR_OFFD` is contained in lines 79–85.

Figure 5.6 shows an example implementation for use case ⊗. Its formal specification is shown in Definition D.7

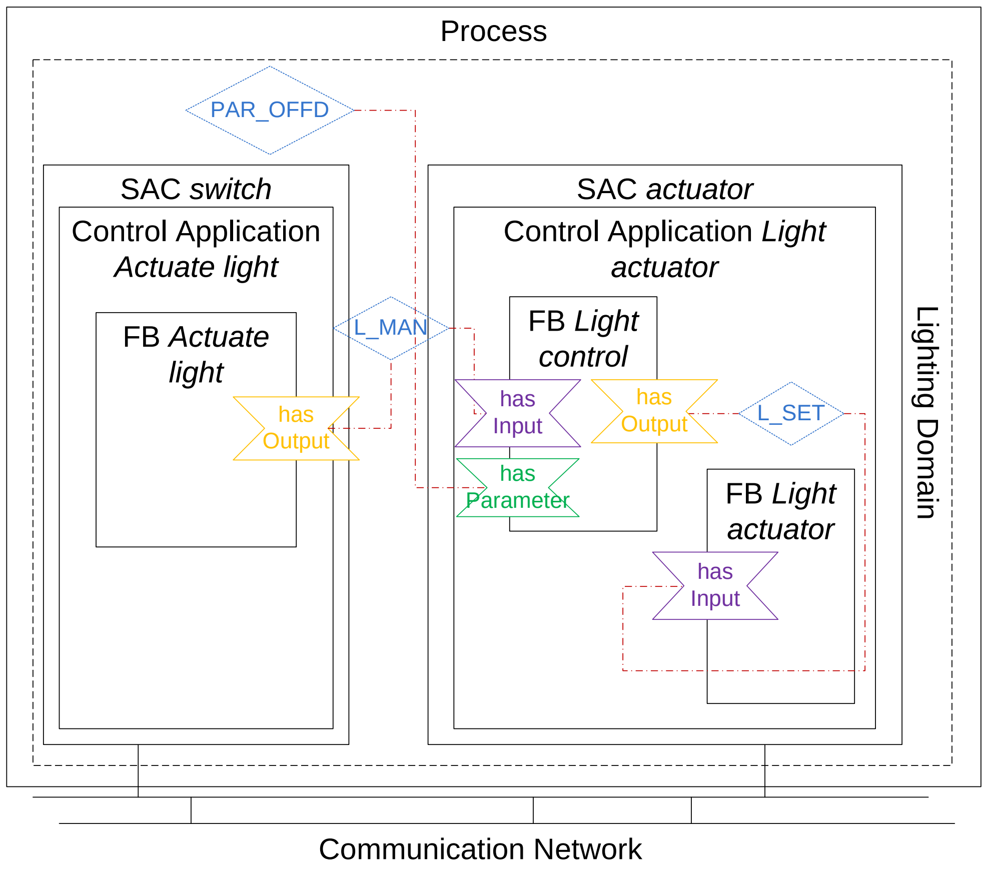


Figure 5.6: Generic Application Model: Example for Use Case ⊗

$$
\begin{aligned}
\text{MDP } PAR\_OFFD : &\quad < p, \ldots > \\
\text{PDP } L\_MAN : &\quad < p, \ldots > \\
\text{PDP } L\_SET : &\quad < p, \ldots > \\
\text{FB } Actuate\ light : &\quad \{\text{DP } L\_MAN\} \\
\text{FB } Light\ control : &\quad \{\text{DP } PAR\_OFFD,\ \text{DP } L\_MAN,\ \text{DP } L\_SET\} \\
\text{FB } Light\ actuator : &\quad \{\text{DP } L\_SET\} \\
\text{CA } Actuate\ light : &\quad \{\text{FB } Actuate\ light\} \\
\text{CA } Light\ actuator : &\quad \{\text{FB } Light\ control,\ \text{FB } Light\ actuator\} \\
\text{SAC } switch : &\quad \{\text{CA } Actuate\ light\} \\
\text{SAC } actuator : &\quad \{\text{CA } Light\ actuator\} \\
LIGHTING\ DOMAIN : &\quad \{\text{SAC } switch,\ \text{SAC } actuator\} \\
PROCESS : &\quad \{Lighting\}
\end{aligned}
\tag{D.7}
$$

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.

5.2 Software Security Policy

The second step to be able to establish adequate countermeasures against the discussed security threats, is to define a security policy. 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 p_i of DPs DP_i, where security requirements for the conditions derived from the policy can be defined, formulated and finally evaluated.

5.2.1 Security Attributes

The model of an integrated BAS as presented in Section 5.1 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 $\bullet_1 \dots \bullet_{n-1}$ which relate to these conditions:

$$security\_attribute\ s :< p, cond_1, \dots, cond_n, \bullet_1, \dots, \bullet_{n-1} > \qquad \text{(D.8)}$$

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.

$$\begin{aligned} &cond(f(p_\alpha(t)), g(p_\beta(t)), \circ) \mid \\ &\qquad\qquad p_\alpha(t) \in A, p_\beta(t) \in B, \\ &\qquad\qquad f : A \to C, g : B \to C, \\ &\qquad\qquad \circ : f \times g \to \{true, false\} \end{aligned} \tag{D.9}$$

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

$$\begin{aligned} &cond(check\_access(p_\alpha(t)), \{read\}, =) \\ &\quad = check\_access(p_\alpha(t)) = \{read\} \\ &\quad = \begin{cases} true \text{ if } p_\alpha(t) \text{ is readable} \\ false \text{ otherwise} \end{cases} \\ &cond(check\_access(p_\alpha(t)), \{write\}, =) \\ &\quad = check\_access(p_\alpha(t)) = \{write\} \\ &\quad = \begin{cases} true \text{ if } p_\alpha(t) \text{ is writeable} \\ false \text{ otherwise} \end{cases} \end{aligned} \tag{D.10}$$

$$\begin{aligned} &cond(check\_value(p_\alpha(t)), g(p_\beta(t)), \circ) \\ &\quad = \circ(p_\alpha(t), g(p_\beta(t))) \end{aligned} \tag{D.11}$$

$$\begin{aligned} &cond(check\_history(p_\alpha(t)), g(p_\beta(t)), \circ) \\ &\quad = \circ(p'_\alpha(t), g(p'_\beta(t))) \end{aligned} \tag{D.12}$$

$$\begin{aligned} &cond(check\_accesstime(p_\alpha(t)), g(p_\beta(t)), \circ) \\ &\quad = \circ(check\_accesstime(p_\alpha(t)), constant) \\ &\quad = \begin{cases} true \text{ if } check\_accesstime(p_\alpha(t)) \circ constant \\ false \text{ otherwise} \end{cases} \end{aligned} \tag{D.13}$$

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

- Access rights: (D.10) 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.11) 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.12)

- 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.13)

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

$$eval(s) = \begin{cases} true \text{ if } cond_1, \ldots, cond_n = true \\ false \text{ otherwise} \end{cases} \tag{D.14}$$

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 5.3):

$$sec\_attr_{DP} : s \mid \{\forall p_i \in DP \land p_i = p, \forall i = 1, \ldots, n\} \tag{D.15}$$

$$sec\_attr_{FB} : s \mid \{\forall p_i \in \text{FB}\} \tag{D.16}$$

$$sec\_attr_{CA} : s \mid \{\forall p_i \in CA\} \tag{D.17}$$

$$sec\_attr_{SAC} : \{security\_attribute_{systemcalls}\} \tag{D.18}$$

$$sec\_attr_{DOMAIN} : s \mid \{\forall p_i \in DOMAIN\} \tag{D.19}$$

$$sec\_attr_{PROCESS} : s \mid \{\forall p_i \in PROCESS\} \tag{D.20}$$

- Security attributes at the DP level cover only local constraints of the present value p. (D.15)
- 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. Security attributes attached to FBs may also be used to handle priorities within the input and output relations. (D.16)
- 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.17)
- 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.18)

- 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.19)
- 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.20)

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 $DP \subseteq \text{FB} \subseteq CA \subseteq SAC \subseteq DOMAIN \subseteq PROCESS$.

Figure 5.7 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.

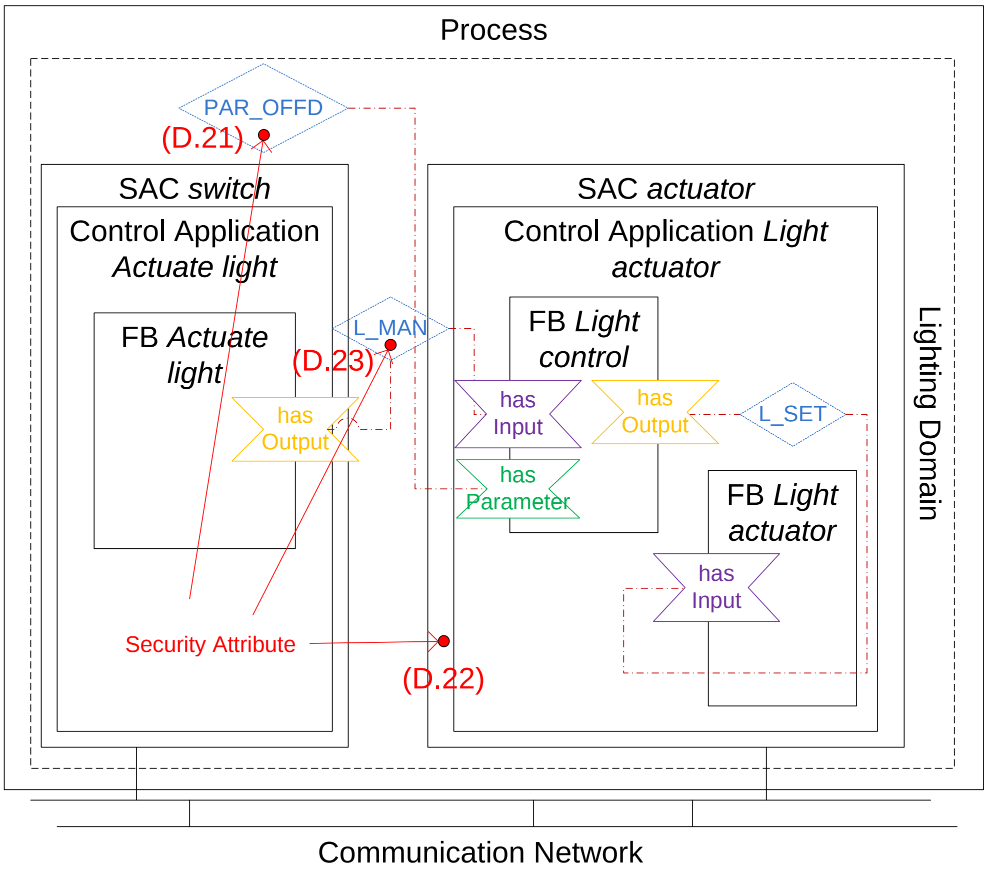


Figure 5.7: Software Security Policy for Use Case ⊗

$$s_{DP\ PAR\_OFFD} :< p_{PAR\_OFFD}, cond1_{p_{PAR\_OFFD}}, cond2_{p_{PAR\_OFFD}}, AND > \tag{D.21}$$

$$eval(p_{PAR\_OFFD}) \rightarrow cond1_{p_{PAR\_OFFD}}\ AND\ cond2_{p_{PAR\_OFFD}}$$

$$cond1_{p_{PAR\_OFFD}} : cond(p_{PAR\_OFFD}, maximum, \leq)$$

$$cond2_{p_{PAR\_OFFD}} : cond(p_{PAR\_OFFD}, minimum, \geq)$$

$$s_{SAC\ actuator} :< systemcall\_switchoutput, \tag{D.22}$$

$$check\_accesstime(systemcall\_switchoutput, maximum\_frequency, \leq) >$$

$$s_{Lighting\ DOMAIN} :< p_{L\_MAN}, cond_{p_{L\_MAN}} > \tag{D.23}$$

$$cond_{p_{L\_MAN}} : cond(p_{L\_MAN}, \{0,1\}, =)$$

5.2.2 Enforcement

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 [132]. 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 IDSs [111]: 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 intrusion detection system being connected to the network and having a global view of the exchanged process data (cf. Section 5.4). Further actions such as alarming the operator can then be taken.

To be able to enforce DP, FB, CA and device security attributes – which only affect a single device – it is necessary, that the execution of the CA can be limited and controlled and that a separation of the device's system software from the CA can take place. Note, however, that the hardware being used for SACs is very limited and any deployed security mechanisms must not exhaust a device's resources.

To reduce the security overhead it is necessary to discuss if and when an enforcement of security attributes needs to occur and when it does not provide an additional level of security. Within the application model, it is not necessary for internal communication connections to monitor or enforce their security attributes, since only a single CA is being executed and all data exchanged represent internal data being accessible to the CA anyway.

	Traditional CAs		Secure CAs	
	reading	writing	reading	writing
Insecure channel	yes	yes	yes	yes
Secure channel	yes	yes	no	yes

Table 5.1: Enforcement of Security Attributes on External Communication Connections

Table 5.1 describes the need to monitor and enforce security attributes for external communication connections between a secure CA and other SACs. Basically only those present values shall exist within a BAS which do not violate the security policy. It, however, depends on the particular structure of a BAS if this is possible or not.

If a secure channel, that uses physical or cryptographic mechanisms to provide authentication, data integrity, data freshness and data confidentiality, and a homogeneous BAS consisting of secure CAs solely are available, CAs only need to monitor and enforce security attributes when writing a DP. No malicious present values can be inserted into the BAS by reading operations.

If, however, an insecure channel is used that does not prevent malicious alteration of present values or communication with a possible insecure CA takes place, a secure CA needs to monitor and enforce security attributes both when reading and writing a DP.

Nevertheless, such a desirable homogeneous network – consisting of secure CAs solely which rely on secure communication – will only be present within completely new installations.

5.3 Secure Software Environment

As described in Section 2.3, today's BASs are implemented using various technologies (e.g. BACnet, EnOcean, KNX, LonWorks, ZigBee), each with own benefits and features. ICDs perform a generic task which is quite similar for all different installations. Their implementation and possible security measures can be performed very generic and do not need to be adapted to individual setups. SACs need to fulfill many different automation tasks which are not predefined by the BAS architecture per se. Traditionally, SAC and CA development is quite a complex task which requires profound expertise. The first time creation of CAs is highly technical since often low level constructs and detailed knowledge about the underlying control network stack are required. Moreover, even if fulfilling the same tasks, CAs are not compatible and the knowledge of CA developers in one technology cannot be easily applied to another BAS because of the different network protocol stacks and CA models. Thus, even the design of a simple stairwell lighting application with configurable lighting duration provides an unnecessary high barrier for developers. A traditional deep integration is far from being straightforward, requires the use of gateways and maintenance of huge mapping tables. Obviously, security is getting more and more important, however, it has not been covered at all in open BAS standards. Thus, dealing with the indispensable security requirements is left open for the security unexperienced application developer, who often is not able to cope with manifold threats and attack possibilities. The left side of the dashed line in Figure 5.8 shows this traditional CA concept, while the right side describes the secure software environment.

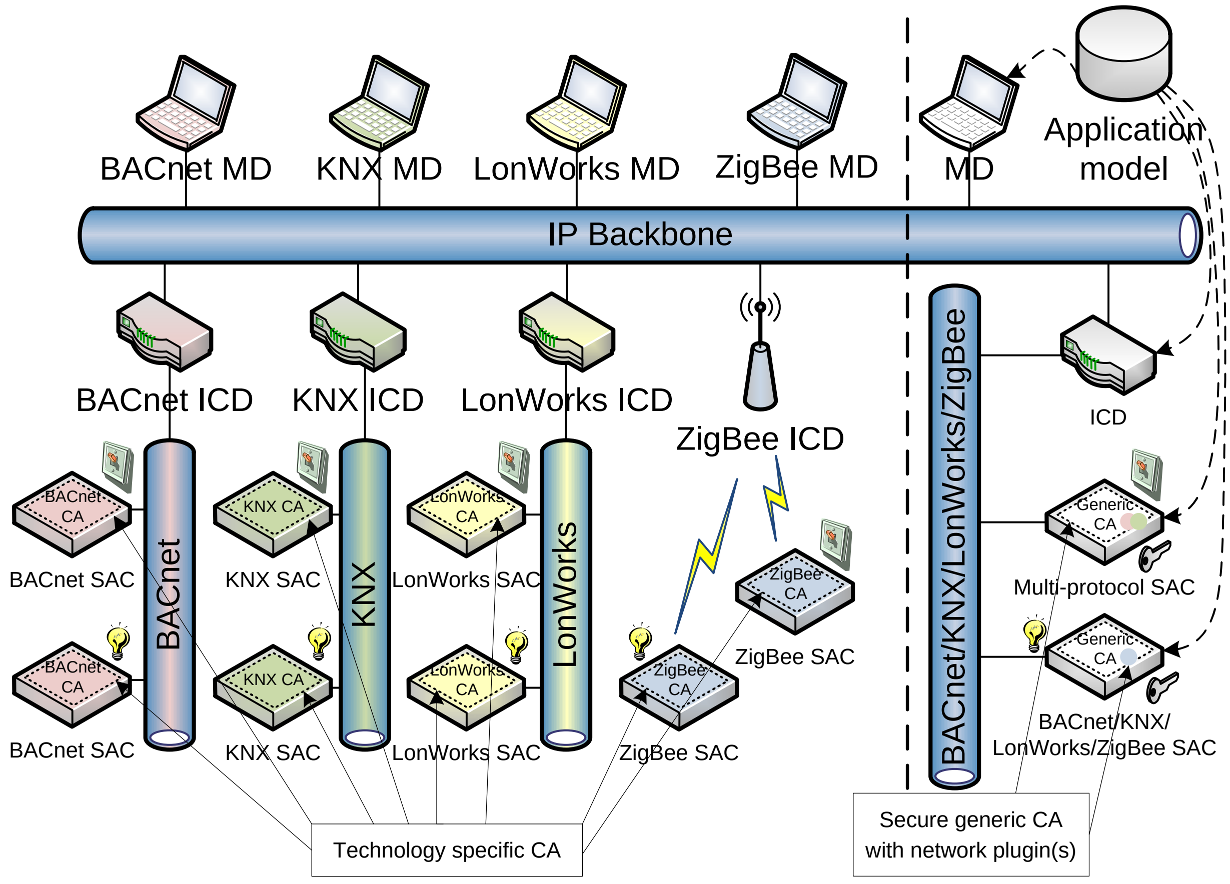


Figure 5.8: Control Applications in Building Automation Systems

To ease the development of secure CAs for low end (<100 KBytes memory, <100 MHz CPU speed) SACs, a secure software environment being able to deal with the threats discussed in Section 3.3 is needed [130]. Application designers shall be supported to allow rapid innovation and implementation, configuration, deployment and execution of arbitrary, uninspected and uncertified software without compromising the overall system security. Besides, this possible erroneous or malicious software shall not compromise the overall system security. Not only attacks evolving from accidental software faults are to be prevented but also attacks resulting from intentional malicious software. Such a solution needs to be low cost and no special hardware modifications are to be required, thus allowing easy and compatible integration.

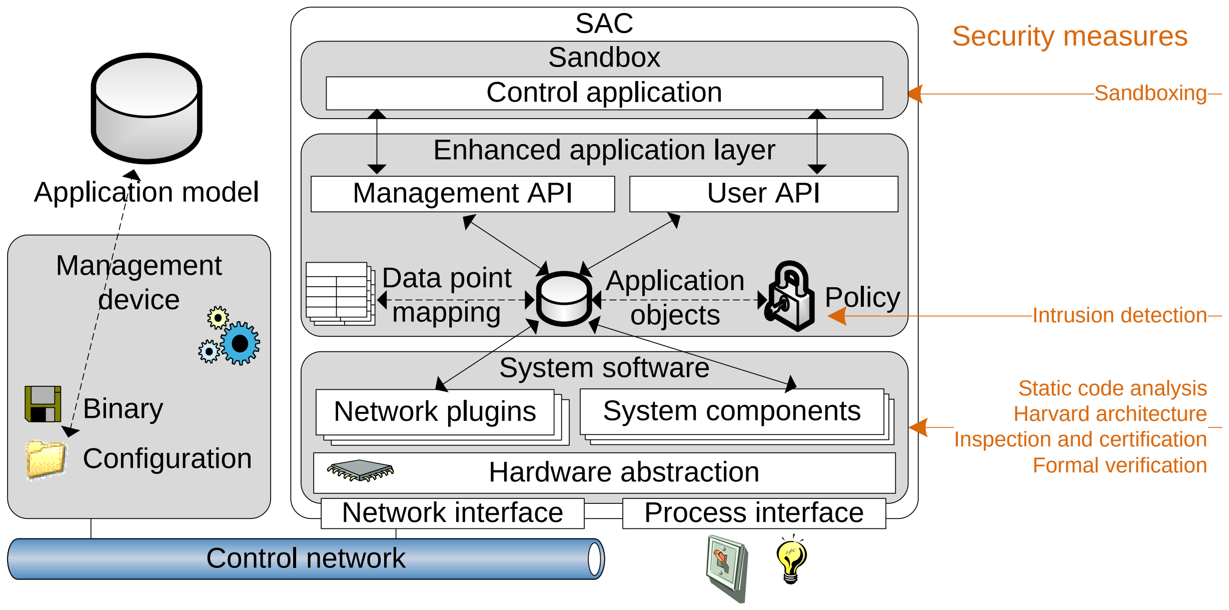


Figure 5.9: Architecture of Sensors, Actuators and Controllers

As resulted in the evaluation in Section 4, no single software protection technique is able to meet these CA security requirements. Static software methods such as e.g. SCA being integrated into the development process could report detected security vulnerabilities to the programmer. They could also be used to detect certain types of malicious code. However, as mentioned, the results of those methods in general cannot be sound and complete and do not seem to provide serious protection against all security threats. Thus, monitoring the CA's actions and allowing or denying them at runtime is a more powerful approach. However, implementing such a SB on lean embedded devices is made difficult by resource constraints. Certain functions may not be available at all due to missing hardware support. Thus, a reasonable combination and extension of the different techniques consisting of a priori analysis and run time monitoring is necessary to provide protection against software attacks primarily, but with other attack scenarios such as physical or side channel attacks in mind.

The idea is to separate the system software (including network stacks) running on the device from the CA as well as the node configuration. This model encapsulates the system software entities in a way which is inspired by the object-oriented paradigm. It also makes use of a generic application model, which describes the application behavior and provides relevant configuration and security parameters.

As outlined in Figure 5.9, the architecture of a secure SAC 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 (cf. Section 5.3.1).
- An enhanced application layer stores the application objects, their DP mappings as well as the security policy for the CA (cf. Section 5.3.2).
- 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 hard- 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. This also allows portability of applications between devices offering the same SB (Section 5.3.3).

5.3.1 System Software

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. Besides, it is designed to maximize code reuse. A change in the combination of the software modules or a change in the hardware design should only require a minimum of modification to the software. It takes care of initialization tasks and manages the available resources of the system.

To access the hardware in an independent and modular way at the lowest level, a Hardware Abstraction Layer (HAL) hides the peculiarities of basic 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. To provide communication security, secure algorithms and cryptographic functions are contained. The network plugins handle the mapping of technology specific application models to the generic application objects. Basically, they keep the application objects up to date when corresponding network messages are received and trigger network messages when the application object is changed by the CA. Since the CA is separated from the system software, it can be ensured that all (bus-)communication is standard compliant and the latter has to be certified only once.

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.

5.3.2 Enhanced Application Layer

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. Note, that multiple network plugins may be deployed, allowing the design of multi-protocol devices [156]. 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 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. Such a generic API also increases portability and compatibility of applications on different platforms and technologies. The design of the user API will vary for different application fields. Nevertheless, it should be kept as general as possible and, in the ideal case, the user API should operate at a very high abstraction level. For example, in the case of networking, only valid incoming messages shall be reported to the CA, while erroneous receptions will automatically be rejected and negatively acknowledged. This way, the CAs can be kept simple and at the same time can concentrate to perform intended actions, only.

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. Moreover, this tool allows customization of an application by adapting runtime parameters for the SB defined by its configuration.

5.3.3 Sandbox

A SB executes the CA in a controlled way and is often used for untrusted programs or untested code with the essential benefit that the system outside the SB is protected from (malicious) actions by the CA. The behavior of the program in the SB can be monitored and controlled and mechanisms to provide memory protection can be integrated. Despite enforcing limits derived from the security policy of a SAC, the SB also may provide watchdog functionality to e.g. reset an application if no specified action occurs within a defined time. Additionally, the execution of CAs can be limited to those being signed and containing valid cryptographic signatures to provide support for DRM. Besides, a SB may also be designed to support the rapid development of CAs. It interfaces the enhanced application layer which provides a clear abstraction of the underlying hard- and software by an object-oriented access to the application objects. The application designer is thus relieved of any hardware or device specific details and can focus on the application development. This also allows portability of CAs between devices offering the same SB. It is obvious that such a SB has to be designed for little memory usage and low overhead. While this approach may at first seem inappropriate for a low-end SAC due to the relatively high resource requirements of such techniques, it offers outstanding possibilities.[5] Besides, the resource requirements can be lowered to a significant extent with the acceptance of certain limitations. The SB does not need to support fully fledged programming models, since the desired operations, especially in control tasks, are often quite simple. For such purposes, CAs more or less consisting of a sequence of simple operations may be sufficient, which can be supported by a resource-saving SB implementation.

5.3.4 Configuration and Management

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 (cf. Listing 5.1), that hosts the technology specific application models (i.e. BACnet objects, KNX standardized FBs, LonWorks 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

[5] Sandboxes might even be designed to be capable of executing native BAS code and to allow the integration of already existing applications.

be gained from methods described in Section 4. 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 (cf. Section 2.2) already provides a generic view with valuable information regarding CA behavior. This information solely has to be provided to the security system.

As always with security issues, a monitoring of policy violations has to be done and appropriate measures have to be specified. If misbehavior is detected, it may not always be considered a severe attack. Therefore, in the implementation different alert levels need to be defined and an attack (prevention and) detection system needs to be deployed.

5.4 Attack Prevention and Detection

5.4.1 Requirements

Securing the BAN by providing secure communication as well as verifying the identity of the involved network nodes (i.e. authentication) prevents security threats like unauthorized interception (e.g. network sniffing), modification (e.g. man-in-the-middle attacks) and fabrication (e.g. replay attacks). As introduced, secure communication can be provided by employing cryptographic algorithms and secured channels.

Providing secure devices and preventing software attacks can be partly achieved using the secure CA architecture presented in the sections before. Code injection or algorithm weaknesses, for example, can likewise be avoided as the requirements such as *FR–memory access* or *FR–protection of environment* can be fulfilled using SBes.

Nevertheless, network as well as device attacks exist, that cannot be prevented using such methods. For BANs typical representatives are interruption attacks, which have the objective of making a service or data unavailable [126]. These attacks are also referred to as DoS attacks. In order to interrupt the communication in a BAN, the adversary is trying to waste network and system resources to prevent the SAC from performing its expected function. DoS attacks are always hard to handle, which is especially true for SACs where nodes are subject to stiff resource limitations.

Regarding device attacks, Section 5.2.2 discussed the enforcement of security attributes for CAs. Attacks or violations on control domain or process location (cf. D.19 and D.20) can only be detected or prevented by a system having a global view of exchanged process data. Besides, such a system is also needed to fulfill *FR–communication relationship*, *FR–availability*, and partly *FR–low level functionality access*.

Hence, an attack prevention and detection system has to

- prevent attacks on security attributes D.15, D.16, D.17 and D.18 using the secure CA architecture and additionally report those violations,
- detect attacks on security attributes D.19 and D.20 and additionally report those violations,
- provide a reasonably small detection latency to be able to react to security attacks,
- be operable without influencing the normal operation of a BAS regarding its performance,
- be secured with respect to attacks directed towards its operation,
- support various information sources such as information regarding communication within a BAN, data coming from ICDs such as a firewall or data derived from a security policy, and
- be update-able with respect to new attack scenarios as well as being flexible with respect to changed system behavior.

5.4.2 Intrusion Detection Systems for Building Automation Systems

The main objective of an IDS is to detect abnormal network communication as well as abnormal behavior of devices. After having detected such an abnormal situation, it must be determined whether it has to be considered as an attack. If it is considered as an attack, countermeasures can be initiated to minimize its consequences. Available BAS do not provide any measures to prevent or detect attacks. A mechanism to provide effective protection is presented in this section.

In the IT domain, various different, well-established IDS exist (e.g. SNORT[6], TC-PLogD[7], Autonomous Agents for Intrusion Detection (AAFID)[8], Cyberarms IDDS[9], GFI EventsManager[10], KFSensor[11], NIDS Sax2[12], Secondary Heuristic Analysis for Defensive Online Warfare (SHADOW)[13]), which due to the radically different system environment cannot be trivially adopted. Nevertheless, whenever possible available IDS components shall be used.

An IDS commonly consists of four components [107] described in the following.

- The data gathering component is responsible for collecting the data by observing the network traffic as well as the behavior of the different network devices. IDS are often classified according to the type of data collection (i.e. the location of the data gathering components). For the BAS domain, host-based, networked-based and hybrid systems are relevant.

  A host-based IDS is executed on a single device and tries to discover abnormal activities [50]. These methods typically observe the activities of the OS and its applications by monitoring changes on system files, system calls as well as logs. Host-based IDSs typically are used for non network communication related checks and are especially applicable for security critical devices (e.g. key servers, gateways). Three

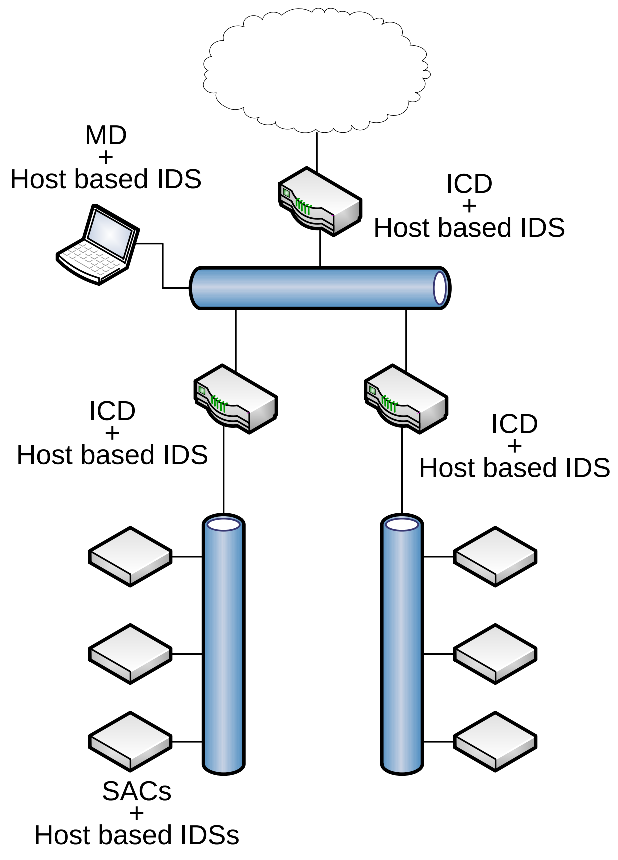
[6] `http://www.snort.org/`, Last access: 2015/08/03
[7] `http://www.securiteam.com/tools/2JUPQQKQ0M.html`, Last access: 2015/08/03
[8] `http://www.cerias.purdue.edu/site/about/history/coast/projects/aafid.php`, Last access: 2015/08/03
[9] `http://cyberarms.net/features/key-features.aspx`, Last access: 2015/08/03
[10] `http://www.gfisoftware.de/eventsmanager/esmfeatures.htm`, Last access: 2015/08/03
[11] `http://www.keyfocus.net/kfsensor/`, Last access: 2015/08/03
[12] `http://ids-sax2.com/`, Last access: 2015/08/03
[13] `http://www.softpanorama.org/Security/IDS/shadow.shtml`, Last access: 2015/08/03

categories can further be distinguished. On system or OS level, the IDS monitors multiple processes and their interaction and compares the behavior pattern with a reference (profiling). IDSs can also monitor specific, security critical applications, but then need to be especially tailored. Finally, IDSs can also be used to regularly check the integrity of the device by e.g. comparing calculated checksums of files to reference values. This has the benefit, that the exact location of security violations can be identified easier. A violation, however, also might imply, that a security attack has already been successfully performed. Summing up, host-based IDSs due to their specific adaption to a single device provide a reasonable attack detection probability. However, such systems need to be installed on each security critical device and thus also affect its performance. They also cannot be hidden from an adversary. Figure 5.10a shows the network topology of a host-based IDS.

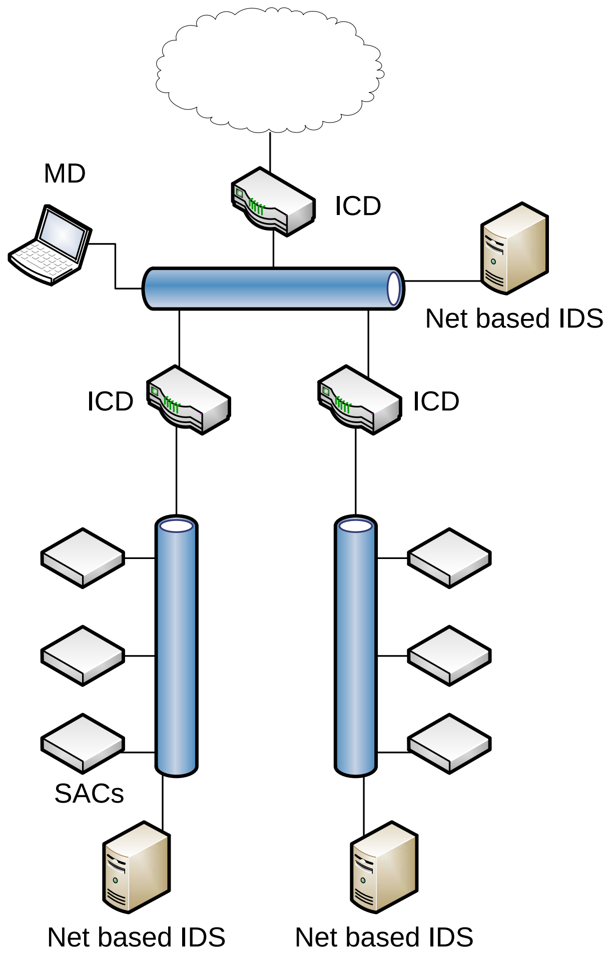
A network-based IDS observes the complete network traffic or the traffic of the network segment it is connected to and is executed on a dedicated device. Therefore, these systems are able to discover anomalies which affect more than a single host. Besides, they have the benefit that they do not affect the performance of the other devices in the network and can be installed without being accessible to a possible adversary. Since high network traffic may be present (especially in the backbone network), network-based IDSs need to be powerful. Figure 5.10b shows the network topology of a network-based IDS.

In a hybrid IDS the architecture consists of distributed host-based and network-based IDSs and a management system (cf. Figure 5.11). The latter is used to collect the information of the entire network, alarm and log in case of a security attack. BASs are typically large networks and monitoring single network nodes is not sufficient. Since a hybrid solution combines the benefits of both types it can be expected to be the most effective, but also most expensive IDS.

- The core unit of an IDS is the data processing component. It processes the collected data and determines whether an attack is present or not. Again, different approaches exist. According to [107], the two most important ones are called misuse/signature based and anomaly based. Misuse/signature based systems (SIDS)



(a) Host-based



(b) Network-based

Figure 5.10: Data Gathering Component

use a-priori knowledge of activities that form an attack. This knowledge (i.e. specific misuse patterns based on a security threat analysis) is stored in a database which contains typical patterns of known attacks (signatures). To determine whether an observation can be classified as an attack, different techniques such as expert systems as well as signature detection mechanisms can be used.

An AIDS tries to detect abnormal behavior by comparing the observed behavior with the normal and expected behavior (also called reference pattern). To achieve such a comparison, an application model must be specified. This model must define the default reference pattern (i.e. network traffic, device behavior) which represents the expected and normal behavior of the system. Obviously, this default behavior is not static since it can change during the lifetime of the system. Therefore, self-

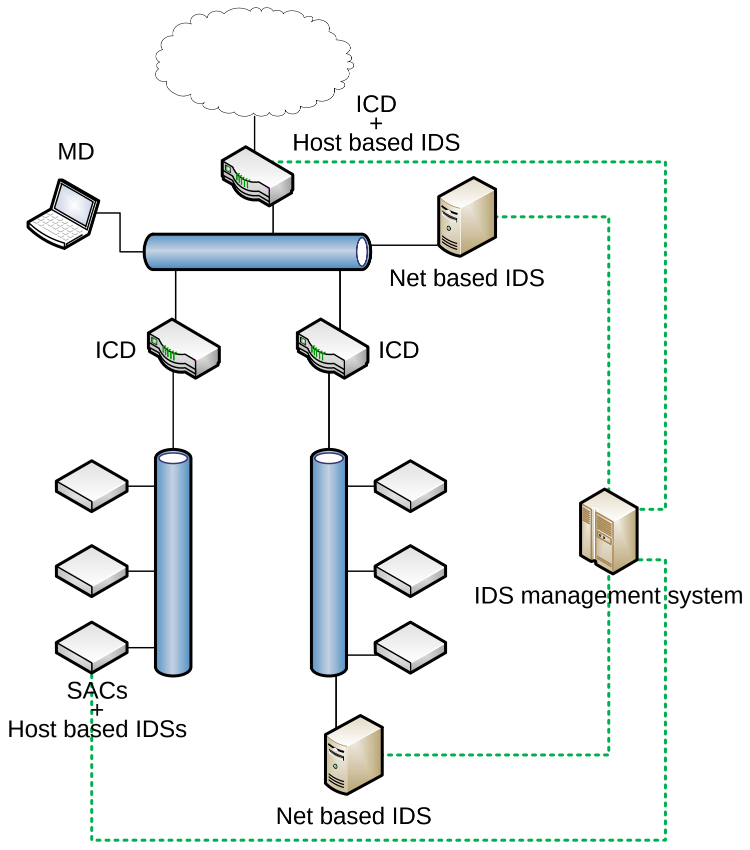


Figure 5.11: Data Gathering Component: Hybrid Approach

learning techniques (e.g. neural networks) are used to provide the opportunity to adapt the reference patterns during runtime.

Again, a hybrid solution which combines the benefits of both approaches seems to be the most practical one for BAS.

- Collecting the results as well as the observed data (communication traces) is the task of the data storage unit.
- The response component, on the other hand, is responsible for initiating actions to minimize the consequences of a detected security attack. This can be done by performing a direct feedback to the BAN. For example, it could decouple the affected control network segment.

The resulting IDS model for BAS is shown in Figure 5.12.

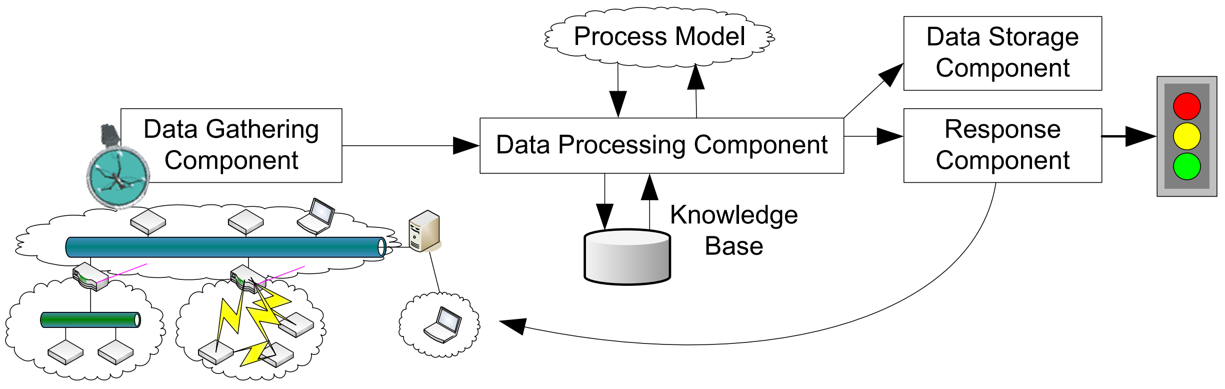


Figure 5.12: Intrusion Detection in Building Automation Systems

5.5 Summary

To summarize this section, the following hybrid software protection mechanisms are deployed to provide security of the architecture:

- SCA, IAC and code reviews are performed on the system software, which provides an abstraction and layering to ease CA development.
- An AIDS based upon a security policy and a generic application model is deployed on process control data and system call level to prevent attacks. An IDS is deployed to detect attacks.
- Uncircumventable sandboxing of CAs is performed to protect the system outside of the SB from attacks. DRM could also be deployed to only execute signed CAs, if desired.

6 Implementation and Evaluation

Hypothesis 6 *The developed secure CA architecture can be implemented on the devices typically found in BAS. It fulfills the requirements for secure CAs and is able to prevent or at least detect attacks.*

As introduced in Section 3.4, the typical security requirements for BAS devices are identification (i.e. user validation), resource access (i.e. network and I/O access only if the device is authorized), communication (i.e. authentication, confidentiality, integrity, freshness), storage of sensitive information (i.e. confidentiality and integrity), content security (i.e. usage restrictions of digital content), and availability (i.e. performing intended functions) [37]. Thus, a sophisticated security architecture as presented in Section 5 needs to provide defense in depth considering the challenging constraints on these systems and to provide secure communication, intrusion detection, a secure software execution environment as well as physical security.

To validate the feasibility of the presented architecture, this section uses the following methods:

- On the basis of use case $\otimes$, prototypes have been implemented for the different device classes to evaluate and test the stability with respect to memory consumption, performance, and security. A test environment (cf. Figure 6.1) has been established for SACs, which interact directly with the physical environment, ICDs which link different networks and network segments, and MDs which are used to configure, maintain and diagnose a BAS. It consists of a KNXnet/IP based backbone with an attached desktop computer running the Engineering Tool Software (ETS) and a network based IDS to provide additional security. Using secure ICDs with firewalls, two KNX lines are connected to the backbone. A secure SAC switch and a standard KNX SAC are located on these different lines. A secure SAC actuator has been implemented as multi-protocol device being connected to a KNX line and a BACnet/IP network. A desktop computer running the Visual Test Shell (VTS) attached for monitoring as well as sending and receiving BACnet service requests and a secure MD are likewise connected to the BACnet/IP network.

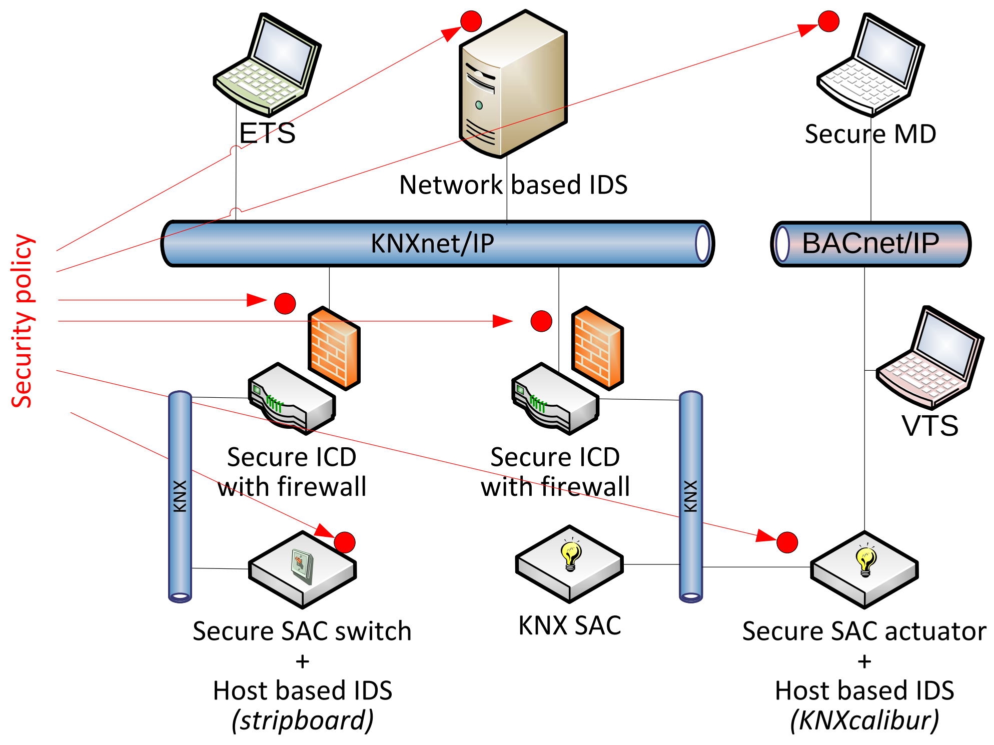


Figure 6.1: Test Environment and Proof-of-Concept Implementation of Use Case ⊗

- To further evaluate the presented concept, a discussion on how it can be used to enable security in today's already existing BASs follows. Attack detection and prevention become possible, if appropriate devices are installed.
- Concluding, an exhaustive discussion evaluates, that all functional requirements are fulfilled by means of the concept. Some organizational requirements still need to be considered, when implementing real live installations.

This section is structured as follows: First, the process of how to implement the secure architecture is described in detail: Section 6.1.1 discusses a common security policy for the prototypes, Section 6.1.2 describes the required steps to implement the CA fulfilling the use case, Section 6.1.3 addresses the configuration of a BAS, and Section 6.1.4 describes how device configurations can be derived out of the previously mentioned components. The following sections describe these prototypes – including but not strictly limited to the implementation of use case ⊗ – in more detail. Section 6.2.1 introduces two secure SAC prototypes, Section 6.2.2 describes an implementation of a secure ICD with a firewall, Section 6.2.3 demonstrates features of a network based IDS, and Section 6.2.4 discusses a secure MD. Performance measurements and an accompanying discussion on how the security process is working demonstrate, that the generic concept is applicable to today's hardware. Then, Section 6.3 demonstrates, that the proposed architecture can be used to provide security for already existing technologies and installations. Finally, Section 6.4 discusses the benefits and disadvantages of the proposed architecture.

6.1 Security Process

Section 5 provides a concept for secure and distributed CAs in BAS. A generic application model based on VDI 3813 and a mapping on today's technologies, a semi-formal framework for the definition of security policies as well as a system architecture and building blocks for implementation of secure devices have been described. Standardization committees can thus be supported in standardizing secure CAs and in defining generic security policies (cf. left side of Figure 6.2). By means of use case ⊗, this section describes the process of how to implement secure CAs (cf. right side of Figure 6.2). It is shown, how the generic architecture can be instantiated, i.e. how the CA can be modeled and how the policies can be adapted. Prototype implementations demonstrate, how to generate configurations for the devices, how commissioning can be performed, and which parts of the architecture guarantee the compliance with the security policy.

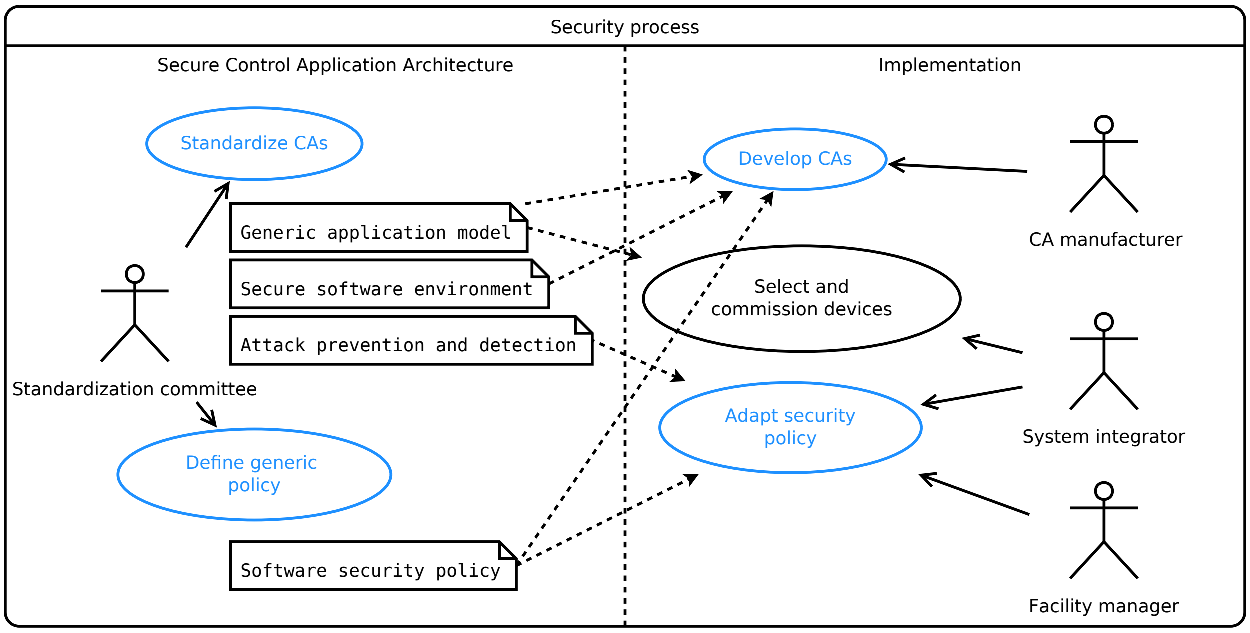

Security process
Secure Control Application Architecture
Implementation
Standardize CAs
Generic application model
Secure software environment
Attack prevention and detection
Standardization committee
Define generic policy
Software security policy
Develop CAs
Select and commission devices
Adapt security policy
CA manufacturer
System integrator
Facility manager

Figure 6.2: Security Process

6.1.1 Security Policy

The generic application model as defined in Section 5.1 contains all DPs which need to be considered by standardization committees for formulating a security policy. The idea of such a generic approach is, that a reasonable template policy with default values is established. This template policy is instantiated and adapted, when a BAS is being implemented. The implementation of use case ⊗ is given in Figure 5.6. The formal definition is found in D.7. This section describes the process of implementing a security policy on the basis of these proof-of-concept CAs.

1. The first step is to find the relevant DPs in the generic application model for the desired CA. As can be seen in Definition D.7, at least two DPs (MDP `PAR_OFFD` and PDP `L_MAN`) are required to implement a stairwell light.

2. The second step is to specify the security attributes. As seen in Definition D.8, a security attribute is always related to a present value p of a DP. For use case ⊗, the generic security attributes are listed in Definitions D.21–D.23. The instances are

formulated the following way:

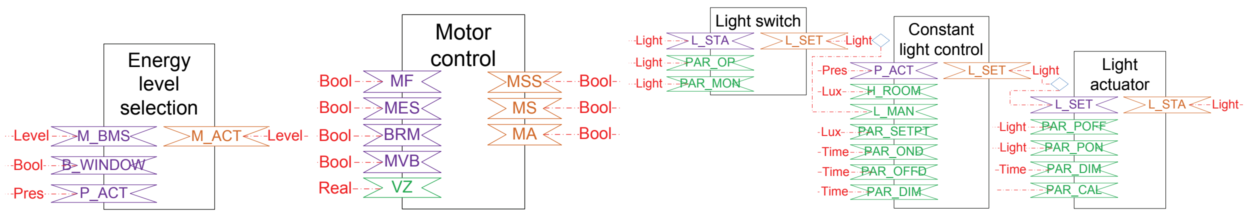
$$\begin{aligned}
s_{DP\ PAR\_OFFD} &: <p_{PAR\_OFFD}, cond(p_{PAR\_OFFD}, 600s, \leq), \\
&\quad cond(p_{PAR\_OFFD}, 60s, \geq), AND > \\
s_{SAC\ actuator} &: <systemcall\_switchoutput, \\
&\quad check\_accesstime(systemcall\_switchoutput, 1Hz, \geq) > \\
s_{DOMAIN\ Lighting} &: <p_{L\_MAN}, cond(p_{L\_MAN}, \{0,1\}, =) >
\end{aligned} \tag{P.1}$$

3. The next step is to associate the DPs to the SACs implementing the FBs as well as ICDs and MDs being connected to the BAN. As seen in Definition D.7, the SAC switch and SAC actuator are implemented. Remember, a binding between SACs is established, if they reference the same DP. For the proof-of-concept, additionally two ICDs, a network based IDS and a MD are connected. Thus, the following security policy can be defined for the BAS, specifying which device has to evaluate which security attribute:

$$\begin{aligned}
\text{SAC } switch &: s_{DOMAIN\ Lighting} \\
\text{SAC } actuator &: s_{SAC\ actuator}, s_{DP\ PAR\_OFFD}, s_{DOMAIN\ Lighting} \\
\text{ICD} &: s_{DOMAIN\ Lighting} \\
\text{network based IDS} &: s_{DOMAIN\ Lighting} \\
\text{MD} &: s_{DP\ PAR\_OFFD}
\end{aligned} \tag{P.2}$$

The SAC switch only has access to p_{L_MAN}, therefore it can evaluate $s_{DOMAIN\ Lighting}$. The SAC actuator has access to all DPs, therefore can evaluate all security attributes. The ICDs have access to the control network, thus giving them the possibility to evaluate $s_{DOMAIN\ Lighting}$, if process data is exchanged via this network. The network based IDS is permanently connected to the backbone, giving it the possibility to evaluate $s_{DOMAIN\ Lighting}$, if process data is exchanged via the backbone. Since also the binding is known in this step, it is also possible to evaluate, which device is allowed to communicate with a partner once the concrete configurations (e.g. addresses) are known. The MD being connected to the backbone, is able to evaluate $s_{DP\ PAR\_OFFD}$, since it has access to configuration data.

In order to demonstrate the feasibility of the presented approach, some more case studies of how security attributes can be applied to enhance existing standards are given.



(a) VDI 3813-2 FB level (b) ISO 16484 FB level (c) VDI 3813-2 Domain level

Figure 6.3: Security Attributes

A security attribute at FB level can e.g. be applied to VDI3813-2, *FB 6.5.1 energy level selection*, whose purpose is to adapt the energy transfer to the utilization of a room. The mapping of this functionality to a FB of the application model is shown in Figure 6.3a. A room is not constantly being kept on comfort temperature level, but the setpoints vary depending on external constraints. For instance, the output energy level p_{M_ACT} depends on the boolean input p_{B_WINDOW}. If the window is open, then p_{M_ACT} has to be at protection level so that no damage to the installation or the basic structure of a building occurs, otherwise internal control functions are executed and p_{M_ACT} may be set to other levels. The corresponding security attribute can be formulated as:

$$s_{vdi3813\_651\_p_{M\_ACT}} :< p_{M\_ACT}, cond_{p_{M\_ACT}}, cond_{p_{B\_WINDOW}}, AND > \tag{P.3}$$

$$cond_{p_{M\_ACT}} : cond(p_{M\_ACT}, protection\_level, =) \tag{P.4}$$

$$cond_{p_{B\_WINDOW}} : cond(p_{B\_WINDOW}, true, =) \tag{P.5}$$

and can be read as: If the window is open (P.5), then p_{M_ACT} needs to be at protection level (P.4). Otherwise the security attribute is violated.

Another example for a FB security attribute is ISO 16484-3, *5.5.3.3.3 motor control* (cf. Figure 6.3b), where its output p_{MSS} must stop the motor if a failure is reported. Thus, the input motor electrical failure p_{MES} has a higher priority than the input motor enable p_{MF} and under all error conditions overrides it. The corresponding security attribute can be formulated as:

$$s_{iso16484\_55333\_p_{MSS}} :< p_{MSS}, cond_{p_{MES}} > \tag{P.6}$$

$$cond_{p_{MES}} : cond(p_{MES}, true, =) \tag{P.7}$$

which states, that irrelevant of the input parameter p_{MF} the motor shall be turned off, if an error is detected (P.7).

Finally, an example of a security attribute for the lighting control domain is the interaction of the VDI3813-2 *5.2 light switch*, *6.3.5 constant light control* and *4.2.1 light actuator* FBs (cf. Figure 6.3c). The light switch can be used as input for a central off function. Its output is connected to p_{L_MAN} of the constant light control and forces its p_{L_SET} output to switch off. All light actuators connected to p_{L_SET} should then switch their corresponding outputs off. The security attribute:

$$s_{vdi3813\_lighting\_p_{L\_STA}} :< p_{L\_STA}, cond_{p_{L\_STA}} > \tag{P.8}$$

$$cond_{p_{L\_STA}} : cond(p_{L\_STA}, p_{L\_SET}, =) \tag{P.9}$$

models exactly this relationship between the p_{L_SET} output of the light switch and all connected p_{L_STA} outputs of the light actuators (P.9).

6.1.2 Control Application Development

When developing a CA, a manufacturer has to perform the following steps:

- First, it has to decide which FBs of the generic application model to implement and which not. Mandatory DPs need to be implemented and optional DPs need to be considered when appropriate. Figure 6.4 shows this development step for the secure SAC actuator. The functionality given by the FB `light control` and the FB `light actuator` will be implemented within a single CA `light actuator`. Listing 6.1 shows how the generic application model (i.e. the FBs) is referenced.

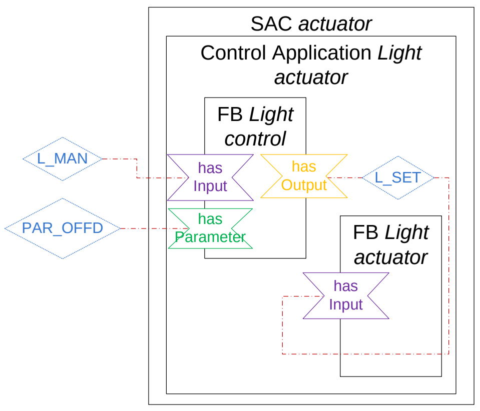


Figure 6.4: Control Application Development for Secure SAC Actuator

```
<?xml version="1.0" encoding="UTF-8"?>
<SAC_actuator xmlns:xsi="http://<!-- [...] -->">
  <!-- implement function blocks -->
  <FunctionBlock idref="3813-2_6_4_2">
    <!-- implement light actuator FB -->
    <ProcessDataPoint idref="L_SET" />
  </FunctionBlock>
  <FunctionBlock idref="3813-2_6_5-6">
    <!-- implement light control FB -->
    <ProcessDataPoint idref="L_SET" />
    <ProcessDataPoint idref="L_MAN" />
    <ManagementDataPoint idref="PAR_OFFD" />
  </FunctionBlock>
<!-- [...] -->
```

Listing 6.1: Control Application Function Blocks

- Second, the underlying BAS technologies need to be chosen. Using the generic application model, it is possible to provide an automatic mapping of generic DPs to technology dependent ones.

- Finally, the building blocks of the secure software environment (i.e. the system software, the enhanced application layer and the SB) need to be realized. It depends on the manufacturers and used technology, whether existing solutions are chosen or the manufacturers implement these blocks on their own.

6.1.3 Building Automation Network Configuration

After a building has been planned, the system integrator is responsible for the realization of the desired functionality and fulfillment of the requirements. It selects suitable devices and CAs and plans the BAN topology. This involves deciding which FBs will be realized by which devices and how these devices are being connected (i.e. binding). This also involves an adaption of the security policy with respect to building owner's requirements. Figure 6.5 shows the configuration (green text) for the test environment and proof-of-concept implementation of use case ⊗.

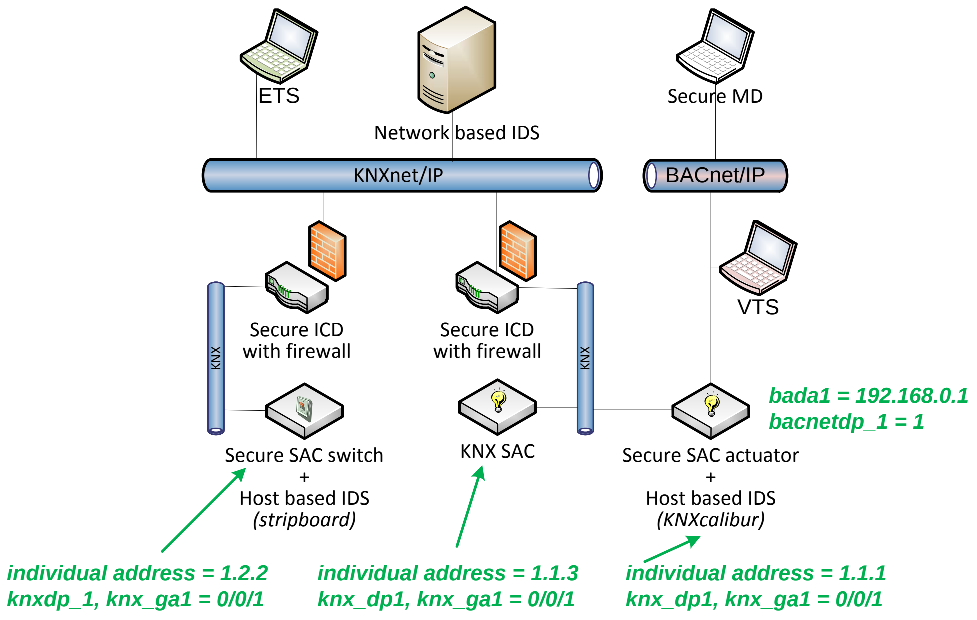


Figure 6.5: Configuration

6.1.4 Device Commissioning

The final step when implementing a BAS is the commissioning of the devices. Using the security policy, the configuration and the CAs, it is possible for the system integrator to derive the device configuration and security policy for each node. Figure 6.6 shows this process for the five prototypes of use case ⊗.

6.2 Prototypes

The introduced device classes obviously need different hard- and software due to the discussed requirements. Nevertheless they may share the same components if appropriate. A modular architecture consists of different, exchangeable hardware blocks which are

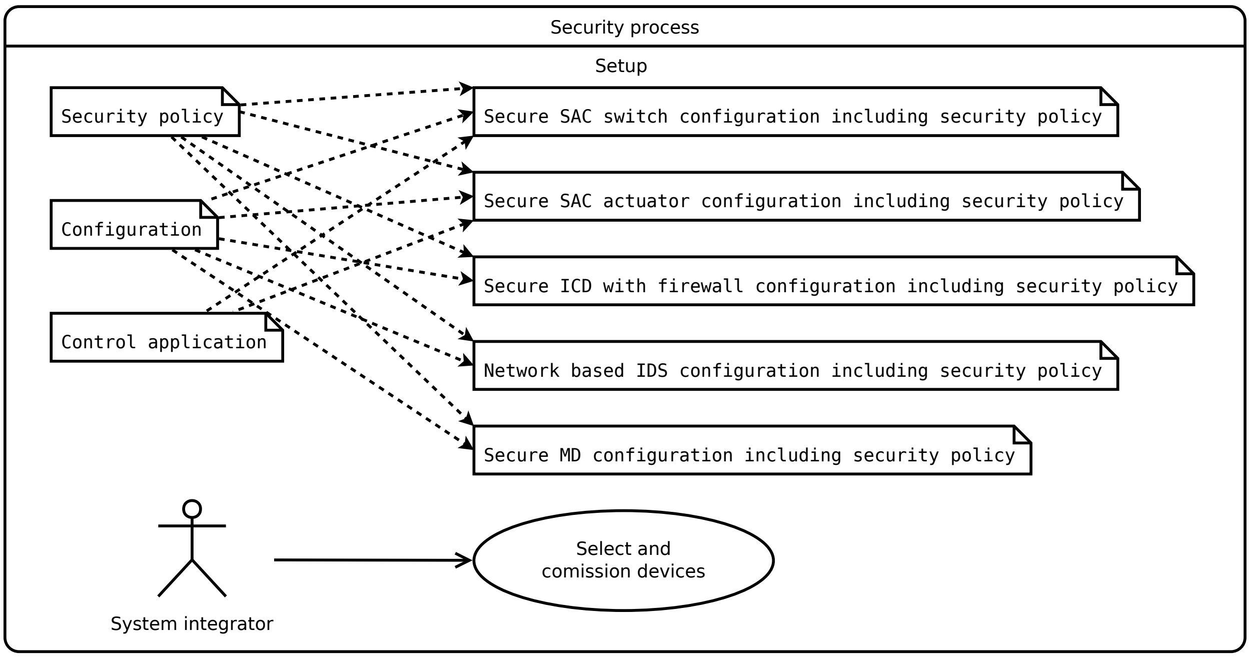

Security process
Setup
Security policy
Configuration
Control application
Secure SAC switch configuration including security policy
Secure SAC actuator configuration including security policy
Secure ICD with firewall configuration including security policy
Network based IDS configuration including security policy
Secure MD configuration including security policy
System integrator
Select and comission devices

Figure 6.6: Commissioning

able to fulfill embedded control, interconnection as well as configuration and management tasks. The resulting devices/platforms are:

- Universally applicable to serve as a basis for further work in the scope of building automation (e.g. test platform for protocol extensions)
- Compact, embedded, robust and low cost
- Flexible (i.e. configurable in hardware by use of jumpers and software, which is especially useful for SACs and ICDs) and extensible
- Easy to use
- Powerful, meaning that enough processing power and memory shall be available to implement representatives up to MDs

At the left side of Figure 6.7, a mapping of the general hardware model (cf. Figure 2.2) to a modular architecture is shown. At the right side, possible implementations derived from this architecture are displayed.

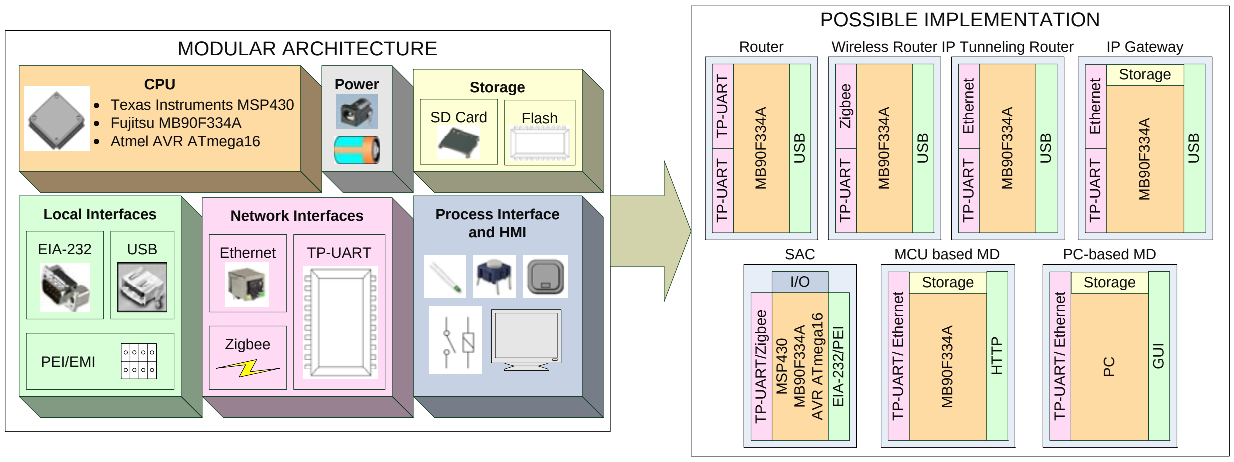


Figure 6.7: Hardware Architecture for Proof-of-Concept

6.2.1 Sensor, Actuator and Controller

This section presents an implementation of secure SAC prototypes, which allow untrusted, possible intentional malicious software to be executed securely on those different low end ESs. Application designers are supported to allow rapid innovation and implementation, configuration, deployment and execution of arbitrary CAs. The outcome is a secure and customizable CA environment targeted to work with low end SACs featuring MCUs of limited memory (<100 KBytes in total) and tight processing power (<25 MHz).

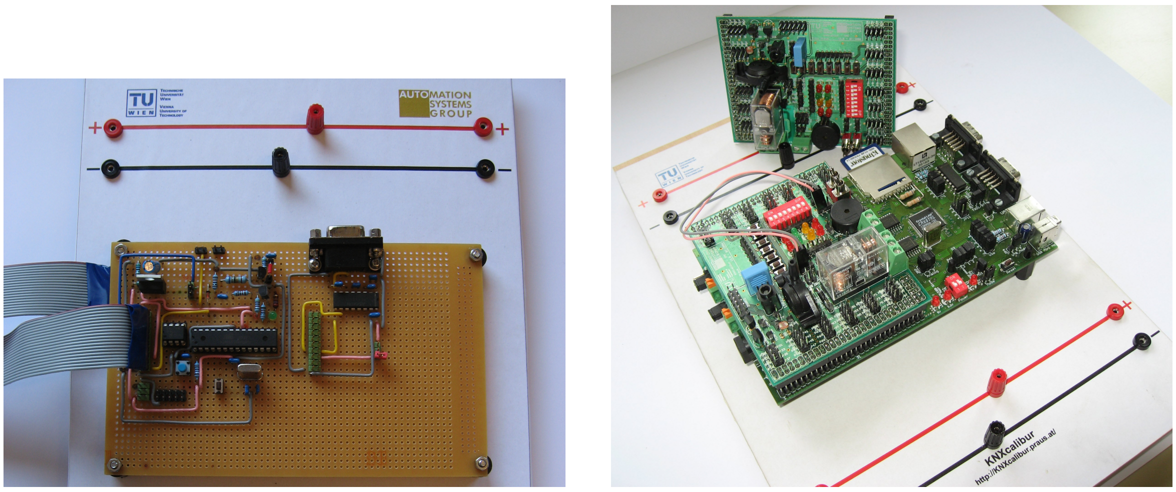
Based on the identified demands, modular hardware platforms have been designed, which serve as a basis for not only the development of software for SACs but also can be used for e.g. ICDs. Several aspects have to be considered for the SAC hardware design. Although application tasks are not complex, enough resources have to be available to handle security tasks such as encryption. Hardware support for CA isolation has to be considered. Power consumption is of major concern since SACs are often supplied via link power to avoid an additional power cable. In special cases, they may be even driven by a battery (e.g. glass break sensors). Finally, SACs have to be reasonably robust when placed in the field. All these requirements have to be achieved in a cost efficient way since the number of SACs in a BAS is high. In addition, a modular software design has been implemented. Based on a HAL, hardware independent modular software components have been implemented (e.g. system software, enhanced application layer, SB).

With the first proof-of-concept implementation `Secure SAC switch` (cf. Figure 6.1) it is possible to control a KNX compliant light `KNX SAC` as well as the second `Secure SAC actuator`. The latter implements use case ⊗ and has been designed as an enhanced multi-protocol device being integrated into a BACnet and KNX network. Communication between sensor and the enhanced actuator has been secured using a subset of [61].

6.2.1.1 Hardware Architecture

The presented concept has been deployed on two different hardware architectures, each with its own use cases.

First, a SAC stripboard [129] (cf. Figure 6.8a) intended to be powerful enough to handle the presented SB approach and control a KNX compliant light (on, off) has been assembled, with the main goal to verify the approach regarding the imposed overhead and gain performance results. Moreover, it is intended to provide a low level BAN access in order to use it as test device for security analyses. Offering bit level access to KNX using dedicated hardware, security attacks can be performed and results be observed. The SAC stripboard stays very small scale and represents the lowest end SAC devices. While it appeared reliable during development and testing of the software running on it, it is not intended for real-life use. Hardware assisted methods in mind, an Atmel ATMega168 MCU (implementing the HA) clocked at 8 MHz, featuring 16 KBytes flash, 1 KByte SRAM, 512 Bytes EEPROM was chosen. Among others, the controller provides 23 programmable I/O lines, two 8-bit Timer/Counters and one 16-bit Timer/Counter with various compare modes, a programmable serial USART, a byte-oriented 2-wire serial interface and two external interrupts. Some external components have been equipped to provide additional functionality: For debugging, testing and basic interfacing, a LED and a pushbutton are attached. An additional external EEPROM (MICROCHIP 24LC16B/P) with a memory size of 2048 Bytes is used to provide extra non-volatile memory. It is connected via the 2-wire serial interface supported by the ATMega168, which is actually an I^2C connection (the naming is different due to licensing reasons). The connection is clocked at 400 kHz, which is the fastest possible with the external EEPROM. It still limits the data rate significantly, since there is some overhead contained in the required data exchange protocol. A MAXIM MAX 3232CPE dual channel driver/receiver chip is equipped to convert controller signals to Electronic Industries Association standard RS-232-C (EIA232) levels to enable serial communication, for example with a PC. On the side of the MCU, it is directly connected to the USART pins of the ATMega168. To access the KNX bus, a slightly adapted version of the basic circuit of the Freebus project[1] has been considered to convert the MCU signals to KNX signals and to power the device using the bus line.



(a) SAC stripboard (b) KNXcalibur

Figure 6.8: Sensor, Actuator and Controller Prototypes

Second, an integrated, compact and powerful platform named KNXcalibur was built [128], [135] (cf. Figure 6.8b). It is applicable as SAC, but can also be used as ICD and MCU based MD with an integrated web server in mind. KNXcalibur is based on a Fujitsu 16 bit MB90330 family MCU. It operates at a maximum frequency of 24 MHz and provides 24 Kilobytes RAM, 384 KB flash, 4 UARTs, a 8/10 Bit A/D converter and a Serial Peripheral Interface as well as an external bus interface similar to the ISA bus. Moreover, USB functionality and Ethernet connectivity via the Cirrus Logic CS8900A Ethernet LAN controller are realized. To support persistent storage of large amounts of data on KNXcalibur, a SD/MMC card connection has been integrated. Connection to KNX is realized with Siemens TP-UART ICs. Via the available pin headers an extension daughter board has been connected. It provides 8 push buttons, 8 LEDs, 8 switches, 1 relay, 1 buzzer

[1] `http://www.freebus.org`, Last access: 2015/19/08

and a MAX3471 for EIA-485 connectivity. In addition the following sensors have been integrated: an HSP15 humidity sensor, a MPX4250 pressure sensor, a NSL19M51 light dependent resistor, an LM335Z analog temperature sensor, a DS1820 digital temperature sensor and a DCF77 connector for attaching a FSZ01020 antenna.

Using pin headers, all devices can be connected to "header boards" with modular components (power supply, point-to-point interface, process interface, network interface). Simple buttons and LEDs are used as process interfaces. Moreover, a PEI connector, which offers digital and analog access to external sensors and actuators, is present.

6.2.1.2 Software Architecture

To allow flexibility and different hardware configurations, the software has been implemented modular, based on the generic architecture (cf. Figure 5.9). With limited hardware resources of SACs in mind, an embedded VM forms a SB and provides a lean environment for Java based CAs. Interfacing to the BAN (e.g. KNX or BACnet) and the specific hardware is done by a well-defined Java API. In addition, a management API allows to download and configure CAs inside the VM. The programming concept, configuration and management issues as well as the workflow using the open development platform Eclipse are described. Figure 6.9b shows the software implementation structure.

The main basis for hardware independent software modules forms the HAL, which has been implemented in an event based way.

Using the HAL it is possible to implement generic modules in the system software, such as the KNX network plugin, which provides bit stuffing support for controlling the Freebus hardware as well as a TP-UART driver for more simplified KNX network access.

Independent of the underlying OSI layer 1/2, the KNX stack offers KNX group communication as well as group object handling (`A_GroupValue_Read`, `A_Group-Value_Write`, `A_GroupValue_Response`) and management communication (`A_Prop-ertyValue_ Read`, `A_PropertyValue_Write`, `A_PropertyValue_ Response`). In a first prototype, it has been configured to keep a standard KNX light actuator `FB_Light_Actuator` with associated group address synchronized with the generic application object `On/Off Light`. This way, runtime compliance to the KNX standard can be demonstrated. In a second prototype, it is used for implementing use case ⊗ and to show the implementation of a security policy.

Likewise the BACnet plugin provides access to BACnet/IP. It maps the `On/Off Light` to the BACnet `Lighting-Output` and provides the services `ReadProperty` and `WriteProperty` to manipulate its properties (e.g. `PresentValue`).

Besides, drivers for the sensors mentioned before are contained in the system software within further software modules. The system software also manages and stores configuration parameters and data of the enhanced application layer in non-volatile memory.

The user API forms the main component for the CA developer and is the only way to interact with the environment. Native Java classes have been defined which form the interface for the CA. The SB maps these Java methods and variables to their C implementations, which then access the application objects. The KNX plugin hides KNX specific details and provides a high-level access to group objects. Communication is solely performed by reading and writing these objects. Any messaging related tasks such as frame and checksum generation and checking, handling of acknowledgments and retransmission in case of errors are carried out by the library. Likewise, the BACnet plugin handles BACnet communication. The I/O plugin handles access to peripherals and is, as mentioned, common to every hardware configuration. Listing 6.2 shows an extract of how CAs can access a generic (networked) application object or a peripheral.

```java
class FunctionBlock {
  public ProcessDataPoint readValue(String id); // Reads and returns the
      current value of the application object from the memory.
  public void WriteValue(ProcessDataPoint value); // Writes the value of the
      application object. Automatically triggers network plugins to update
      their values.
  [...]
}
class IO {
  public static int out(int pin, int value); // 1 turns on pin, 0 off

  public static int initDCF77(void); // initialize DCF77
  public static Date getTime(); // returns current date and time

  public static int initTempDigital(void); // initialize digital temperature
      sensor
  public static Float getTempDigital(void); // returns temperature value

  public static int initPressure(void); // initialize pressure sensor
  public static Float getPressure(void); // returns pressure value

  public static int initLDR(void); // initialize LDR
  public static Integer getBrightness(); // returns brightness value

  public static int initHumidity(void); // initialize humidity sensor
  public static Integer getHumidity(); // returns humidity value
[...]
```

Listing 6.2: Extract of User Application Programming Interface

Configuration of a node is XML based and, as described, can be generated using e.g. a configuration tool. Using the generic application model the required mappings of the specific addresses to the generic DPs can be calculated. Likewise, the corresponding security attributes for the plugins can be generated out of the generic application object. The desired generic FBs and DPs are instantiated using their `id` attributes, and their parameters are configured. Then a configuration of the network plugins can be calculated. Basically, this means binding the DPs to addresses. Security policy restrictions for the different APIs can also be derived. Besides, relevant parameters for further system components (e.g. mapping a MCU output pin to the corresponding light) can be set.

Finally, this representation is transferred into a suitable binary format and stored in the EEPROM of the target node using the management API. Listing 6.3 shows the configuration of the SAC actuator of use case ⊗. The security policy is derived from P.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<DeviceConfig name="SAC actuator" xmlns:devconf="http://[...]">
  <Generic>
    <!-- instantiate generic function blocks -->
    <FunctionBlock idref="3813-2_6_5-6">
      <!-- set security attribute s_DOMAIN Lighting -->
      <ProcessDataPoint idref="L_MAN" value="0,1" />
      <!-- set MDP to 120 and security attribute s_DP PAR_OFFD-->
      <ManagementDataPoint idref="PAR_OFFD" largerthan="60" smallerthan="600">
          120</ManagementDataPoint>
    </FunctionBlock>
  </Generic>

  <IO> <!-- configuration and restrictions for I/O library -->
    <!-- set security attribute s_SAC actuator-->
    <Systemcall id="out" maximumfrequency="1">
  </IO>

  <KNX> <!-- configuration for KNX plugin -->
    <!-- KNX physical address -->
    <IndividualAddress>1.1.1</IndividualAddress>
    <!-- define group address -->
    <GroupAddress id="knx_ga1">0/0/1</GroupAddress>
    <!-- bind group address to datapoint -->
    <DP idref="knxdp_1" garef="knx_ga1" />
  </KNX>

  <BACnet> <!-- configuration for BACnet plugin -->
    <!-- define device address -->
    <BACnetDeviceAddress id="bada1">192.168.0.1</BACnetDeviceAddress>
    <!-- define object identifier -->
    <Property idref="bacnetdp_1">1</Property>
  </BACnet>
<!-- [...] -->
```

Listing 6.3: Device Configuration of Sensor Actuator and Controller Actuator

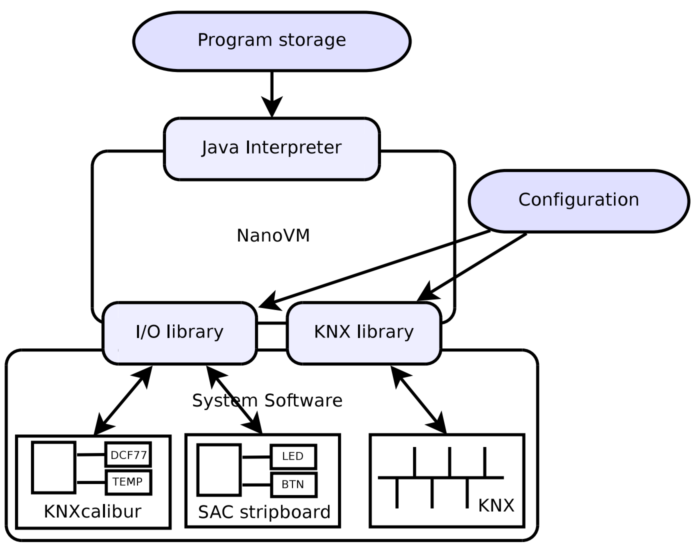
As shown in Section 4.4, a lot of different SBes are available. Due to the memory requirements (160-512 KBytes of total memory) and the requirement for a 16 Bit processor clocked at 16 MHz or higher the Java ME is slightly beyond the limitations of the targeted hardware. Besides, the official Java versions, there is a number of implementations available which are targeted at systems with very low resources. These implementations often come at the price of only offering a subset of the full Java functionality. Two suitable solutions will be discussed in the following: NanoVM and TinyVM. Both VM implementations allow the execution of standard Java bytecode, at least after preparation with an automated tool. They target very low-profile ESs. The NanoVM was originally written for the Atmel ATMega8 MCU included in the Asuro robot, and has a memory footprint of approximately 7 KBytes. Some of its features are 15/31 bit integer arithmetic, floating point operations, garbage collection, simple application upload, inheritance, a unified stack and heap architecture and about 20k Java opcodes per second on a 8 MHz AVR. The TinyVM operates on Lego Mindstorms RCX programmable bricks which are equipped with an Hitachi H8 MCU. It has memory requirements of around 10 KBytes. Both VMs have some limitations regarding the Java language features. For example, the NanoVM does not support multi-threading. The TinyVM is generally more advanced (of course also due to the more powerful hardware it runs on), and supports multi-threading, exceptions and synchronization but for example misses floating point operations. But despite these limitations both VMs have their possible application areas, since the omitted functionality is often not required for their intended use.

Finally, the NanoVM was chosen for sandboxing the CA, since a port to the AVR MCU already existed and only a port to the Fujitsu MCU had to be done. Nevertheless, considerable modifications have been applied to achieve the targeted level of functionality. First, the CA is not stored in the flash memory of the MCU but on external storage (external EEPROM in case of the stripboard, SD card for KNXcalibur). Thus, the NanoVM had to be modified to allow fetching the CA instructions from external program storage. In addition, Java libraries have been written to access the peripherals (via the user

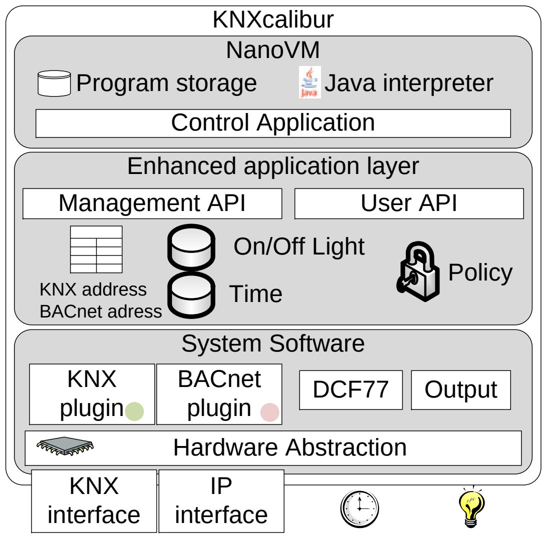
API) of the hardware platforms and to enable BAS specific methods (e.g. KNX compatible data exchange). The NanoVM has also been extended to provide a basic AIDS-like protection mechanism. During execution, all instructions are monitored and checked against the predefined security policy. This way, problematic behavior caused by malfunctioning or malicious software can be detected and the execution can be stopped. The policy is extracted out of the configuration file described before. Currently, this step is performed manually, however, a model-driven approach would allow automatic transformation [144]. Listing 6.4 briefly shows the implemented SB and policy monitoring.

```
// Number of policy rules
#define SEC_POLICY_RULES_CNT 1
// Macro to store rule in memory
#define COMPOSE_RULE(limit, time, action) ( (time & 0x3f) | ((limit & 0x3f)
    <<6) | ((action & 0x3)<<14))
// Macro to get limit per time
#define GET_RULE_LIMIT(rule) ((rule >> 6) & 0x3f)
// Macro to get time of last call
#define GET_RULE_TIME(rule) (rule & 0x3f)
// Macro to get action on policy violation
#define GET_RULE_ACTION(rule) ((rule >> 14) & 0x3)

// Compose rule to halt the CA if io.out is called more often than one time
    per second
policy_rule_t sec_policy_rules[SEC_POLICY_RULES_CNT]  =
{
  {NATIVE_CLASS_IO, NATIVE_METHOD_OUT, COMPOSE_RULE(1, 1, RULE_ACTION_HALT)}
};
// This method is called on every instruction of the CA
u08_t sec_check_invocation(u16_t mref) {
  // For all defined policies
  for (u08_t i=0; i<SEC_POLICY_RULES_CNT; i++) {
    // If policy rule is available for instruction
    if (NATIVE_ID2CLASS(mref) == sec_policy_rules[i].class_ref) &&
      (NATIVE_ID2METHOD(mref) == sec_policy_rules[i].method_ref) {
      // Count number of calls
      invocations[i].count += 1;
      // If rule violated, halt CA
      if (invocations[i].count >= GET_RULE_LIMIT(sec_policy_rules[i].rule)) {
        switch (GET_RULE_ACTION(sec_policy_rules[i].rule)) {
          case RULE_ACTION_HALT:
            error(ERROR_POLICY_VIOLATION);
            break;
[...]
// Timer interrupt
void TIMER1_COMPA_vect(void) {
  // For all defined policies
  for (u08_t i=0; i<SEC_POLICY_RULES_CNT; i++) {
    // Reset counters if rule not violated
    if (invocations[i].time >= GET_RULE_TIME(sec_policy_rules[i].rule)) {
      invocations[i].count = 0;
      invocations[i].time = 0;
[...]
// Invoke instruction of CA
void native_invoke(u16_t mref) {
  // Check if invokation is allowed by policy
  if (sec_check_invocation(mref) != 0) return;
  // Select native function and invoke it
  if(NATIVE_ID2CLASS(mref) == NATIVE_CLASS_PORT) {
    native_avr_port_invoke(NATIVE_ID2METHOD(mref));
```

Listing 6.4: Sandbox and Policy Implementation



(a) Components



(b) KNXcalibur

Figure 6.9: Software Implementation

Figure 6.9a outlines the flow of information between the software components. CA Java bytecode instructions are fetched from the program storage and are interpreted by the Java interpreter. The instructions may be operations on internal Java variables or calls to library methods. The library methods read possible configuration values, execute the designated functions on the hardware platform or communicate with peripherals.

In the following, the development workflow – starting from program development to final CA download – and the stairwell lighting CA are described. The system software itself has been compiled and downloaded using the MCU specific toolchain (e.g. avr-gcc and AVR Downloader/UploaDEr, avrdude). CAs may be developed and compiled using standard Java toolchains (e.g. Eclipse SDK and the OracleJDK) and utilizing the User API. XML device configuration files currently have to be written manually. The configuration tool (NanoVMTool) is being used to prepare the compiled class files, which includes stripping unnecessary and unsupported instructions from the binary as well as mapping native Java library calls to their corresponding implementations in C. Besides, the NanoVMTool has been extended to provide functionality for parsing the XML based configuration file and changing application parameters. Finally it is used to upload the CA into the SB via the serial interface.

Listing 6.5 shows the required CA to implement the SAC actuator (`Stairwell light`). A node configuration defining the required application and parameters is assumed (like in Listing 6.3).

```
import nanovm.UserAPI.*;
class StairwellLight {
  FunctionBlock fb1=new FunctionBlock("3813-2_6_5-6"); // configuration is
       automatically read using the id "3813-2_6_5-6"
  public static void main() {
    ProcessDataPoint pdp = new ProcessDataPoint(); // temporary pdp
    Date date_temp = new Date(); // temporary date
    while(true) {
      if (!pdp.equals(fb1.readValue("L_MAN"))) { // if pdp has changed
        pdp = fb1.readValue("L_MAN"); // read its value
        if (pdp.intValue() == 1) { // if L_MAN is set to on
          io.out(0,1); // immediately switch lamp 0 on
        }
        date_temp = new Date(); // store time since last change of L_MAN
      }
      if (Date().getTimeSeconds()-date.getTimeSeconds() > fb1.readValue("
          PAR_OFFD") { // if time longer than PAR_OFFD elapsed
          io.out(0,0); // switch lamp 0 off
[...]
```

Listing 6.5: Stairwell Light Control Application

The main application code is placed in an infinite loop. Here, first the FB object `fb1` is checked for changes, which could be present if a related BAS message has been received. If there are any messages the PDP value is read. If an `on` telegram has been received, the light is immediately switched on. If an `off` packet is received, the light is switched off after the `PAR_OFFD`.

6.2.1.3 Experimental Results

Investigations regarding memory consumption reveal that the implementation on the SAC stripboard results in about 8 KBytes code size thus leaving another 8 KBytes Flash memory for further extensions. The internal 512 Bytes and external 2 KBytes EEPROM remain free and are used to store the CA and configuration parameters. Analysis of the usage of the 1 KByte application SRAM has been aggravated due to the virtual heap architecture provided by the NanoVM. In the original version, it reserves 256 Bytes for the internal operation thus leaving 768 Bytes SRAM for execution of CAs. With the imple-

Matrix dimension	Native C	Java (int. EEPROM)	Java (ext. EEPROM)
2	31 μ s	13.2 ms	420 ms
4	440 μ s	20 ms	2.62 s
8	3.92 ms	595 ms	18.5 s
16	32.8 ms	4.64 s	/

Table 6.1: Runtime Matrix Multiplication

mented extensions, approximately 70 Bytes of extra memory are used and therefore the heap size was reduced to 668 Bytes. To get a first estimation of memory consumption of a typical CA, a prepared code involving switching of the LED and KNX communication several times has been executed. As expected, the remaining heap size was shrinking every cycle until it reached zero. Then the built-in garbage collector was executed, freeing all the unused heap memory. The freed heap size is considered to be the actual unused heap memory being available to CAs. This size was around 580 Bytes resulting in a heap use of around 90 Bytes which leaves enough room for CAs dedicated for SACs.

The performance overhead of the imposed security measures has been evaluated from several points of view. Regarding the simple CA being able to control a light, no subjective feeling of slowdown can be observed. In addition, a simple PID controller has been implemented. Again no subjective slowdown can be detected and temperature control works as expected. Since SACs typically do not perform more computationally expensive tasks and the used stripboard represents the lowest end of todays available hardware the approach can be considered working. To objectively estimate the performance impact of the SB approach the raw calculation capability of square matrix multiplications with different matrix dimensions has been measured. It has to be noted, that no optimizations have been performed. Implementations in C and Java, running in the NanoVM using either internal or external EEPROM as program storage, are compared (cf. Table 6.1). The measured durations of the native C-implementation are always three to four orders of a magnitude smaller than the corresponding Java versions using the external EEPROM. Memory access operations due to slow connection of this EEPROM have been identified as a major reason for slowdown. A replacement of the external EEPROM with a faster one seems reasonable. For performance analysis the CA has been moved to the internal EEPROM. As can be seen, this clearly improves the execution time. The applicability of the presented approach to a particular ES depends on the acceptable slowdown of the CA or any other imposed overhead (e.g. battery gap). If the performance is good enough to enable the ES to fulfill the required tasks, the SB approach clearly improves security. Further it has to be noted, that an implementation of complex calculations in native C is also possible (with lack of SB security) if the corresponding Java code is too slow. The additionally required encryption of process and management communication data obviously takes resources of the involved devices. To evaluate the performance of KNXcalibur, a freely available AES implementation [138] was ported to the prototype (without further optimization). A first performance analysis shows that the encryption of a single KNX telegram takes approximately 0.59 ms net processor time. Since the total transmission time[2] of such a message is 39 ms the additional processing time clearly is not a limiting factor. The prototype thus can be expected to be able to handle encrypted messages at the maximum theoretical data rate of the KNX network.

6.2.2 Interconnection Device

This section is dedicated to a secure ICD with firewall (cf. Figure 6.1). First, a generic firewall architecture is presented, which is then implemented for a KNX based BAS [51]. The developed prototype is able to prevent attacks by physically separating two or more different BANs. Process data exchange on the network interfaces is being analyzed and depending on the deployed security policy data telegrams are filtered.

On the one hand, it is possible to check the communication relationship between CAs– i.e. which device is allowed to talk to which other device. The required information is gained out of the binding information, which has been generated when linking DPs. On the other hand, the data being exchanged – i.e. the present values – are monitored and checked against the security attributes gained out of P.2.

The left side of Figure 6.10 shows the typical filter rules and filter chains of a firewall, which are used to implement a security policy. Using the rule `DENY`, telegrams are silently dropped by the firewall. Neither sender nor receiver are informed, that delivery of telegrams has been filtered. Using the rule `REJECT`, telegrams are also dropped with

[2] This covers the whole transmission cycle including immediate acknowledgment and required bus idle times.

the difference that the sender is informed. The rule `ALLOW` permits telegrams to pass the firewall. Filter chains accumulate filter rules, which are evaluated step by step. The filter chain `INPUT` is used for filtering process data exchange targeting the application layer of the firewall. The filter chain `OUTPUT` is applied, when the application layer of the firewall transmits a telegram. In fact, the use of the `INPUT` and `OUTPUT` chain is only feasible, if a firewall is implemented directly on a SAC. If an ICD without CA is deployed, only the `FORWARD` chain is required. It is responsible for filtering telegrams being transmitted from one interface to the other. Two approaches are possible, when implementing a firewall policy. The first approach, being more secure but also harder to configure and maintain, is to `DENY` or `REJECT` all telegrams per default and to provide `ALLOW` rules for each telegram, that should be allowed to pass the firewall. Second, the default rule is set to `ALLOW` and `DENY` or `REJECT` rules are specified for specific telegrams that need to be blocked.

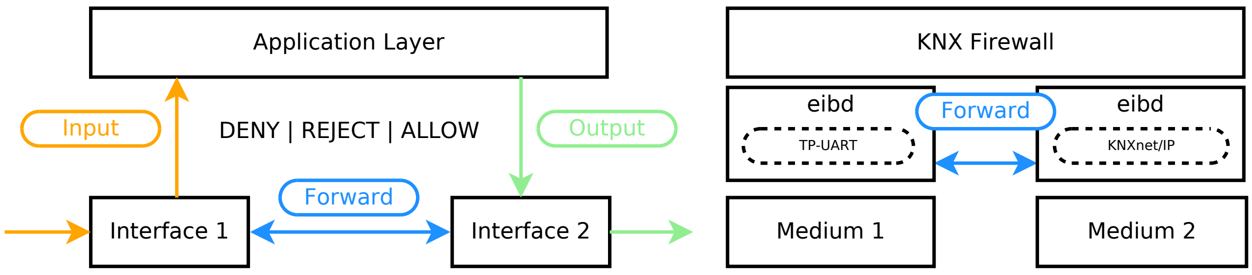


Figure 6.10: Secure Interconnection Device and Implementation of KNX Firewall

To validate the approach, a KNX based prototype ICD has been implemented for use case ⊗ (cf. right side of Figure 6.10). It provides connection to two KNX lines using two instances of the eibd[3]. The first interface is connected to TP-1 using a TP-UART connection, whereas the second interface is connected to an IP backbone using a KNXnet/IP router. A `FORWARD` chain between those lines is configured, no `INPUT` or `OUTPUT` chain is defined. To generate firewall rules, the following KNX characteristics need to be considered, when formulating `ALLOW`, `DENY` or `REJECT` rules. The communication relationships between CAs can easily be gained out of the installation configuration being available in the ETS. Physical destination addresses are only used for management communication

[3] `https://www.auto.tuwien.ac.at/~mkoegler/index.php/eibd`, Last access: 2015/09/01

to e.g. (re-) configure devices, read mask version or memory areas. A `REJECT` rule generates a `NACK`, and an `ALLOW` rule generates an `IACK` on TP-1 lines. The Application Protocol Data Unit (APDU) type can be used to further filter process data exchange. The `A_GroupValue_Write`, for instance, is used to set a PDP.

The following security policy can thus be formulated (cf. Listing 6.6). The syntax is similar to the one used in the standard Linux firewall:

- Drop physically addressed telegrams (cf. Line 3).
- Prohibit process data exchange between unknown CAs (cf. Line 3).
- Evaluate the security attribute $s_{DOMAIN\ Lighting}$ (cf. P.1 and Line 16):

  `-a` append a new rule to chain `FORWARD`

  `-s` source address is always a physical address

  `-d` destination address can be physical or group address

  `-i` in-interface

  `-o` out-interface

  `-p` APDU type and value

  `-j` action being applied on rule match

```
 1 [global]
 2 # deny all telegrams (including physical addressing)
 3 standardFirewallRule=DENY
 4 
 5 # configuration for first interface
 6 [KNX-IF-1]
 7 connectionurl=ip://auto.tuwien.ac.at:6720
 8 
 9 # configuration for second interface
10 [KNX-IF-2]
11 connectionurl=ip://auto.tuwien.ac.at:6721
12 
13 # rules
14 [rules]
15 # allow telgrams from secure SAC switch originating from KNX-IF-1 to KNX-IF-2
       with target secure SAC actuator, present value must be 0 or 1
16 -a FORWARD -s 1.2.2 -d 0/0/1 -i KNX-IF-1 -o KNX-IF-2 -p A_GroupValue_Write
       ={0,1} -j ALLOW
```

Listing 6.6: Configuration of Secure Interconnection Device with Firewall for Use Case ⊗

Implementation of the policy check can be realized using simple `if` statements (cf. Listing 6.7).

```
// Check every received telegram
static gboolean CheckReceivedTelegram(struct FirewallRuleChain *chain,
    eibaddr_t source, eibaddr_t destination, gchar in_interface, gchar
    out_interface, APDU *data) {
  // Check source address
  if (CheckSourceAddress(chain->source_from, source)) {
    // Check destination address
    if (CheckDestinationAddress(chain->destination_to, destination)) {
      // Check in and out interface
      if ((chain->in_interface & in_interface) && (chain->out_interface &
          out_interface)) {
        // Check APDU type and value
        if (CheckAPDU(chain->data, data) {
          // Rule matched, apply jump
          return (chain->jump);
[...]
```

Listing 6.7: Implementation of Firewall

The secure ICD with firewall has been implemented on an embedded Linux platform. Regarding performance, no reasonable measurements can be performed. In fact, the processing time of the firewall chain and rules is less than 1 ms and thus not measurable using system time. Also, the speed of the TP-1 medium is so low, that even on full bus load no reasonable telegram delay can be measured using this platform.

6.2.3 Network Based Intrusion Detection System

To be able to validate that the requirements as stated in Section 5.4.1 can be fulfilled by the presented attack detection system, this section describes a prototype implementation of an IDS. A network based approach has been chosen. It is thus possible to avoid influencing the normal operation of the BAS, since no change of SACs is required, and also no performance penalty on those nodes is generated. A dedicated IDS can also be equipped with more processing and memory resources. Besides, such an IDS can be deployed to existing installations, since only an access to the BAN is required, which can be achieved easily. The following attack detection functionality has been realized in the KNX based prototype implementation of use case ⊗ (based on [106]):

- Alert on process data exchange between CAs, if it either violates a statically defined policy, or additional abnormal communication relationships are detected.
- Alert on management communication i.e. physical destination addresses.
- Alert on high bus load (the threshold needs to be defined a priori).

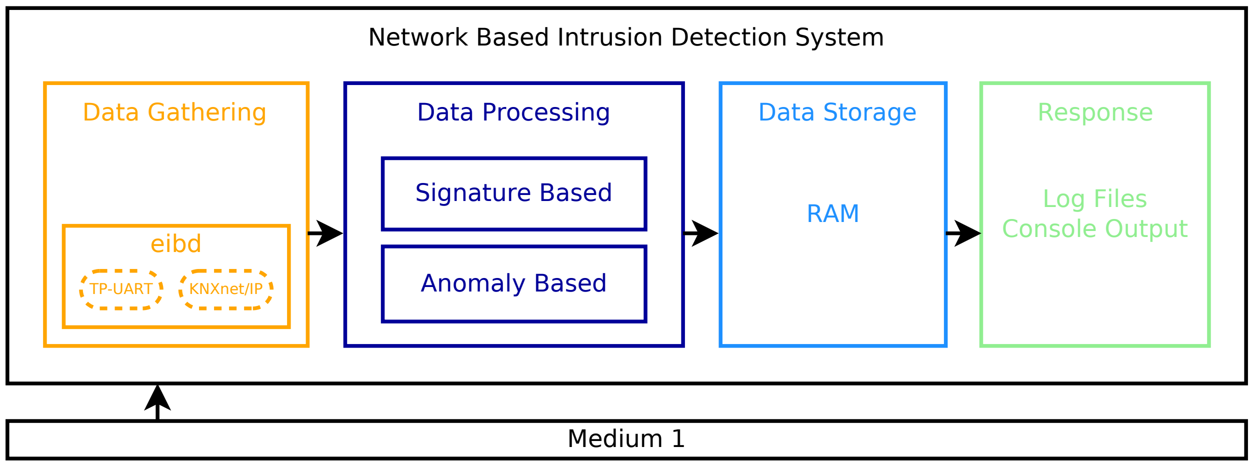


Figure 6.11: Network Based KNX Intrusion Detection System

Figure 6.11 shows the software structure of the prototype, which has been implemented on the same embedded Linux platform as the secure ICD. The data gathering component relies on one or more instances of eibd. KNX telegrams from various BANs can thus be gathered by the IDS.

The data processing component has been implemented in a hybrid way. A SIDS approach is used to detect the following issues:

- Violation of communication relationships: The policy (i.e. the security attribute $s_{DOMAIN\ Lighting}$) is provided, which defines the normal communication behavior between CAs. Likewise, security attributes are specified, which define forbidden communication behavior. Alerts are then immediately generated, if e.g. a sending address is not allowed to write to a specific receiving address.

- Management communication

- High bus load

Listing 6.8 shows the configuration containing the static rules for the prototype. Syntax is based on the SNORT IDS and enhanced with KNX specifics. A rule is structured as follows: `<command> <source address> -> <destination address> [APDU] [msg:"<message to display>"]`

Command can either be `alert`, to trigger the response component to e.g. log to a file or `pass` to allow communication. It is followed by the source and destination address, which is a group or physical address in KNX notation or `any` for a wild card. Then an optional specification of the APDU can be provided, to enable evaluating security attributes. Finally, a message can be specified, which is also sent to the response component.

```
[global]
# Configuration for first interface
[KNX-IF-1]
connectionurl=ip://auto.tuwien.ac.at:6720
# Alert on bus load higher than 60%
bus-load-warning=60

# Alarm on management communication i.e. physical destination addresses
alert any -> *.*.* (msg:"Management communication detected")

# Do not alarm on communication from secure SAC switch to secure SAC actuator
pass 1.2.2 -> 0/0/1 A_GroupValue_Write={0,1}
```

Listing 6.8: Configuration of Network Based Intrusion Detection System for Use Case ⊗

Additionally, an AIDS approach is implemented, where the IDS first gathers data from the BAN. The data processing component then uses anomaly based algorithms, to learn the normal communication relationships between CAs and also their behavior regarding telegram rates. Malicious communication between CAs can then be detected and alerted during runtime.

Implementation of the policy check in the data processing component can be realized using simple `if` statements (cf. Listing 6.9).

```
int checkRule(T_NetworkDataStorage *rule, T_Telegram *telegram) {
  // check source address
  if (checkSourceAddress(rule->source, telegram->source)) {
    // check destination address
    if (checkDestinationAddress(rule->destination, telegram->destination)) {
      // check APDU type and value
      if (checkAPDU(rule->apdu, telegram->data) {
        // rule matched, apply command
        return (rule->command);
[...]
```

Listing 6.9: Implementation of Intrusion Detection System

The data storage component simply stores relevant data in the memory and does not rely on a database. This allows implementation on simple non-Linux based MCUs.

The response component is also only implemented partially as a passive system, which simply displays alert messages on the standard output and logs them to files.

Experimental tests with the implementation show similar results regarding performance and memory overhead as the ICD prototype. No reasonably measurements can be performed during normal runtime, and even when stress tests have been executed at maximum bus speed of the TP-1 medium, the CPU load and memory usage are negligible. The requirement to provide a minimal detection latency is fulfilled, since every telegram is analyzed immediately and alerts are generated, if appropriate. Using a network based IDS also ensures, that the standard process data exchange between SACs is not influenced.

6.2.4 Management Device

Finally, a secure MD prototype has been implemented, to demonstrate the evaluation of the security attribute $s_{DP\ PAR\_OFFD}$ (cf. P.1) on a MD. The MD is being connected to the backbone, where it monitors devices based on active queries: It generates telegrams to detect malfunctions and attacks. Care has to be taken to not flood the BAN with traffic and influence the normal operation of the BAS. Thus, the query cycles have to be set appropriately, depending on the deployed technologies. The goal is to:

- Monitor the availability of devices (i.e. if they are on- or offline due to malfunctioning or DoS attacks).
- Monitor, if the security policy regarding the MDPs of CAs has been violated or not.

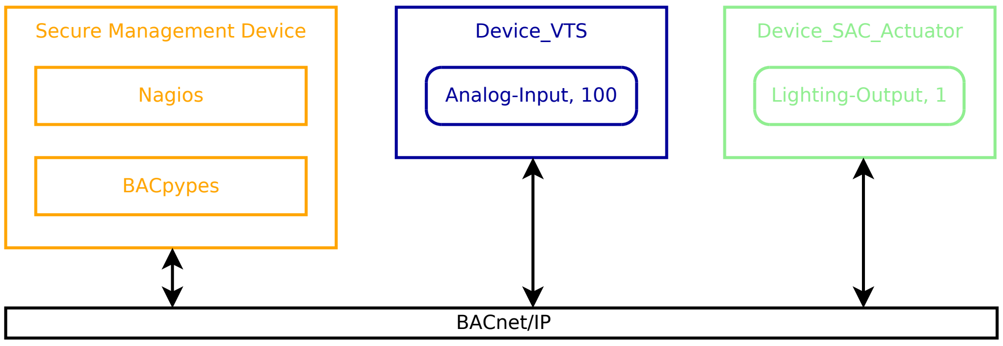


Figure 6.12: Secure BACnet Management Device

A setup consisting of a BACnet/IP BAN, a virtual BACnet device being implemented in the VTS (`Device_VTS`) and a secure SAC actuator based on KNXcalibur (`Device_SAC_Actuator`), and a secure MD has been deployed (cf. Figure 6.12). The device `Device_VTS` implements an `Analog-Input` object with ID `100`. The SAC is implemented as multi-protocol device and thus connected to KNX and BACnet/IP likewise. Mapping of the DPs is established using the generic application model. The object `Lighting-Output` with ID `1` contains the MDP `PAR_OFFD` of use case ⊗. The MD accesses the BAN using the BACpypes network stack[4] and implements a monitoring system using Nagios[5] [173], [174]. As shown in Figure 6.13, the availability of devices is checked using Internet Control Message Protocol (ICMP) `Echo Requests`. The security attribute $s_{DP\ PAR\_OFFD}$ (cf. P.1) is evaluated using the implemented Nagios script, which performs a `ReadProperty` request on `192.168.0.1, Lighting-Output 1, Off-Delay` on a configurable interval. Listing 6.10 shows the configuration of the Nagios plugin. Listing 6.11 shows its implementation in Python.

```
define command{
  command_name    check-bacnet-object
  command_line    $USER1$/check_bacnet -a $_HOSTADDRESS$ -t $_HOSTTYPE$ -i
      $_HOSTINSTANCE$ -p $_HOSTPROPERTY$ -w $_HOSTWARNING$ -c $_HOSTCRITICAL$
}
define host{
  use             host-template-bacnet
  host_name       Device_1_LO_1
  alias           Object_Lighting-Output-1
  _address        192.168.0.1
  _type           LightingOutput
  _instance       1
  _property       Off-Delay
  _critical       60:600
}
```

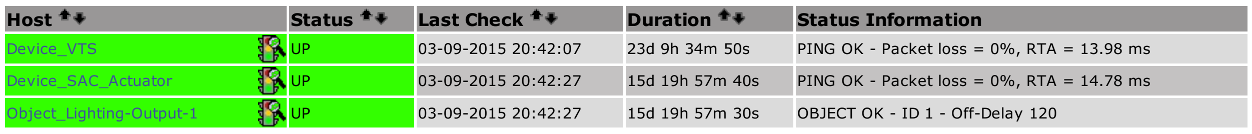
Listing 6.10: Configuration of Nagios Plugin

```python
import nagiosplugin

def main():
  argp = argparse.ArgumentParser(description=__doc__)
  argp.add_argument('-w', '--warning', metavar='RANGE', default='',
                    help='return warning if value is outside RANGE')
  [...]

  args = argp.parse_args()
  check = nagiosplugin.Check(
    Object(args.address, args.type, args.instance, args.property),
    nagiosplugin.ScalarContext('value', args.warning, args.critical))
    check.main(verbose=args.verbose)
```

Listing 6.11: Implementation of Nagios Plugin

To gain experimental results regarding the execution time of a single security attribute check, a performance analysis has been performed. The execution time of the Nagios

[4] `http://bacpypes.sourceforge.net/`, Last access: 2015/09/03
[5] `https://www.nagios.org/`, Last access: 2015/09/03



Host	Status	Last Check	Duration	Status Information
Device_VTS	UP	03-09-2015 20:42:07	23d 9h 34m 50s	PING OK - Packet loss = 0%, RTA = 13.98 ms
Device_SAC_Actuator	UP	03-09-2015 20:42:27	15d 19h 57m 40s	PING OK - Packet loss = 0%, RTA = 14.78 ms
Object_Lighting-Output-1	UP	03-09-2015 20:42:27	15d 19h 57m 30s	OBJECT OK - ID 1 - Off-Delay 120

Figure 6.13: Monitoring BACnet Devices and Objects

script has been measured for 100 successive executions. Minimum, maximum and mean execution time are shown in Table 6.2. Considerable overhead can be assumed, if larger amounts of security attributes need to be evaluated.

Device	Object ID	Minimum	Maximum	Mean
`Device_VTS`	100	174 ms	3189 ms	664 ms
`Device_SAC_Actuator`	1	175 ms	3144 ms	497 ms

Table 6.2: Performance Analysis of Execution Time

6.3 Enabling Security in Existing Installations

To validate that the presented CA architecture can be used to enhance security in current technologies and installations, this section is dedicated to additional case studies, demonstrating 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 unprotected. Case study 2 targets attacks performed on EnOcean based CAs, since no adequate protection mechanisms are available, yet. Case study 3 and Case study 5 deal with today's most common software vulnerabilities (cf. Figure 3.12). Case study 4 has been selected, since KNX seems to be one of the most spread technology today[6].

Case study 1 *Prevent attacks on current installations*

The analysis in Section 3.1 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

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

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 ETS for KNX based BAS or the corresponding configuration tool for BACnet based BAS.

Case study 2 *Detect attacks on current installations*

Regarding protection against attacks on EnOcean based CAs (cf. Section 3.2) it has to be distinguished whether the attack is performed passively or actively. In the former case, an adversary just listens to the BAN and does not actually transmit telegrams. Thus, no possibilities exist to prevent attacks which target the eavesdropping of unsecured teach-in telegrams or encrypted telegrams relying on the VAES encryption algorithm. In the latter case, an adversary generates telegrams on its own. Such a behavior can be detected if an IDS is deployed and within transmission range of the wireless adversary.

Case study 3 *Prevent attacks on KNX based SACs*

Listing 6.12 shows an implementation of the SAC light actuator of use case ⊗ on a standard KNX 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 6.14 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.

```
void main (void) {
 // do some initializations
 AppInit();
 // declare BYTE buffer of size 1
 BYTE buf[1];
 // set port to output
 PM0=0xFF;
 // start the main loop of the
      application program
  for(;;) {
   // if pdp L_MAN (id 0) has
        changed, copy 2 bytes into
        buf
   if(U._TestAndCopyObject
            (0, (void*) buf, 2)) {
    // if it is set to on
    if (buf[0] == 0x01) {
     // immediatley swich lamp 0 on
     P0_bit.no0 = 1;
    }
   }
   // handle switch off light after
        PAR_OFFD
  }
}
```

Listing 6.12: Implementation of Use Case ⊗ on a KNX Bus Interface Module-M130

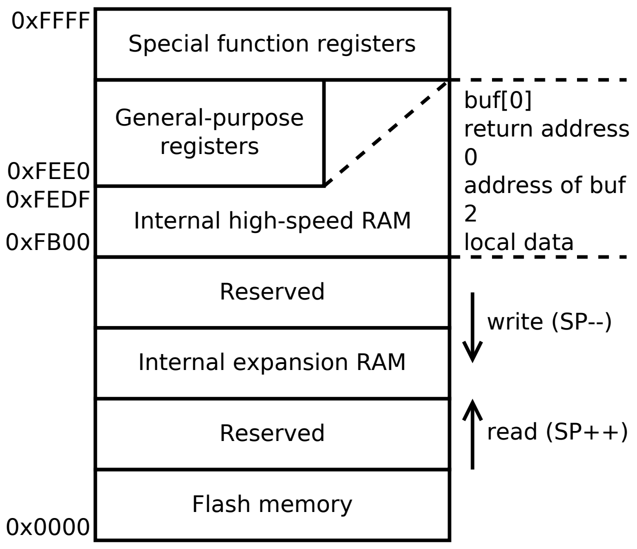


Figure 6.14: Memory Map (μPD78F0534) and Stack

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 presented before, an `ArrayIndexOutOfBoundsException` will be thrown by the SB and the CA will not have direct access to the control functions.

Case study 4 *Prevent attacks on KNX based BASs using secure ICDs*

Using the KNX technology, couplers are used to interconnect different physical lines. They 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. However, no mechanisms are present which allow context-based filtering. Thus, also only simple security policies can be monitored by those couplers. Using the presented secure ICDs, however, complex policies can be defined and be enforced by ICDs.

Case study 5 *Prevent Attacks on LonWorks based SACs*

Listing 6.13 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.

```
 1 // NV input: pdp L_MAN
 2 network input SNVT_switch nviLampValue;
 3 // lamp is connected to I/O 0 pin 0
 4 IO_0 output bit io_lamp_control = 0;
 5 // if flag is equal to TRUE
 6 when (flag=TRUE) {
 7   // switch lamp on
 8   io_out(io_lamp_control, 1);
 9   flag = FALSE;
10 }
11 // if flag is equal to FALSE
12 when (flag==FALSE) {
13   // handle switch off light after PAR_OFFD
14 }
15 // when new value of L_MAN is received
16 when (nv_update_occurs(nviLampValue)) {
17   // store value in flag
18   if (nviLampValue == 1) {
19     flag = TRUE;
20   }
21   else {
22     flag = FALSE;
23   }
24 }
```

Listing 6.13: Implementation of Use Case ⊗ 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. P.1), since the SB controls the execution of the CA and the policy specifies a maximum execution rate of the `switchoutput` function of 1Hz.

6.4 Security Evaluation

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 to prove Hypothesis 6.

First, it will be discussed, how the deployment of the secure CA architecture on the different device classes enables compliance with the FRs, as presented in Section 3.4.1. A summary is shown in Table 6.3.

***FR–memory access*:** To fulfill this FR, a secure software environment being deployed on a SAC (cf. Section 5.3) 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. This analysis has to be performed only once and does not have to be repeated on CA exchange. Code injection attacks are hardened, if a HA is used on the SAC (cf. Section 4.2). It has to be noted, that providing security for the system software and SB itself is of utmost importance. If flaws are present in the implementations, attacks cannot be prevented. In practice, this will not be easy to achieve. In the prototype implementations, no security analyses of the system software have been performed.

***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. Besides, a CA is not able to damage or overtake the SB. It has, however, to be noted that some limitations in the implementations of the prototypes are present. The code size of the used SB is targeted to be as small as possible. Therefore, it lacks several mechanisms of a full blown Java VM (e.g. no exceptions, threads or inheritance from native classes). In addition, code checking capabilities have been reduced to a minimum. It is required, that the Java compiler produces (semantically) valid bytecode in order not to crash the NanoVM. Full implementations such as the Oracle JRE, however, fulfill this requirement.

***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. High bus load or abnormal telegram telegram rates can be detected by an IDS, whereas the availability of CAs can be monitored by a MD. The desired detection latency needs to be adjusted with respect to the tolerable performance overhead.

In the following, it will be discussed, if and how the secure CA architecture complies with the ORs (cf. Section 3.4.2). A summary is shown in Table 6.4.

***OR–limited resources*:** The deployment of dynamic security measures being executed during runtime clearly imposes a performance overhead. Depending on the appli-

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 6.3: Security Evaluation of Functional Requirements

cation, 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. Although considerable performance overhead came up in the experimental tests with the matrix multiplications, the ongoing trend in hardware development with more available memory and processing power, facilitates the fulfillment of this OR. Also, no optimizations have been performed. They can be implemented, once the requirements as well as soft- and hardware specifications for a concrete CA are available.

***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. CAs being easier to understand are also less error prone. It can further be ensured that all (bus-)communication is standard compliant, even if the CA developer does not know a particular technology. Since the CA is separated from the system software, the latter has to be certified only once. This eases a possible certification process.

***OR–high level language support*:** The architecture supports the use of high-level languages, in particular Java. Standard Java toolchains can be used for development, offering object oriented development on SACs. Thus, the desired application behavior can be implemented more easily and BAS control tasks can be carried out very flexible and with an adaptable configuration. Portability of CAs can additionally be achieved due to their separation from the system software and the network stack.

***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. In fact, it is not easy to update the system software of MCUs, once they are deployed in BAS. This is, however, out of scope of this dissertation. Likewise, the security of the CA update process, and also the organizational process covering who and how updates to CAs can be performed after their installation, are not discussed.

***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. Besides, the concept is generic enough, that different network technologies are supported. It has to be noted, that no analyses regarding possible real-time requirements have been performed. Except for the safety domain (e.g. fire alarm systems), BAS do not require special treatment. At most, soft-real-time requirements are needed sometimes. Due to e.g. dynamic memory management of the SB, however, the implied delay or jitter can be quite high.

***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. This way, also existing, non-secure SACs can be monitored regarding security violations. It has to be noted, that existing CAs need to be adapted to be executable in a SB.

***OR–physical access*:** Detecting physical manipulations of SACs, requires special treatment in hardware. The required mechanisms are not discussed in this dissertation. Alarming of intrusions, however, can be performed by e.g. an IDS, monitoring other security related issues. ICDs and MDs can be assumed as being located in a secure environment (e.g. switchboard or server rack).

***OR–usability*:** Providing usability of security measures seems to be hardest task, when they are being put into practice. As shown in Section 3.1, 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. Some guidelines and recommendations to provide overall CA security are given in Section 7.2.

6.5 Summary

Various hypotheses have been formulated throughout this dissertation. This section briefly summarizes how they have been verified.

Hypothesis 1 covers the main problem statement of this thesis. It can be split into Hypothesis 2 – Hypothesis 6 and evaluated likewise. Hypothesis 2 is discussed in Section 2, which analyzes common standards and Section 5, which introduces a secure 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*
Secure CA Architecture	∼	+	+	∼	+	+	∼	*n/a*	*n/a*

Table 6.4: Security Evaluation of Organizational Requirements

and a mapping on these standards. Hypothesis 3 is validated in Section 3, which demonstrates that today's CAs can be attacked by an adversary. Hypothesis 4 is evaluated in Section 4, which illustrates that no sound protection mechanisms are available that fulfill the requirements of BAS. Hypothesis 5 is discussed in Section 5, which introduces a secure CA architecture relying on hybrid protection mechanisms. Finally, Hypothesis 6 is validated with the help of prototype implementations and case studies. Section 6 shows that the proposed architecture fulfills the requirements for secure CAs.

Summarizing, the overall security of the architecture relies on

- the security measures provided by e.g. system software, SB, ICDs, MDs not containing any flaws (e.g. buffer overflows) and not being bypassed,
- the interfaces from the CA to the system software being limited enough, and
- the user (e.g. system integrator) being able to determine, if the behavior of a CA defined in the policy is malicious.

7 Conclusion and Future Work

7.1 Summary

The history of IT security can serve both as a good and bad example for the future of BAS security. Important Internet-related protocols were developed for a virtually closed user community. Consequently, security was neglected and services and applications that used these protocols remained unprotected against security attacks. Due to the ubiquitous use of the Internet, unprotected services and applications became attractive for adversaries. The resulting economic damage through viruses, trojan horses, and worms is still clearly tangible today. To counteract these threats, protocol extensions and security mechanisms have been developed that are widely used today.

This dissertation shows, that BAS are equally prone to security attacks (as the IT world years before) because existing technologies lack state of the art security features. Besides, as the performed scans during this research show, security measures in the BAS domain are still deeply missing. Thousands of BASs are being directly connected to the Internet and allow unauthenticated and unauthorized access to their underlying DPs and CAs. Two BAS technologies have been analyzed, other relevant standards (e.g. LonWorks, Modbus) can be assumed as insecure, too. Security is currently still neglected in installations due to various reasons. First, security features have only been added recently. A lot of current installations (and most of the devices still being sold) use “old” protocols without appropriate security features. Second, there is a huge lack of security awareness in the BAS domain. Besides, even if a BAS provides security mechanisms and these mechanisms are enabled in installations, security weaknesses might be present in specifications and implementations. Based on a security analysis of EnOcean, Section 3.2 outlined as an example, that this standard had to undergo multiple improvements in the past. A weak encryption method, for instance, had been specified and some minor and major security issues (e.g. plain text private key transfer, static private keys) are still present.

Thus, it is mandatory to identify and set up security mechanisms as it has been done in the IT domain – otherwise adversaries will soon single out unprotected BAS as their next target on a large scale. In order to enhance security in BASs, a comprehensive approach is needed: On the one hand technologies need to provide mechanisms for secure communication, secure software and update routines as well as protection to provide availability. While secure communication has been addressed recently, a secure environment for CA has been missing. On the other hand also education covering security mechanisms and accompanying measures are needed in the BAS domain. Immediately, mechanisms such as firewalls helping to prevent access and VPNs allowing to connect remote sites need to be deployed in BASs.

This dissertation concentrates on the problems of secure CAs in BAS and contributes the following novelties:

- A comprehensive identification of security requirements for CAs: Until now, it has not been clear, which requirements hinder the use of available software protection techniques in the BAS domain and security awareness has been missing. As a result, thousands of BAS installations are being connected directly to the Internet, often without any security mechanisms available or enabled.

- An application model capable of depicting CAs in a formal way: Within this dissertation, 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.

- The concept of security attributes, being able to formally specify a security policy for a BAS: As has been shown in this dissertation, the available domain knowledge can be utilized to formulate a security policy and to provide an overall security.
- A framework, which allows the secure development and execution of CAs and enforcement of the defined security policy.
- Evaluation prototypes describing the experimental results of an integration of all components into a common secure BAS and validating the presented secure CA architecture.

Additional work, however, is required to formulate generic security policies. In fact, this dissertation has shown, that security attributes can be mapped onto today's open BAS technologies. A generic policy being ready to deploy on real installations, however, is missing. Using ontologies consisting of the definition of the application model, a mapping to the different technologies and a device description as knowledge base, seems to be a promising approach. A generic policy can then be formulated. Using the ontology it is possible to automatically express the security attributes for a specific SAC, ICD or MD. In fact, a broad agreement in standardization committees and between manufacturers is required to support developers, users and integrators in establishing secure BAS.

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.

Finally, an open point in this dissertation is the usability of security measures. If they are too difficult or complicated to deploy, it can be assumed that they will not be enabled in practice. Standardization committees and manufacturers have to take care on providing security by design and not to burden users.

7.2 Security Recommendations

The following section gives some final security recommendations. Comparing, [3] gives an insight into securing distributed and critical infrastructure.

- Security cannot be gained using security by obscurity (which is especially true for an open standard). A system has to be secure by design.
- Security needs to be considered for a whole system. Nobody will attack the most secure part of a system, but its weakest point. Therefore, multiple countermeasures and defense in depth need to be deployed.
- Security needs to be seen as a process and not as a one time event. Since a building's life cycle typically is 20 to 30 years, it is very likely, that security requirements will change during this time. Thus, a secure standard has to discuss this requirement, which is clearly missing in today's standardization committees.
- A secure standard needs to cover education (e.g. during certification) covering security requirements and measures. Maintenance, commissioning personnel and integrators are no security experts but domain experts. Simple and clear guidelines have to be available.
- During a building's life cycle multiple parties (e.g. electrician, integrator, visualization developer, maintainer) are involved. Each of them needs to be trusted to sustain security.
- Secure communication needs to be provided.
- Device attacks need to be prevented or detected to reduce the risk of unauthorized access or sabotage to a BAS installation. Secure software architectures (e.g. a secure system software/stack) and an API need to be provided to be able to develop secure CAs. Update routines need to be available, to be able to address security issues after installation. Embedded systems such as SACs need to be likewise update-able as it is common for PCs (e.g. Windows update).
- Protection to provide availability of installations (e.g. IDSs) is required.

Bibliography

[1] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti, "Control-Flow Integrity", in *CCS '05: Proceedings of the 12th ACM Conference on Computer and Communications Security*, Alexandria, VA, USA: ACM, 2005, pp. 340–353, ISBN: 1-59593-226-7.

[2] B. Aigner, "How to Hack EnOcean: An Evaluation of EnOcean's Security", Master's thesis, AAT-Lab@Department of Embedded Systems, FH Technikum Wien, Jun. 2014.

[3] J. Akerberg, "Towards Securing Distributed and Critical Infrastructure", in *Proc. 19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2014)*, Sep. 2014.

[4] Anonymous Author, "Once Upon a Free()", *Phrack*, vol. 11, no. 57, Aug. 2001.

[5] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha, "Hardware-Assisted Run-Time Monitoring for Secure Program Execution on Embedded Processors", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 12, pp. 1295–1308, 2006, ISSN: 1063-8210.

[6] D. Aucsmith, "Tamper Resistant Software: An Implementation", in *Proceedings of the First International Workshop on Information Hiding*, London, UK: Springer-Verlag, 1996, pp. 317–333, ISBN: 3-540-61996-8.

[7] *BACnet – A Data Communication Protocol for Building Automation and Control Networks*, ANSI/ASHRAE 135, 2012.

[8] F. Bachmann, "Evaluation of and Recommendations for the Security of EnOcean Radio Networks", Master's thesis, Technische Universität München Fakultät für Elektro- und Informationstechnik, Lehrstuhl für Sicherheit in der Informationstechnik, 2012.

[9] M. Bishop and M. Dilger, "Checking for Race Conditions in File Accesses", *Computing Systems*, vol. 9, pp. 131–152, 1996.

[10] M. Blum and S. Kannan, "Designing Programs that Check their Work", *Journal of the ACM (JACM)*, vol. 42, no. 1, pp. 269–291, 1995, ISSN: 0004-5411.

[11] H. Boehme, "Virtuelle Java-Maschinen für kleine eingebettete Systeme", PhD thesis, Technische Universität Carolo-Wilhelmina zu Braunschweig, 2007.

[12] B. Bray, "Compiler Security Checks In Depth", Microsoft Corporation, Visual Studio Technical Articles, 2002.

[13] D. Brickley and R. Guha, *RDF Vocabulary Description Language 1.0: RDF Schema*, W3C Recommendation, Feb. 2004.

[14] W. Burgstaller, S. Soucek, and P. Palensky, "Current Challenges in Abstraction Data Points", in *IFAC Int. Conf. on Fieldbus Systems and their Applications*, 2005, pp. 40–47.

[15] W. R. Bush, J. D. Pincus, and D. J. Sielaff, "A Static Analyzer for Finding Dynamic Programming Errors", *Softw. Pract. Exper.*, vol. 30, no. 7, pp. 775–802, 2000, ISSN: 0038-0644.

[16] P. Caseley and M. Hadley, "Assessing the Effectiveness of Static Code Analysis", *IET Conference Publications*, vol. 2006, no. CP515, pp. 227–237, 2006.

[17] H. Chang and M. J. Atallah, "Protecting Software Code by Guards", in *DRM '01: Revised Papers from the ACM CCS-8 Workshop on Security and Privacy in Digital Rights Management*, London, UK: Springer-Verlag, 2002, pp. 160–175, ISBN: 3-540-43677-4.

[18] K. J. Charatsis, A. P. Kalogeras, M. Georgoudakis, and G. Papadopoulos, "Integration of Semantic Web Services and Ontologies into the Industrial and Building Automation Layer", *EUROCON, 2007. Int. Conf. on Computer as a Tool*, pp. 478–483, Sep. 2007.

[19] S. Checkoway, A. J. Feldman, B. Kantor, J. A. Halderman, E. W. Felten, and H. Shacham, "Can DREs Provide Long-Lasting Security? The Case of Return-Oriented Programming and the AVC Advantage", in *Proceedings of EVT/WOTE 2009*, D. Jefferson, J. L. Hall, and T. Moran, Eds., USENIX/ACCURATE/IAVoSS, Aug. 2009.

[20] H. Chen and D. A. Wagner, "MOPS: an Infrastructure for Examining Security Properties of Software", University of California at Berkeley, Berkeley, CA, USA, Tech. Rep., 2002.

[21] Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, and M. H. Jakubowski, "Oblivious Hashing: A Stealthy Software Integrity Verification Primitive", in *IH '02: Revised Papers from the 5th International Workshop on Information Hiding*, London, UK: Springer-Verlag, 2003, pp. 400–414, ISBN: 3-540-00421-1.

[22] B. Chess and G. McGraw, "Static Analysis for Security", *IEEE Security and Privacy*, vol. 2, no. 6, pp. 76–79, 2004, ISSN: 1540-7993.

[23] *Chinese Standard for Home and Building Control based on KNX*, GB/Z 20965, SAC TC 124, 2007.

[24] C. S. Collberg and C. Thomborson, "Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection", in *IEEE Transactions on Software Engineering*, vol. 28, Aug. 2002, pp. 735–746.

[25] *Communication Systems for Meters and Remote Reading of Meters*, EN 13757, 2004.

[26] *Control Network Protocol Specification*, ANSI/EIA/CEA 709 Rev. B, 2002.

[27] M. Cova, V. Felmetsger, G. Banks, and G. Vigna, "Static Detection of Vulnerabilities in x86 Executables", *Computer Security Applications Conference, 2006. ACSAC '06. 22nd Annual*, vol. 22nd Annual, pp. 269–278, Dec. 2006, ISSN: 1063-9527.

[28] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks", in *Proc. 7th USENIX Security Conference*, San Antonio, Texas, Jan. 1998, pp. 63–78.

[29] C. Cowan, M. Barringer, S. Beattie, G. Kroah-Hartman, M. Frantzen, and J. Lokier, “FormatGuard: Automatic Protection from printf Format String Vulnerabilities”, in *SSYM’01: Proceedings of the 10th Conference on USENIX Security Symposium*, Washington, D.C.: USENIX Association, 2001, pp. 15–15.

[30] C. Cowan, S. Beattie, J. Johansen, and P. Wagle, “Pointguard™: Protecting Pointers from Buffer Overflow Vulnerabilities”, in *SSYM’03: Proceedings of the 12th Conference on USENIX Security Symposium*, Washington, DC: USENIX Association, 2003, pp. 7–7.

[31] C. Cowan, S. Beattie, C. Wright, and G. Kroah-Hartman, “RaceGuard: Kernel Protection from Temporary File Race Vulnerabilities”, in *In Proceedings of the Tenth USENIX Security Symposium*, USENIX Association, 2001, p. 12.

[32] G. Cretu, J. Parekh, K. Wang, and S. Stolfo, “Intrusion and Anomaly Detection Model Exchange for Mobile Ad-Hoc Networks”, in *Proc. 3rd IEEE Consumer Communications and Networking Conference CCNC 2006*, vol. 1, Aug. 2006, pp. 635–639.

[33] M. Debbabi, M. Saleh, C. Talhi, and S. Zhioua, “Security Evaluation of J2ME CLDC Embedded Java Platform”, *Journal of Object Technology*, vol. 5, no. 2, pp. 125–154, 2006.

[34] H. Dibowski, “Semantischer Gerätebeschreibungsansatz für einen automatisierten Entwurf von Raumautomationssystemen”, PhD thesis, Technische Universität Dresden, 2013.

[35] H. Dibowski, J. Ploennigs, and K. Kabitzsch, “Automated Design of Building Automation Systems”, *IEEE Transactions on Industrial Electronics*, vol. 57, no. 11, pp. 3606–3613, 2010.

[36] M. Dworkin, “Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication”, National Institute of Standards and Technology, Tech. Rep., May 2005, NIST Special Publication 800-38B.

[37] D. Dzung, M. Naedele, T. Von Hoff, and M. Crevatin, “Security for Industrial Communication Systems”, *Proceedings of the IEEE*, vol. 93, no. 6, M. Naedele, Ed., pp. 1152–1177, 2005, ISSN: 0018-9219.

[38] EnOcean Alliance – Technical Task Group Interoperability, *EnOcean Equipment Profiles (EEP), Version: 2.6*, Last Access: 2015/01/01, Dec. 2013.

[39] EnOcean GmbH, *AN509: Explanation of EnOcean Security in Applications*, Last Access: 2015/01/01, Sep. 2013.

[40] ——, *AN510: Adding Security to EnOcean Receivers*, Last Access: 2015/01/01, Sep. 2013.

[41] ——, *AN511: Advanced Security in Self-Powered Wireless Applications*, Last Access: 2015/01/01, Sep. 2013.

[42] ——, *EnOcean Link*, Last Access: 2015/01/01.

[43] ——, *Security of EnOcean Radio Networks, Version: 1.9*, Last Access: 2015/01/01, Dec. 2013.

[44] ——, *USB 300 Gateway*, Last Access: 2015/06/16, Sep. 2014.

[45] H. Etoh, *GCC Extension for Protecting Applications from Stack-Smashing Attacks (ProPolice)*, 2003.

[46] D. Evans, J. Guttag, J. Horning, and Y. M. Tan, "LCLint: A Tool for Using Specifications to Check Code", in *In FSE*, 1994, pp. 87–96.

[47] H. Feng, H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong, "Anomaly Detection Using Call Stack Information", in *Proc. Symposium on Security and Privacy*, O. Kolesnikov, Ed., 2003, pp. 62–75.

[48] D. Fensel, H. Lausen, A. Polleres, J. de Bruijn, M. Stollberg, D. Roman, and J. Dominque, "Enabling Semantic Web Services", in, 1st ed. Springer, 2007, ch. 3.

[49] A. Fernbach, W. Granzer, and W. Kastner, "Interoperability at the Management Level of Building Automation Systems: A Case Study for BACnet and OPC UA", in *Proc. of 16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA '11)*, Sep. 2011.

[50] E. A. Fisch and G. B. White, *Secure Computers and Networks: Analysis, Design and Implementation*. CRC Press, 1999.

[51] G. Forstner, "Security in Smart Homes - Eine modulare KNX Firewall", Master's thesis, AAT-Lab@Department of Embedded Systems, FH Technikum Wien, May 2013.

[52] A. Francillon and C. Castelluccia, "Code Injection Attacks on Harvard-Architecture Devices", in *CCS '08: Proceedings of the 15th ACM Conference on Computer and Communications Security*, Alexandria, Virginia, USA: ACM, 2008, pp. 15–26, ISBN: 978-1-59593-810-7.

[53] T. Garfinkel, "Traps and Pitfalls: Practical Problems in System Call Interposition based Security Tools", in *Proc. Network and Distributed Systems Security Symposium*, Feb. 2003.

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

[55] L. Gong, "Java Security: Present and Near Future", *Micro, IEEE*, vol. 17, no. 3, pp. 14–19, May 1997, ISSN: 0272-1732.

[56] L. Gong, M. Müller, H. Prafullchandra, and R. Schemers, "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java™Development Kit 1.2", in *USITS'97: Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, Monterey, California: USENIX Association, 1997, pp. 10–10.

[57] A. Gössling and M. Wollschläger, "On Working with the Concept of Integration Ontologies", in *Proc. 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08)*, Sep. 2008, pp. 709–712.

[58] W. Granzer, "Secure Communication in Home and Building Automation Systems", PhD thesis, Vienna University of Technology, Feb. 2010.

[59] ——, "Security in Networked Building Automation Systems", Master's thesis, Vienna University of Technology, Institute of Computer Aided Automation, 2005.

[60] W. Granzer, W. Kastner, G. Neugschwandtner, and F. Praus, "A Modular Architecture for Building Automation Systems", in *Proc. 6th IEEE International Workshop on Factory Communication Systems (WFCS '06)*, Jun. 2006, pp. 99–102.

[61] ——, "Security in Networked Building Automation Systems", in *Proc. 6th IEEE International Workshop on Factory Communication Systems (WFCS '06)*, Best Paper Award of WFCS '06, Jun. 2006, pp. 283–292.

[62] W. Granzer, F. Praus, and W. Kastner, "Security in Building Automation Systems", *IEEE Transactions on Industrial Electronics*, vol. 57, no. 11, pp. 3622–3630, Nov. 2010.

[63] L. Gu and J. A. Stankovic, "T-Kernel: Providing Reliable OS Support to Wireless Sensor Networks", in *SenSys '06: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, Boulder, Colorado, USA: ACM, 2006, pp. 1–14, ISBN: 1-59593-343-3.

[64] V. Gungor and G. Hancke, "Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches", *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 4258–4265, 2009.

[65] M. Hauswirth, C. Kerer, and R. Kurmanowytsch, "A Flexible and Extensible Security Framework for Java Code", Technical Report TUV-1841-99-14, Technical Univ. of Vienna, Tech. Rep., 2000.

[66] ——, "A Secure Execution Framework for Java", in *Proceedings of the 7th ACM Conference on Computer and Communications Security*, ACM, 2000, pp. 43–52.

[67] S. Hegler and M. Wollschläger, "The Semantic Web in Action: Semantically Enabled Device Descriptions", *5th IEEE Int. Conf. on Industrial Informatics*, vol. 2, pp. 1013–1018, Jun. 2007, ISSN: 1935-4576.

[68] heise Security. (May 2013). Kritische Schwachstelle in hunderten Industrieanlagen. Last Access: 2015/07/23, [Online]. Available: `http://heise.de/-1854385`.

[69] ——, (Apr. 2013). Vaillant-Heizungen mit Sicherheits-Leck. Last Access: 2015/07/23, [Online]. Available: `http://heise.de/-1840919`.

[70] I. Hiroaki, M. Edahiro, and J. Sakai, “Towards Scalable and Secure Execution Platform for Embedded Systems”, in *ASP-DAC '07: Proceedings of the 2007 Conference on Asia South Pacific Design Automation*, Washington, DC, USA: IEEE Computer Society, 2007, pp. 350–354, ISBN: 1-4244-0629-3.

[71] S. A. Hofmeyr, S. Forrest, and A. Somayaji, “Intrusion Detection Using Sequences of System Calls”, *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.

[72] D. G. Holmberg, “BACnet Wide Area Network Security Threat Assessment”, National Institute of Standards and Technology, Tech. Rep., 2003.

[73] G. Holzmann, “The Power of 10: Rules for Developing Safety-Critical Code”, *IEEE Computer*, vol. 39, no. 6, pp. 95–99, Jun. 2006, ISSN: 0018-9162.

[74] *Home and Building Electronic Systems (HBES)*, EN 50090, CENELEC, 1997-2007.

[75] B. Horne, L. R. Matheson, C. Sheehan, and R. E. Tarjan, “Dynamic Self-Checking Techniques for Improved Tamper Resistance”, in *Digital Rights Management Workshop*, 2001, pp. 141–159.

[76] M. Howard, D. LeBlanc, and J. Viega, *19 DEADLY SINS OF SOFTWARE SECURITY*. New York, NY, USA: McGraw-Hill, Inc., 2006, ISBN: 0072260858, 9780072260854.

[77] D. Hwang, P. Schaumont, K. Tiri, and I. Verbauwhede, “Securing Embedded Systems”, *IEEE Security and Privacy*, vol. 4, no. 2, pp. 40–49, Mar. 2006, ISSN: 1540-7993.

[78] IEEE Computer Society, *IEEE Standard for Local and Metropolitan Area Networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*, New York, Sep. 2011.

[79] H. Inoue, H. Inoue, A. Ikeno, M. Kondo, J. Sakai, and M. Edahiro, “VIRTUS: a New Processor Virtualization Architecture for Security-Oriented Next-Generation Mobile Terminals”, in *Proc. 43rd ACM/IEEE Design Automation Conference*, A. Ikeno, Ed., 2006, pp. 484–489.

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

[81] International Electrotechnical Commission, *Digital Addressable Lighting Interface*, IEC 62386, 2009.

[82] ——, *Function Blocks (FB) for Process Control*, IEC 61804-2, Oct. 2007.

[83] ——, *Function Blocks – Part 1: Architecture*, IEC 61499-1, Dec. 2011.

[84] ——, *Programmable Controllers - Part 3: Programming Languages*, IEC 61131-3, Dec. 2003.

[85] International Organization for Standardization, *Building Automation and Control Systems (BACS) – Part 2: Hardware*, ISO 16484-2, 2004.

[86] ——, *Building Automation and Control Systems (BACS) – Part 3: Functions*, ISO 16484-3, International Organization for Standardization, Dec. 2005.

[87] ——, *Building Automation and Control Systems (BACS) – Part 6: Data Communication Conformance Testing*, ISO 16484-6, 2009.

[88] ——, *Information Technology – Home Electronic System (HES) Architecture – Part 3-10: Wireless Short-Packet (WSP) Protocol Optimised for Energy Harvesting – Architecture and Lower Layer Protocols*, ISO/IEC 14543-3-10, Geneva, Switzerland, 2012.

[89] ——, *Information Technology – Home Electronic Systems (HES) Architecture*, ISO/IEC 14543-3, Geneva, Switzerland, 2006-2007.

[90] ——, *Information Technology – Security Techniques – Evaluation Criteria for IT Security*, ISO/IEC 15408, 2005.

[91] ——, *Building Automation and Control Systems (BACS) – Part 5: Data Communication Protocol*, ISO 16484-5, May 2014.

[92] ——, *Open Data Communication in Building Automation, Controls and Building Management – Control Network Protocol*, ISO/IEC 14908, 2008.

[93] T. Jensen, D. Le Metayer, and T. Thorn, "Verification of Control Flow Based Security Properties", in *Proc. IEEE Symposium on Security and Privacy*, D. Le Metayer, Ed., 1999, pp. 89–103.

[94] A. Judmayer, L. Krammer, and W. Kastner, "On the Security of Security Extensions for IP-Based KNX Networks", in *Proc. of the 10th IEEE International Workshop on Factory Communication Systems (WFCS'14)*, May 2014.

[95] S. Karnouskos, “Stuxnet Worm Impact on Industrial Cyber-Physical System Security”, in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, 2011, pp. 4490–4494.

[96] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman, “Communication Systems for Building Automation and Control”, *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1178–1203, Jun. 2005.

[97] J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith, “Proactive Detection of Computer Worms Using Model Checking”, *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 424–438, 2010.

[98] J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith, “Detecting Malicious Code by Model Checking”, in *GI SIG SIDAR Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA'05)*, K. Julisch and C. Krügel, Eds., ser. Lecture Notes in Computer Science, vol. 3548, Vienna, Austria: Springer, Jul. 2005, pp. 174–187, ISBN: 3-540-26613-5.

[99] V. Kiriansky, D. Bruening, and S. Amarasinghe, “Secure Execution Via Program Shepherding”, in *Proceedings of the 11th USENIX Security Symposium*, USENIX Association, Aug. 2002, pp. 191–206, ISBN: 1-931971-00-5.

[100] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, “seL4: Formal Verification of an OS Kernel”, in *SOSP '09: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, Big Sky, Montana, USA: ACM, 2009, pp. 207–220, ISBN: 978-1-60558-752-3.

[101] *KNX System Specifications – Application Note 158/13 v02 – KNX Data Security*, KNX Association, May 2013.

[102] *KNX System Specifications – Application Note 159/13 v05 – KNXnet/IP Secure*, KNX Association, Jul. 2015.

[103] *KNX System Specifications, Version 2.1*, KNX Association, ISO/IEC 14543-3, Jan. 2014.

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

[105] M. J. Kofler, "An Ontology as Shared Vocabulary for Distributed Intelligence in Smart Homes", PhD thesis, Vienna University of Technology, 2014.

[106] M. Kreilach, "Ein netzbasiertes Intrusion Detection System für KNX", Master's thesis, AAT-Lab@Department of Embedded Systems, FH Technikum Wien, May 2013.

[107] C. Krügel, "Network Alertness – Towards an Adaptive, Collaborating Intrusion Detection System", PhD thesis, Vienna University of Technology, 2002.

[108] C. Krügel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Automating Mimicry Attacks Using Static Binary Analysis", in *SSYM'05: Proceedings of the 14th Conference on USENIX Security Symposium*, Baltimore, MD: USENIX Association, 2005, pp. 11–11.

[109] W. Landi, "Undecidability of Static Analysis", *ACM Letters on Programming Languages and Systems*, vol. 1, no. 4, pp. 323–337, Dec. 1992.

[110] D. Larochelle and D. Evans, "Statically Detecting Likely Buffer Overflow Vulnerabilities", in *SSYM'01: Proceedings of the 10th Conference on USENIX Security Symposium*, Washington, D.C.: USENIX Association, 2001, pp. 14–14.

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

[112] *LonMark Functional Profiles: Lamp Actuator: 3040, Scene Controller: 3251, Scene Panel: 3250*, LonMark International, 1997.

[113] Louis-F. Stahl. (Aug. 2013). Kritisches Sicherheitsupdate für 200.000 Industriesteuerungen. Last Access: 2015/07/23, heise Security, [Online]. Available: `http://heise.de/-1934787`.

[114] S. Mao and T. Wolf, “Hardware Support for Secure Processing in Embedded Systems”, in *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, Apr. 2007, pp. 483–488.

[115] MaXX, “Vudo Malloc Tricks”, *Phrack*, vol. 11, no. 57, Aug. 2001.

[116] D. L. McGuinness and F. van Harmelen, *OWL Web Ontology Language Overview*, W3C Recommendation, Feb. 2004.

[117] J. McHugh, “Intrusion and Intrusion Detection”, *International Journal of Information Security*, vol. 1, no. 1, pp. 14–35, 2001.

[118] *Modbus Application Protocol V1.1b*, Modbus Organization, 2006.

[119] G. C. Necula and P. Lee, “Safe, Untrusted Agents Using Proof-Carrying Code”, in *Mobile Agents and Security*, London, UK: Springer-Verlag, 1998, pp. 61–91, ISBN: 3-540-64792-9.

[120] T. Newsham. (Sep. 2000). Format String Attacks, Guardent, Inc.

[121] A. One, “Smashing The Stack For Fun And Profit”, *Phrack*, vol. 7, no. 49, Nov. 1996.

[122] *Open Data Communication in Building Automation, Controls and Building Management – Control Network Protocol*, EN 14908, 2005.

[123] *Open Data Communication in Building Automation, Controls and Building Management – Home and Building Electronic System – Part 1: Product and System Requirements.* EN 13321-1, Brussels, Belgium, CEN, 2006.

[124] *Open Data Communication in Building Automation, Controls and Building Management – Home and Building Electronic System – Part 2: KNXnet/IP Communication.* EN 13321-2, Brussels, Belgium, CEN, 2006.

[125] J. Park, R. Sandhu, and J. Schifalacqua, “Security Architectures for Controlled Digital Information Dissemination”, in *Proc. 16th Annual Conference Computer Security Applications ACSAC '00*, Nov. 2000, pp. 224–233.

[126] C. P. Pfleeger and S. L. Pfleeger, *Security in Computing*. Prentice Hall Professional Technical Reference, 2002, ISBN: 0130355488.

[127] J. Plönnigs, B. Hensel, H. Dibowski, and K. Kabitzsch, "BASont - A Modular, Adaptive Building Automation System Ontology", in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 4827–4833.

[128] F. Praus, "A Versatile Networked Embedded Platform for KNX/EIB", Master's thesis, Vienna University of Technology, Institute of Computer Aided Automation, Automation Systems Group, 2005.

[129] F. Praus, T. Flanitzer, and W. Kastner, "Secure and Customizable Software Applications in Embedded Networks", in *Proc. 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08)*, Sep. 2008, pp. 1473–1480.

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

[131] F. Praus and W. Kastner, "Identifying Unsecured Building Automation Installations", in *Proc. 19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'14)*, Sep. 2014.

[132] ——, "Secure Control Applications in Building Automation Using Domain Knowledge", in *Proc. 8th IEEE International Conference on Industrial Informatics (INDIN'10)*, Jul. 2010, pp. 52–57.

[133] ——, "Spotting Unsecured KNX Installations", in *Proc. KNX Scientific Conference*, KNX Scientific Award 2014, Nov. 2014.

[134] ——, "User Applications Development Using Embedded Java", in *Proc. KNX Scientific Conference*, Nov. 2008.

[135] F. Praus, W. Kastner, and G. Neugschwandtner, "A Versatile Networked Embedded Platform for KNX/EIB", in *Proc. KNX Scientific Conference*, Nov. 2006.

[136] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in Embedded Systems: Design Challenges", *Transactions on Embedded Computing Systems*, vol. 3, no. 3, pp. 461–491, 2004, ISSN: 1539-9087.

[137] C. Reinisch, W. Granzer, F. Praus, and W. Kastner, "Integration of Heterogeneous Building Automation Systems Using Ontologies", in *Proceedings of 34th Annual Conference of the IEEE Industrial Electronics Society (IECON '08)*, Nov. 2008, pp. 2736–2741.

[138] V. Rijmen, A. Bosselaers, and P. Barreto, *Optimised ANSI C code for the Rijndael Cipher (now AES)*, http://www.iaik.tu-graz.ac.at/research/krypto/AES/, version 3.0, 2000.

[139] R. Riley, X. Jiang, and D. Xu, "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*, Washington, DC, USA: IEEE Computer Society, 2007, pp. 30–40, ISBN: 0-7695-2855-4.

[140] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-Oriented Programming: Systems, Languages, and Applications", *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 1, 2:1–2:34, Mar. 2012, ISSN: 1094-9224.

[141] A. D. Rubin and D. E. Geer, Jr., "Mobile Code Security", *IEEE Internet Computing*, vol. 2, no. 6, pp. 30–34, 1998, ISSN: 1089-7801.

[142] T. Sauter, D. Dietrich, and W. Kastner, *EIB Installation Bus System*. Publicis, 2000.

[143] A. Saxena and B. Soh, "Authenticating Mobile Agent Platforms Using Signature Chaining Without Trusted Third Parties", in *Proc. IEEE International Conference on e-Technology, e-Commerce and e-Service EEE '05*, Mar. 2005, pp. 282–285.

[144] D. Schachinger and W. Kastner, "Model-driven integration of building automation systems into web service gateways", in *Factory Communication Systems (WFCS), 2015 IEEE World Conference on*, May 2015, pp. 1–8.

[145] M. Schoeberl, "JOP: A Java Optimized Processor for Embedded Real-Time Systems", PhD thesis, Vienna University of Technology, 2005.

[146] C. Schwaiger and A. Treytl, "Smart Card Based Security for Fieldbus Systems", in *Proc. IEEE Conference on Emerging Technologies and Factory Automation (WFCS)*, vol. 1, 2003, pp. 398–406.

[147] S. Seifried, L. Krammer, and W. Kastner, "A Reliable and Flexible KNX Gateway", in *Proc. KNX Scientific Conference*, Oct. 2014.

[148] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni, "A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors", in *Proc. IEEE Symposium on Security and Privacy S&P*, M. Bendre, Ed., 2001, pp. 144–155.

[149] R. Sekar, V. Venkatakrishnan, S. Basu, S. Bhatkar, and D. C. DuVarney, "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*, Bolton Landing, NY, USA: ACM, 2003, pp. 15–28, ISBN: 1-58113-757-5.

[150] H. Shacham, "The Geometry of Innocent Flesh on the Bone: return-into-libc Without Function Calls (on the x86)", in *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, Alexandria, Virginia, USA: ACM, 2007, pp. 552–561, ISBN: 978-1-59593-703-2.

[151] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, "On the Effectiveness of Address-Space Randomization", in *CCS '04: Proceedings of the 11th ACM Conference on Computer and Communications Security*, Washington DC, USA: ACM, 2004, pp. 298–307, ISBN: 1-58113-961-6.

[152] N. Shaylor, D. N. Simon, and W. R. Bush, "A Java Virtual Machine Architecture for Very Small Devices", *SIGPLAN Not.*, vol. 38, no. 7, pp. 34–41, 2003, ISSN: 0362-1340.

[153] J. E. Smith and R. Nair, "The Architecture of Virtual Machines", *IEEE Computer*, vol. 38, no. 5, pp. 32–38, 2005, ISSN: 0018-9162.

[154] Solar Designer. (Aug. 1997). return-to-libc Attack.

[155] L. Sommaruga, A. Perri, and F. Furfari, "DomoML-env: An Ontology for Human Home Interaction", in *SWAP 2005: Proc. of the 2nd Italian Semantic Web Workshop*, P. Bouquet and G. Tummarello, Eds., vol. 166, Dec. 2005.

[156] S. Soucek and D. Loy, "Vertical Integration in Building Automation Systems", in *Proc. 5th IEEE INDIN*, Jun. 2007, pp. 81–86.

[157] P. Stanley-Marbell and L. Iftode, “Scylla: A Smart Virtual Machine for Mobile Embedded Systems”, in *3rd IEEE Workshop on Mobile Computing Systems and Applications, WMCSA2000*, Dec. 2000.

[158] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas, “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*, Boston, MA, USA: ACM, 2004, pp. 85–96, ISBN: 1-58113-804-0.

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

[160] *Tunneling Component Network Protocols Over Internet Protocol Channels*, ANSI/EIA 852, 2002.

[161] R. Venkitaraman and G. Gupta, “Static Program Analysis of Embedded Executable Assembly Code”, in *CASES '04: Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, Washington DC, USA: ACM, 2004, pp. 157–166, ISBN: 1-58113-890-3.

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

[163] ——, *Building Automation and Control Systems (BACS) – Fundamentals for Room Control*, VDI 3813-1, May 2011.

[164] ——, *Building Automation and Control Systems (BACS) – Legislation, Technical rules*, VDI 3814-2, Jul. 2009.

[165] ——, *Building Automation and Control Systems (BACS) – Room Control Functions (RA functions)*, VDI 3813-2, May 2011.

[166] ——, *Product Data Exchange in the Building Services*, VDI 3805, 2002–2011.

[167] D. A. Wagner, “Janus: An Approach for Confinement of Untrusted Applications”, Master's thesis, University of California, Dec. 1999, p. 65.

[168] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham, "Efficient Software-Based Fault Isolation", in *SOSP '93: Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, Asheville, North Carolina, United States: ACM, 1993, pp. 203–216, ISBN: 0-89791-632-8.

[169] D. S. Wallach, D. Balfanz, D. Dean, and E. W. Felten, "Extensible Security Architectures for Java", *SIGOPS Oper. Syst. Rev.*, vol. 31, no. 5, pp. 116–128, 1997, ISSN: 0163-5980.

[170] D. S. Wallach, "A New Approach to Mobile Code Security", PhD thesis, Princeton University, 1999.

[171] D. Wallach and E. Felten, "Understanding Java Stack Inspection", in *Proc. IEEE Symposium on Security and Privacy*, E. Felten, Ed., 1998, pp. 52–63.

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

[173] R. Woischke, "BACnet Monitoring", AAT-Lab@Department of Embedded Systems, FH Technikum Wien, Tech. Rep., Feb. 2013.

[174] ——, "Entwicklung eines BACnet Monitoring Tools auf Basis von Nagios", AAT-Lab@Department of Embedded Systems, FH Technikum Wien, Tech. Rep., May 2013.

[175] J. Zachary, R. Brooks, and D. Thompson, "Secure Integration of Building Networks into the Global Internet", National Institute of Standards and Technology, Tech. Rep., 2002.

[176] J. Zambreno, A. Choudhary, R. Simha, B. Narahari, and N. Memon, "SAFE-OPS: An Approach to Embedded Software Security", *ACM Trans. Embed. Comput. Syst.*, vol. 4, no. 1, pp. 189–210, 2005, ISSN: 1539-9087.

[177] X. Zhang, L. van Doorn, T. Jaeger, R. Perez, and R. Sailer, "Secure Coprocessor-Based Intrusion Detection", in *EW10: Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, Saint-Emilion, France: ACM, 2002, pp. 239–242.

[178] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl, and M. A. Vouk, "On the Value of Static Analysis for Fault Detection in Software", *IEEE Transactions on Software Engineering*, vol. 32, no. 4, pp. 240–253, 2006, ISSN: 0098-5598.

[179] *ZigBee: Home Automation Public Application Profile*, Version 1.1, ZigBee Alliance, Feb. 2010.

[180] *ZigBee Specification*, ZigBee Alliance, San Ramon, Sep. 2012.

[181] T. Zillner, "ZigBee Exploited. The Good, the Bad and the Ugly", cognosec, Tech. Rep. 1.1, Aug. 2015.

[182] M. Zitser, R. Lippmann, and T. Leek, "Testing Static Analysis Tools Using Exploitable Buffer Overflows from Open Source Code", *SIGSOFT Softw. Eng. Notes*, vol. 29, no. 6, pp. 97–106, 2004, ISSN: 0163-5948.

[183] *Z-Wave System Design Specification: Z-Wave Protocol Overview*, Zensys A/S, Fremont, 2005.

Lebenslauf Friedrich Praus

1. Persönliche Informationen

Name:	FH-Prof. Mag.rer.soc.oec. Dipl.Ing. Praus Friedrich
Adresse:	Hallergasse 11/29, 1110 Wien, Österreich
Telefon:	+43 699 11075818
E-Mail:	fritz@praus.at
Internet:	http://www.praus.at/
Geburtsdatum:	08.01.1981
Nationalität:	Österreich

2. Ausbildung

2006-jetzt:
Doktoratsstudium der technischen Wissenschaften – Technische Universität Wien
Dissertationstitel: „Secure Control Applications in Smart Homes and Buildings“

2007-2008:
Mag.rer.soc.oec. in Informatikmanagement (mit Auszeichnung) – Technische Universität Wien

2000-2005:
Dipl.-Ing. in Informatik (mit Auszeichnung) – Technische Universität Wien

1999-jetzt:
Bundesheer (Milizoffizier, S6)

3. Beruflicher Werdegang

2014-2015:
Studiengangsentwicklung: Mitglied des Entwicklungsteams für den Bachelor Studiengang Smart Homes und Assistive Technologien, FH Technikum Wien

2014-jetzt:
Projektleiter: Stadt Wien Projekt ViTAL: Institut für Embedded Systems, Fachhochschule Technikum Wien

2014:
Verleihung des Titels FH Professor

2012-jetzt:
FFG benefit Projekt moduLAAr: Projektverantwortlich am Institut für Embedded Systems, Fachhochschule Technikum Wien

2010-2014:
Stadt Wien Stiftungsprofessor Ambient Assistive Technologies: Institut für Embedded Systems, Fachhochschule Technikum Wien

2007-2010:
Forschungsassistent: Institut für rechnergestützte Automation, Technische Universität Wien, *seBAS - Security in Buidling Automation* (FWF Projekt)

2006-2007:
Forschungsassistent: Institut für integrierte Sensorsysteme, Österreichische Akademie der Wissenschaften, Wr. Neustadt, *IMAGINE - Introduction of Master Group Based Industrial Ethernet Highly precise clock synchronization and more* (BRIDGE Projekt)

4. Publikationen

Diplomarbeit:

Friedrich Praus, A Versatile Networked Embedded Platform for KNX/EIB, Vienna University of Technology, Institute of Computer Aided Automation, 2005

Artikel:

Wolfgang Granzer, **Friedrich Praus**, and Wolfgang Kastner. Security in Building Automation Systems. *IEEE Transactions on Industrial Electronics*, 57(11):3622-3630, November 2010.

Bücher:

Wolfgang Kastner, **Friedrich Praus**, Georg Neugschwandtner, Wolfgang Granzer, KNX. In *The Industrial Electronics Handbook, Second Edition*, Part 4: Industrial Communication Systems, J.D. Irwin, and B. M. Wilamowski, Eds., chapter 42, pages 42-1 – 42-14. CRC Press, 2011

Workshops und Konferenzen:

1. **Friedrich Praus** and Wolfgang Kastner. Spotting Unsecured KNX Installations. *In Proc. KNX Scientific Conference,* November 2014. KNX Scientific Award 2014.
2. **Friedrich Praus** and Wolfgang Kastner. Identifying Unsecured Building Automation Installations. *In Proc. 19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2014)*, number 978-1-4799-4845-1, September 2014.
3. Richard Wagner, Peter Wolff, Klaus Schaeffer, **Friedrich Praus**. Ambient Assitive Technologies: The Mobile Robot P3AAT, P*roceedings of the Austrian Robotics Workshop (ARW-13)*, May 2013
4. Wolfgang Granzer, **Friedrich Praus**, Peter Balog. Source Code Plagiarism in Computer Engineering Courses. *3rd International Conference on Education, Training and Informatic (ICETI12)*, March 2012, Paper ID: EB266TP.
5. Johannes Kropf, Barbara Prazak-Aram, Lukas Roedl, **Friedrich Praus**, and Christian Siegel. Large Scale Integration and Evaluation of AAL Technologies in Eastern Austria - the moduLAAr Project. In *AAL Forum*, volume Session D3, September 2013.
6. Richard Isaacs, **Friedrich Praus**. Design & Development of a Prototype Android App for a KNX Home. In *Proc. KNX Scientific Conference*, November 2012.
7. Luka Samardzija, **Friedrich Praus**. ANT goes KNX - an Open Platform Gateway for ANT and KNX, KNX Scientific Conference. In *Proc. KNX Scientific Conference*, November 2012.
8. Wolfgang Granzer, **Friedrich Praus**, and Peter Balog. Source Code Plagiarism in Computer Engineering Courses. In *Proc. 3rd International Conference on Education, Training and Informatic (ICETI12)*. Paper ID: EB266TP, March 2012. Best Presentation Award.
9. **Friedrich Praus**, Christian Reinisch, Paul Leitner, and Wolfgang Kastner. Open Source Approaches to Integrate KNX into Media Centers. In Proc. *KNX Scientific Conference*, November 2010.
10. **Friedrich Praus** and Wolfgang Kastner. Secure Control Applications in Building Automation Using Domain Knowledge. In *Proc. 8th IEEE International Conference on Industrial Informatics (INDIN '10)*, 2010, pages 52-57, July 2010.
11. Wolfgang Granzer, Daniel Lechner, **Friedrich Praus**, and Wolfgang Kastner. Securing IP Backbones in Building Automation Networks. In *Proc. 7th IEEE International Conference on Industrial Informatics (INDIN '09)*, pages 410-415, June 2009.

12. **Friedrich Praus**, Wolfgang Granzer, and Wolfgang Kastner. Enhanced Control Application Development in Building Automation. In *Proc. 7th IEEE International Conference on Industrial Informatics (INDIN '09)*, pages 390-395, June 2009.
13. **Friedrich Praus** and Wolfgang Kastner. User Applications Development Using Embedded java. In *Proc. KNX Scientific Conference*, November 2008.
14. Christian Reinisch, Wolfgang Granzer, **Friedrich Praus**, and Wolfgang Kastner. Integration of Heterogeneous Building Automation Systems Using Ontologies. In *Proceedings of 34th Annual Conference of the IEEE Industrial Electronics Society (IECON '08)*, pages 2736-2741, November 2008.
15. **Friedrich Praus**, Thomas Flanitzer, and Wolfgang Kastner. Secure and Customizable Software Applications in Embedded Networks. In *Proc. 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '08)*, pages 1473-1480, September 2008.
16. **Friedrich Praus**, Wolfgang Granzer, Georg Gaderer, and Thilo Sauter. A Simulation Framework for Fault-Tolerant Clock Synchronization in Industrial Automation Networks. In *Proc. of 12th IEEE Conference on Emerging Technologies and Factory Automation (ETFA '07)*, pages 1465-1472, September 2007.
17. **Friedrich Praus**, Wolfgang Kastner, and Georg Neugschwandtner. A Versatile Networked Embedded Platform for KNX/EIB. In *Proc. KNX Scientific Conference*, November 2006.
18. Christian Reinisch, Wolfgang Granzer, **Friedrich Praus**, and Wolfgang Kastner. Wireless Communication in KNX/EIB. In *KNX Scientifc Conference*, November 2006.
19. Wolfgang Granzer, Wolfgang Kastner, Georg Neugschwandtner, and **Friedrich Praus**. Security in Networked Building Automation Systems. In *Proc. 6th IEEE International Workshop on Factory Communication Systems (WFCS '06)*, pages 283-292, June 2006.
20. Wolfgang Granzer, Wolfgang Kastner, Georg Neugschwandtner, and **Friedrich Praus**. A Modular Architecture for Building Automation Systems. In *Proc. 6th IEEE International Workshop on Factory Communication Systems (WFCS '06)*, pages 99-102, June 2006.
21. **Friedrich Praus** and Wolfgang Kastner. A Versatile Networked Embedded Platform for KNX/EIB. In *Junior Scientific Conference*, pages 59-60, April 2006.
22. **Friedrich Praus**, Wolfgang Kastner, and Oliver Alt. Yet Another All-Purpose EIBNet/IP Gateway. In *Proc. KNX Scientific Conference*, October 2004.

5. Awards

1. KNX Scientific Award 2014
2. Best Presentation Award der 3rd International Conference on Education, Training and Informatic für *Source Code Plagiarism in Computer Engineering Courses*.
3. Best Paper Award des 6th IEEE International Workshop on Factory Communication Systems für *Security in Networked Building Automation Systems*

6. Invited Talks

29.10.2014: ENISA European Cyber Security Month, Security in Smart Homes and Buildings

04.06.2013: Wiener Technikgespräche, Smart Homes – Intelligentes Wohnen durch Automatisierung