JOHANNES KEPLER
UNIVERSITY LINZ

INFORMATION SECURITY OF INDUSTRIAL AUTOMATION PRODUCTS AND MEASURES OF IMPROVEMENT

Author
Fabian Fisecker

Submission
Institute of Networks and Security

Thesis Supervisor
Univ.-Prof. Priv.-Doz. DI Dr. René Mayrhofer

Assistant Thesis Supervisor
DI Dr. MLBT Heinrich Schmitzberger

July 2017

Master's Thesis

to confer the academic degree of

Master of Science

in the Master's Program

066 921 Computer Science

Table of Contents

Abstract

Information Security is a trending topic in the world. The development of digitalization with its latest developments (like the Internet of Things) has brought us computers in all kinds of shapes into our everyday environment. However, no day passes without troubling news of digital systems failing in terms of information security. These systems suffer from problems like collection, disruption, denying or destroying of confidential information system resources.

This thesis describes the current state of information security in the industrial environment. In a theoretical approach, the issue of how to improve the information security of industrial automation products is analysed. Thereby it is investigated how to improve the process of developing new products with a focus on the information security of the provided software. The second investigation settles on how to improve the security of already existing products.

With a practical information security analysis, based on an industrial product of the medium-sized Austrian automation business Bernecker + Rainer Industrie Elektronik Ges.m.b.H., this approach is immediately tested to complete the thesis.

The aim of the thesis is to help the automation industry make headway into the big issue of ensuring information security in products specifically produced for an industrial environment.

Abstract

Informationssicherheit (Information Security) ist ein weltweit diskutiertes Thema, das sich immer mehr in den Schlagzeilen der Presse wiederfindet. Der Fortschritt der Digitalisierung und ihre letzten Ausprägungen wie das „Internet der Dinge“ bringen Computer in jeglicher Form und Größe in unser tägliches Leben. Es vergeht jedoch kaum ein Tag, an dem nicht über das Sammeln, Entwenden, Vorenthalten oder Zerstören von vertraulichen digitalen Informationen im öffentlichen, geschäftlichen oder privatem Raum berichtet und diskutiert wird.

Diese Masterarbeit beschreibt den aktuellen Stand der Technik der Informationssicherheit im industriellen Automatisierungsumfeld. Mittels eines theoretischen Ansatzes wird anhand aktueller Forschungen und Standards gezeigt, wie man die Informationssicherheit von industriellen Produkten nachhaltig verbessern kann.

Dabei wird im Speziellen analysiert, wie man den Prozess zur Erzeugung der Software dieser Produkte so anpassen kann, dass Informationssicherheit schon per Design inkludiert wird. Darauffolgend wird ein zweiter Ansatz diskutiert, wie man bei bereits existierenden Produkten mit der Informationssicherheit umgeht.

Mittels einer praktischen Analyse der Informationssicherheit eines Industrieprodukts des österreichischen Automatisierungsunternehmens Bernecker + Rainer Industrie Elektronik Ges.m.b.H. wird die Arbeit abgerundet.

Das Ziel dieser Masterarbeit ist es eine Hilfestellung zu bieten, um nachhaltig die Informationssicherheit von industriellen Automatisierungsprodukten zu verbessern.

1. Introduction

1.1. Motivation

Information Security is a widely-discussed topic known to almost every technology-interested human on the planet. The media coverage of security problems is overwhelming and not a day goes by without news about new security breaches and their impact.
Moreover, the topic of embedded information security is gaining attention. Reports like [1] show that networks of routers, webcams and other embedded devices are frequently taken over by malicious attackers. These botnets are then used against infrastructure and can affect even potent services with distributed denial-of-service attacks.

Security experts like Bruce Schneier have been emphasizing this issue for years:
“We're at a crisis point now with regard to the security of embedded systems, where computing is embedded into the hardware itself -- as with the Internet of Things. These embedded computers are riddled with vulnerabilities, and there's no good way to patch them.” [2]

In contradiction to this matter, the trend of the “Industrial Internet of Things” (IIoT) demands that embedded devices controlling critical infrastructure should connect to the internet and communicate with each other.

The question remains why this is even an issue in our modern and technologically-advanced world? The IT industry and the internet community should have learned their lesson. Security is known to be a needed asset. It has to be implemented by design and starts with the first line of code of software and with the first layer of hardware. Developers, administrators and users are aware of the problem and are trained to ensure security in their respective fields. IT companies are aware that they cannot solve security issues on their own. They are opening up to the security community and support research like bug bounty programs with financial means. Even though these measures are helping, security problems are still occurring on a daily basis.

However, in the world of embedded devices and in the automation industry, these aspects do not apply immediately. Currently operatives like component vendors, system integrators or machine operators lack information security know-how.
The embedded devices do not implement modern information security technology. A rapid time-to-market is demanded and the pricing is very competitive. Additionally, the period of usage is usually higher for embedded devices. Ten to 20 years are very common for industrial appliances to be used. And most critically the maintenance process in this field is not designed to deal with updates. Machines are often certified in the commissioning process and no changes can occur, whether in hardware or software, or a recertification has to be applied, which leads to high costs. [3]

There are indications that governments all over the world [4][5] are striving for regulations and for the enforcement of information security in the future, and the argument of liability is paramount. It is thus good advice for every company in this industry to take measures in the field of information security to keep up with these requirements coming in the near future.

This thesis focuses on the problem of securing embedded devices and attempts to give advice on how to improve the given information security in a sustainable manner. As a practical example, an industrial embedded device from the Austrian vendor Bernecker + Rainer Industrie Elektronik Ges.m.b.H. (henceforth abbreviated as B&R) will be tested.

1.2. Introduction to Information Security

Information Security, sometimes shortened to InfoSec, seeks to ensure three objectives, the so-called CIA Triangle.
The three pillars of this triangle are confidentiality, integrity and availability.

This definition applies worldwide and is also stated in legal definitions, for example in the US Code, Title 44, Chapter 35, Subchapter 3, § 3542:
“The term “information security” means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide

- integrity, which means guarding against improper information modification or destruction, and includes ensuring information nonrepudiation and authenticity;
- confidentiality, which means preserving authorized restrictions on access and disclosure, including means for protecting personal privacy and proprietary information; and
- availability, which means ensuring timely and reliable access to and use of information.” [6]

Any potential forms of action against these objectives are so called Information Security **Threats**, or as the RFC4949 (Internet Security Glossary) [7] puts it:
“1a. (I) A potential for violation of security, which exists when there is an entity, circumstance, capability, action, or event that could cause harm. [...]
1b. (N) Any circumstance or event with the potential to adversely affect a system through unauthorized access, destruction, disclosure, or modification of data, or denial of service.[...]”
Threats to a system are always linked to some kind of soft spots or chinks in a given system. Without such **vulnerabilities** in the system the threats would be pointless as an actual threat action (e.g. an attack) would cause no harm. The RFC4949 defines vulnerabilities as “A flaw or weakness in a system's design, implementation, or

operation and management that could be exploited to violate the system's security policy."

Nearly every system has vulnerabilities and, therefore, potential threats which could cause harm, and pose a risk. Therefore, we have to clarify the term **risk**, which is defined in the RF4949 as:

"An expectation of loss expressed as the probability that a particular threat will exploit a particular vulnerability with a particular harmful result."

In modern approaches of handling information security, it is assumed that no system is 100% safe and, therefore, managing the risk of vulnerability exploitation is a key chapter when thinking about information security.

The definition and type of attacks against the information security objectives will be discussed in the ensuing chapter 1.4. "Attacks and Countermeasures".

1.2.1. Information Security Engineering

The field of security engineering has existed as a thriving working area for several centuries. For example, the history of cryptography reaches back to the ancient Greeks and the first occurrences of locks were found in Mesopotamia.

Information security engineering focuses on designing and building information technology systems and thereby ensuring the information security objectives. This is somewhat counterintuitive to the approach of classic software engineering, as Schneier argues in [8]:

"Programming a computer is straightforward: keep hammering away at the problem until the computer does what it's supposed to do. [...]

Writing a secure computer program is another matter entirely. Security involves making sure things work, not in the presence of random faults, but in the face of an intelligent and malicious adversary trying to ensure that things fail in the worst possible way at the worst possible time… again and again."

To conquer this matter, the process of engineering secure programs will be analyzed thoroughly in chapter 2.3. "Information Security in the Development Lifecycle".

1.3. Introduction to Embedded Systems

In the last years, the majority of newly built processors have been used for embedded systems. [9] For example, as of 2015, more than 15 billion ARM-type microcontrollers were sold, leading to an 85% market share in mobile application processors (e.g. in smartphones). [10]

[11] states that there are estimates that over 50 billion microcontroller units will be sold in 2019 "dwarfing PC-style microprocessor sales".

So what exactly is an embedded system? A possible definition could be the following one:

“A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function.” [12]
Some sources add properties like special resources limitations (e.g. in computing power, general size or power source) to this definition.
It is obvious that this definition fits many systems in our world today. The classification in categories like general purpose computer, dedicated system or embedded system is rather abstract and there may always be devices where these definitions intersect. As an example, prior to the introduction of smartphones it was common to categorize a mobile phone as an embedded device with a dedicated purpose. Newer generations of smartphones are considered more and more a general purpose device with powerful hard- and software and, therefore, are no longer bound to specific use cases.
Evidently it is necessary to focus on different aspects of embedded systems to get a better understanding of their special properties and dedication.

1.3.1. Characteristics of Embedded Systems

The characteristics of embedded systems are small size and weight, low power consumption and low costs per unit. Such systems should have a high reliability, high performance and often operate as real-time systems. This leads to limited processing resources and makes them more difficult to program and interact with. These restrictions and the fact that embedded devices are increasingly connected to the internet make it more difficult to protect embedded systems against attackers.
Beside ordinary microprocessors, embedded systems often use microcontrollers. In both cases the processors which are used, can be general purpose microprocessors, specialized in a certain class of computation or custom designed ones for this specific application. An example for a common standard class of dedicated processors is the digital signal processor. Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance.

1.3.2. Application Areas of Embedded Systems

In the past, embedded systems were used for safety-critical industrial systems. For example, they were used to control power plants, rockets and satellites. This is also true today and will be discussed in chapter 1.5. “Introduction to Industrial Control Systems.”
Nowadays, embedded devices can also be found in transportation systems like planes or automobiles and in telecommunication systems like routers, network bridges or telephone switches. They are used in medical equipment and as home control systems. Even in things that we wear, embedded systems are increasing, for example smart watches or other wearable devices.
This enumeration is not a complete list of their application fields. It is just a brief overview of the areas where embedded systems are established and will become increasingly important in the future.

1.3.3. **Security Requirements**

The security requirements of embedded devices can be very different depending on the function of the device. For instance, a nuclear power plant control system needs a higher level of security than a coffee machine. This means that there is no security solution that fits all embedded devices.
In general, the increasing connectivity of such devices makes security of embedded systems more important than ever.
Security requirements need to consider the cost of a potential security failure, the risk of an attack, possible attack vectors and the cost of implementing a security solution.

An excerpt of common security requirements that need to be considered are stated by [13]:

- User identification, authentication and authorization: is a mechanism that ensures that only authorized users have access to the system and the different resources of the system. It also can verify the identity of the user if needed. Besides classical methods, embedded systems often use biometric identification techniques for identification and verification.
- Secure storage: embedded systems need to store information of users who have access to the system resources. This information can be sensitive like PINs, credit card numbers, personal data and information for authorization. As a result it is highly important to protect information of this kind.
- Secure network access: is a mechanism that provides access to the system resources only if the device is authorized.
- Secure communications: data in a public network goes through a number of untrusted intermediate points. Therefore, embedded systems should have functions that ensure data confidentiality, data integrity and user authentication. This can be achieved by using cryptographic methods.
- Secure content: enforces the usage restrictions of the digital content used in the system.
- Availability: the embedded system functions are always obtainable for the legitimate users and perform the intended functionality.

1.3.4. **Challenges of Embedded Systems and Their Security**

Compared with general purpose computers there are some properties of embedded systems that can lead to restrictions concerning information security. [14]

- Limited resources in terms of energy, communication and power: Many of the embedded systems have lower computational power compared to classic systems. Therefore, it is not always possible to use firewalls, Intrusion Detection Systems or sophisticated cryptographic techniques. Many embedded systems

are restricted in consuming energy. Possible security solutions lead to additional energy consumption, which contradicts this requirement.
- Physical accessibility: A further problem of embedded systems is that attackers often have easy physical access to embedded devices (e.g. stealing smartcards). This is a potential danger as an attacker could get the private key, for example, through a side-channel-attack as described later.
- Restricted maintainability: In classical systems it is possible to update a system in case of a known vulnerability. This is often not valid for embedded systems because software patches cannot be installed that easily or some functions are hardware-related and, therefore, not patchable. So, it is very important to think of security aspects in the design phase of an embedded system.
- Special cost sensitivity: Embedded systems are often developed on a very low cost per unit basis. The margin for profit per device is very low and the business model relies on economy of scales. Therefore, to have a competitive advantage, vendors tend to keep the hardware costs extremely low. For example, an 8-bit microcontroller instead of a 16-bit will be used whenever possible, even if this will imply that no state-of-the-art cryptographic key can be handled by this system.
- Strict safety requirements: Embedded systems must meet the industry standards in functional safety to address the issue of being protected from harmful conditions.
- Long life circle: Embedded systems are typically used much longer than general purpose computers. Building systems that will fulfill the security requirements for many years is a challenging task.
- Deployment in large numbers and in a hostile environment: Once an embedded device is developed, they are often produced in large numbers. If there is an existent attack against such a device, the attack can be performed against all others as well.

1.3.5. Different Levels of Embedded Systems Security

Because of the properties that are mentioned above, embedded security needs to include all abstraction layers. We can differ between five abstraction levels: [15]

- Protocol level: considers the design of protocols to achieve security objectives.
- Algorithm level: includes the design of cryptographic primitives that are used at the protocol level.
- Architecture level: consists of embedded software techniques to prevent software hacks.
- Microarchitecture level: deals with the hardware design of the building blocks.
- Circuit level: is concerned with the lowest level of hardware systems design to prevent attacks at the physical layer.

An embedded system needs to be secure at each layer. Security issues may affect only one single level which is called a single-level security issue. Furthermore, there may be trans-level security issues where a bug-fix needs to consider more than one level.

1.4. Attacks and Countermeasures

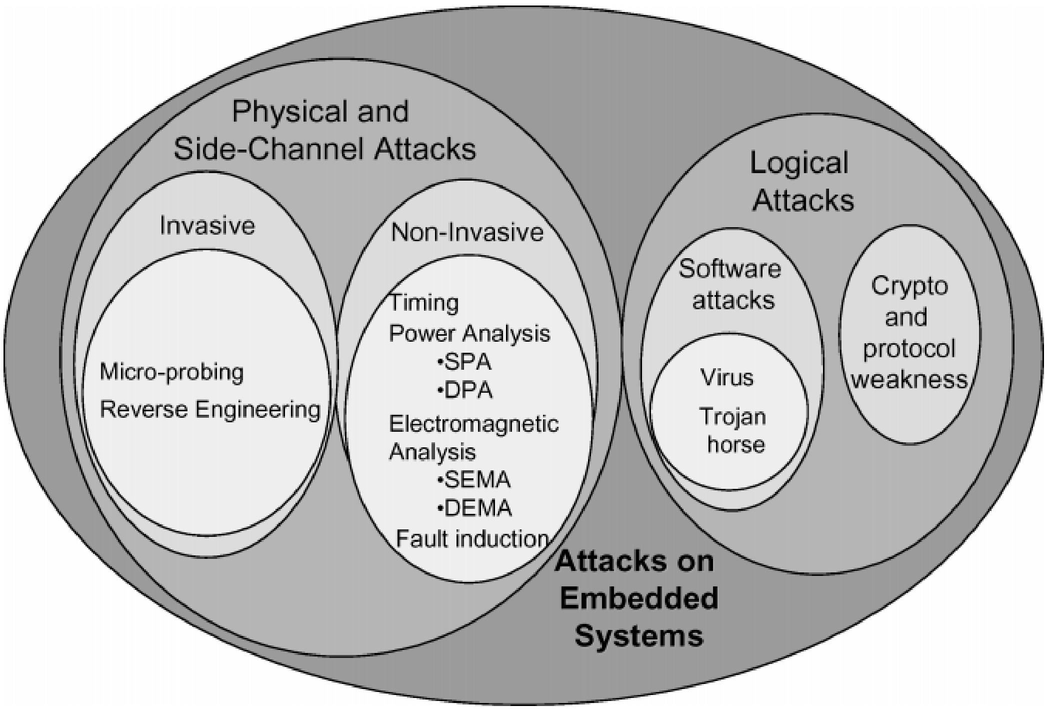


Figure 1 – Attacks on Embedded Systems[1]

1.4.1. Physical Attacks

Physical attacks can be categorized in two dimensions:

- invasive or non-invasive
- active or passive

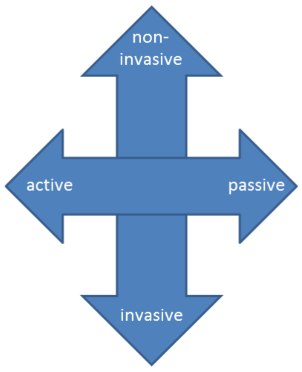


Figure 2 - Dimensions of Physical Attacks[2]

[1] Taken from: Ravi S., Raghunathan A., Kocher P. and Hattangady S. - Security in Embedded Systems: Design Challenges - http://users.ece.gatech.edu/~dblough/8823/embedded_security.pdf
[2] [selfmade diagram by author]

1.4.1.1. **Invasiveness**

Invasive physical attacks have to temper with the embedded device (in most cases with the interfaces, chips or the processor itself). This often includes opening the cartridge or packaging of the device and making direct contact with a conductive item, e.g. by soldering or pinning. As a logical consequence of smaller packaging sizes and more increasing complexity of devices such attacks are getting more demanding for potential attackers.
Non-invasive attacks make no contact at all and just need physical presence of the device. The attacker only observes externally available information and therefore the analysis itself is undetectable for the device.
In between are so-called semi-invasive attacks, which often need some sort of package opening, but no direct contact with conducting material.

1.4.1.2. **Activity**

Active physical attacks rely on signal tampering or other means of altering the device to obtain a certain result.
Passive attacks just observe and infer from these observations. The resources of the target system are not affected by the passive attack.

1.4.1.3. **(Semi-) Invasive Attacks**

Attacks that need direct access to internal components of an electronic device are called invasive attacks. Typically these types of attacks are the most expensive, time-consuming and require very knowledgeable attackers with sophisticated equipment like microscopes, probes, positioning devices, laser cutters, soldering stations, focused ion beam stations, etc.
In order to get to the internal components of a device it has to be opened (e.g. the casing of a USB stick) or in harder cases the packaging of the controller has to be removed. More sophisticated hardware has to be relieved of the passivation layer of the chip in order to get to the internal lanes. [16]
In most cases this leaves tamper evidence of the attack or may even destroy the device.
The major benefit of invasive attacks is the near unlimited possibility in extracting information of the target system, if it is understood correctly. But smaller packaging sizes, special countermeasures and increasing chip complexity are rendering invasive attacks more demanding than ever.
A Semi-invasive attack is a new type of attack, situated between non-invasive and invasive attacks. Similar to invasive attacks, the depackaging of the target device is necessary to get access to the internals of the chip. The difference to invasive attacks is that there is no need to achieve direct electrical contact to the lanes or chips. A

common example of a semi-invasive attack is the injection of UV-light into an EPROM to disable the security fuse. [17]

A more comprehensive list of typical representatives of invasive and semi-invasive attacks is:

- Decapsulation (this is often a prerequisite for further invasive methods)
- Laser cutting
- Focused Ion Beam analysis, deposition and ablation
- General reverse engineering
- Chip modification
- Fault-Injection (by UV-light, heat, optical, photon injection...)
- Laser and IR scanning
- Optical Imaging
- Microprobing

1.4.1.4. Side-Channel Attacks

Side-channel attacks are often symptomatic in being passive, non-invasive physical attacks. These attacks are based on physical properties from a real implementation of a cryptographic algorithm. An attacker measures physical properties while a cryptographic operation is performed. These measurements can be used to gain some additional information for cryptographic analysis. Traditionally, the quantification of the security level is often based only on mathematical properties. Side-channel attacks can give information about internal states. A side-channel analysis is much less general than a traditional one because it is specific to a given implementation. It requires that an attacker has access to the cryptographic device and can make physical measurements. In some scenarios, this is not very realistic. On the other hand, they can be considered very plausible if the device is a smart card that draws power from an external, untrusted source. [18]

There are many physical properties that can be measured and used by a side-channel attack. For example, an attacker can use the timing which is needed for cryptographic operations or power consumption behavior. A few examples of side-channel attacks are:

1.4.1.4.a Timing Attacks

Every cryptographic operation takes time and can differ based on the inputs. If such operations include secret parameters, these timing variation can leak information. With statistical methods it could be possible to reconstruct the secret parameters. The basic assumptions of the timing analysis are:

The runtime of a cryptographic analysis depends on the key. A sufficient number of encryption operations can be performed while the key stays the same. The time can be measured with a known error. If the error is small, fewer samples are needed.

1.4.1.4.b Power Consumption Attacks

The power consumption of a cryptographic device is a further physical property that can be used to gain information about operations and their parameters. This kind of attack is only practicable to hardware implementations of cryptographic systems such as smartcards.

Attacks that are based on power consumption are divided into two groups:

Simple Power Attacks:
Such an attack tries to guess what particular instruction is being performed at a certain time from a power trace. It involves a direct interpretation of power consumption measurements collected during cryptographic operations. The amount of power consumed varies depending on the instruction that is carried out.

Differential Power Attacks:
This attack consists of data collection and data analysis which uses statistical methods to gain information. Differential power attacks automatically detect correlated regions in the power consumption of a device. These attacks can be automated and no information about the target implementation is required.

1.4.1.4.c Electromagnetic Attacks

Electromagnetic attacks analyze the emitted electromagnetic radiation of an electronic device using an induction coil. It is also sometimes referred to as Van Eck phreaking after the researcher Wim van Eck, who demonstrated a possible attack on cathode ray tube displays by detecting its electromagnetic emissions. Popular targets of research in this field are smart cards, FPGAs, smartphones and general computers. Attacks are often targeted against cryptographic functionality, for example against the very popular symmetric encryption algorithm AES. In special conditions (e.g. with caching disabled, as the attack focuses on the bus between CPU core and memory), [19] shows an attack against 256bit AES encryption with equipment in the price-range of 200€.
As with power attacks, electromagnetic attack methods can be categorized in simple or differential electromagnetic analysis.

1.4.1.4.d Acoustic Attacks

It is possible to analyze acoustic sounds that are emitted by computers, including their peripherals. The devices vulnerable for these attacks are typically printers, keyboards or other input/output devices. For example, the acoustic sound emissions from printers with ultrasonic printing-heads can be used to reconstruct the printed information without actually seeing it.
Newer research, like [20] uses special acoustic timing attacks against CPUs performing cryptographic operations. The attack was fueled by analyzing ultrasonic sounds emitted by the mainboard and power supply capacitors and inductors.

1.4.1.5. Countermeasures

In the case of physical attacks, the attack range looks overwhelming at the beginning. But there are many possibilities to mitigate such attacks or at least make them exponentially harder and costlier. In essence, the countermeasure to physical attacks is tamper protections.
Some possibilities in this field are:

- protective casing (glued/em protection/..)
- chemical-mechanical-planarization
- multiple-layers, smaller size transistors
- higher frequency, less power
- memory access protection
- bus encryption
- crypto-processors
- secure ASIC/FPGA/custom IC
- top-layers with detecting sensors
- internal voltage and clock frequency sensors
- randomization of data
- masking, generation of noise

As tamper resistant devices have a long history, most of these countermeasures are well-known and could be selectively applied to any newly developed device.
However, the industry has shown that most of these possibilities are rarely used, because they are too costly to implement. These mechanisms are used only for special devices (e.g. hardware security modules or smartcards).
In the mass industry of low-cost embedded devices the engineering that goes into the device is kept to a bare minimum. Security concerns are rarely part of the requirement phase of a new product and even if the engineers had the know-how to implement such security features, they usually lack the time to do so.
This leads to the simple conclusion that in terms of physical attack countermeasures there is often just security by obscurity (which does not work in reality).

1.4.2. Logical Attacks

Logical attacks often represent well-known attacks from the non-embedded IT-industry. They are focused on the software-level and rely on weaknesses in software design, e.g. in the application layer, network protocols or APIs.
Typical attacks today are centered on the Ethernet interface of a device, but also all other logical interfaces that can be reached by an attacker are a potential threat vector.
Categorization of software attack types is done differently throughout the field of research. Typically, malicious software is separated into types like Virus, Trojan horse, Worm, Rootkit, Trapdoor, Logic Bomb, etc. depending on the behavior and manifestation in the system.

[21] separates the attack types by the intended result of the attack:

- data collection (e.g. packet sniffing, keystroke monitoring or database siphoning)
- stealth (e.g. hiding data, processes, users of a system,..)
- covert communication (allowing remote access without detection, transferring sensitive data out of the system)
- command and control (allowing remote control of a software system, sabotage or denying system control – DoS)

[13] tries to divide logical attacks in three categories, which will be investigated further:

- Classic Software Attacks
- Protocol Weaknesses
- Cryptographic Weaknesses

1.4.2.1. Software Attacks

Classic software attacks are focused on weaknesses in the implementation or design of the targeted device. A common way to better understand software attacks is to analyze and categorize the vulnerabilities that are exploitable by these attacks: [22]

- validation errors (e.g. input, origin or target validation)
- authentication errors (e.g. operations executable by unauthorized users)
- serialization/aliasing errors (e.g. same name of different objects)
- boundary checking errors (e.g. buffer overflows)
- domain errors (e.g. access to implementation details)
- weak or incorrect design errors (e.g. weak protocol or cryptographic algorithm – will be discussed separately)
- other exploitable logic errors

It is vital that software engineers acknowledge the fact that code quality in software is a key element to ensure the information security of a given system. When thinking of security in programs, even skilled engineers tend to bring up solutions involving cryptography or complex authorization systems. But the basic and more common risks to a software system are simple programming errors and lack of software quality.

1.4.2.2. Protocol Weaknesses

Modern embedded devices are connected to other systems via interfaces, often times via Ethernet, but there are, of course, others (e.g. CAN-Bus in the automotive industry). Some of these protocols are very old and have design weaknesses.
Other possibilities are that the libraries handling these protocols are outdated.
One example scenario is that embedded devices with Ethernet interfaces are not using modern TCP Stacks which can handle source-port randomization or other current security features.

Another problem is that embedded devices often use proprietary protocols to connect to devices from the same vendor. This may look fine at the beginning, as the public does not know details of the design of these protocols. In many cases, these protocols do not use any encryption or even integrity checking, which leads to security by obscurity. When analyzed properly, attackers can often use these protocols to get access to the device or at least trigger some sort of buffer overflow, etc. as these protocols are typically not as well tested as standard protocols.

1.4.2.3. Cryptographic Weaknesses

Even if some sort of cryptographic API is used on the device, these are often times outdated or weakly implemented. Typically embedded devices lack the chance to update their libraries regularly, so attackers only have to find out which version of the cryptographic library is used and search for vulnerabilities.

An example of this is that many embedded devices advertising some sort of encryption are using older standards of SSL/TLS.
Because of the simple nature of modern attacks (e.g. DROWN, POODLE, BEAST, etc.) the probability of such attacks will likely increase.
Another example is the usage of deprecated algorithms, like the Data Encryption Standard (DES), which is considered insecure nowadays, as the key length of 56bit is too short.

1.4.2.4. Countermeasures

To mitigate the risk of logical attacks in modern software applications, a clean software design and architecture is needed. Most of the problems that are exploited by malicious software are design flaws or simply bugs in the developed software. A detailed analysis of logical attack countermeasures will be discussed in chapter 2.“Improving System Security of Industrial Products”.
The biggest challenge for embedded devices though is the lack of updatability of the software. At least this problem could be mitigated in the future, as more and more embedded devices are connected to the internet, so automated updates could be possible and are already done by some embedded systems (e.g. routers). Counterintuitively such automated update systems are a key entry-point for logical attacks. Vulnerabilities in an updater are very critical, as attackers could use the updater to run malicious code and infiltrate the underlying system.
The companies developing embedded devices need to be aware and understand that tackling these security issues is more important than ever.

1.5. Introduction to Industrial Control Systems

1.5.1. ICS - Industrial Control Systems

The general term ICS is often used as an umbrella term for all kind of systems in the industrial field that control some parts of industrial technology. The best analogy is to compare the abbreviation with the term “IT” when it comes to computer technology in general.

As the US National Institute of Standards and Technology puts it: “Industrial control system (ICS) is a general term that encompasses several types of control systems, including supervisory control and data acquisition (SCADA) systems, distributed control systems (DCS), and other control system configurations such as skid-mounted Programmable Logic Controllers (PLC) often found in the industrial sectors and critical infrastructures.” [23]

The underlying terms SCADA, DCS and PLC can be seen as subset of ICS with different characteristics, mainly in the size of the regarding system. In modern environments, the terms are often used synonymously and the borders between the different systems are indistinct.

1.5.2. SCADA

SCADA stands for Supervisory Control and Data Acquisition, which oftentimes stands for big industrial systems with assets in various locations. These assets are controlled and monitored centrally by graphical user interfaces. The control system acts on a high-level for supervisory purposes. Most tasks are event-driven, when a subsystem alerts a warning or error state.
A good analogy is a nuclear power plant with its different sensors and actuators in various environments, e.g. connection to power grid, controlling of the nuclear fuel and control-rods, cooling, steam turbine control, plumbing, etc. All of these parts are controlled by different systems, potentially from different vendors. To be able to control all of these distributed systems in a central monitoring room where the reactor operators have to oversee the incoming data of all these parts to safely operate the plant is one key requirement of a SCADA architecture.

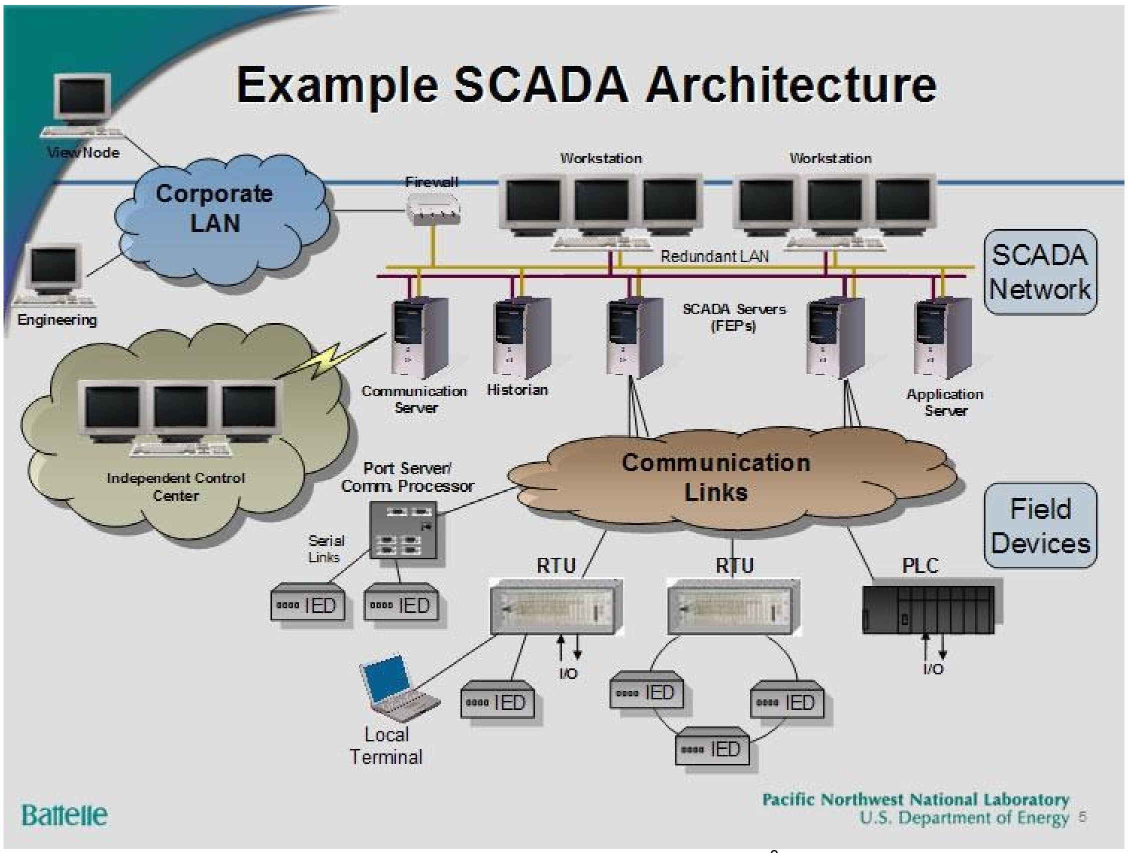


Figure 3 - Example SCADA Architecture[3]

1.5.3. **DCS**

The term Distributed Control System is often used when a local industrial environment serves a distinct purpose. For example, if the special target of a factory is to produce all the plastic injection molding parts, then the ICS equipment used to operate and monitor these processes are often called DCS. Each element of this process is controlled by a dedicated controller (e.g. a PC running dedicated control software) making the whole process autonomous from others.

This is also the main difference between centralized control systems where all of these processes would be controlled from one central control station. The decentralization aspect means that many of these DCS can be located in different geographical locations in a plant and work autonomously.

A DCS differs from a SCADA system because it is centered on fulfilling the process it was designed for. SCADA systems are oriented towards data-gathering. Their major goal is to acquire data and to log the actions and alarms of a system. [24]

[3]Taken from: Pacific Northwest National Laboratory - http://placidtech.com/files/scada.jpg

1.5.4. **PLC**

Programmable Logic Controllers are components to control a specific application or machine that performs single or multiple steps of a process. A PLC provides the control loop to operate an ICS component but is used to monitor data or supervise only in a few scenarios. It is often the source of data which is used in a centralized operator station.

PLCs are the backbone of a complex automated environment, running the actual software that controls the underlying machine actuators and sensors (such as acoustic, optical, thermal or any other type of sensor).

Ruggedized components are used as its hardware and a real-time operation system (RTOS) ensures the deterministic cycle of sensor and actuator interaction.

The specialized field of research regarding industrial control systems has led to a common standard in programming for these devices: the IEC 61131. In Part 3 of the standard 5 programming languages were defined:

- Ladder diagram (LD)
- Function block diagram (FBD)
- Structured text (ST)
- Instruction list (IL)
- Sequential function chart (SFC)

In modern environments with increasing requirements, PLCs take over SCADA or DCS functionality. As the hardware and software in the PLCs gets more capable, it is more common that the monitor and communication functionality is implemented in the PLC software, in addition to the specific process functionality. [25]

1.5.5. **ICS Components, Requirements and Architecture**

Historically, ICS components have had little resemblance to IT hardware or topology. These components were mostly built with proprietary hardware and software and, therefore, also have not been such an easy target to be exploited by attackers. The reason for this is not that these systems had less programmatic flaws or better security designs. The major reason was the lack of interest in creating specific attacks for these proprietary systems. Of course this sense of false security is very dangerous, as skilled attackers could easily exploit these vulnerable systems as well.

Nowadays, ICS components and infrastructure are similar or related to IT consumer hardware with less proprietary parts:

“As ICS are adopting IT solutions to promote corporate business systems connectivity and remote access capabilities, and are being designed and implemented using industry standard computers, operating systems (OS) and network protocols, they are starting to resemble IT systems.” [23]

Therefore, security concerns known from the IT environment are more and more applicable in the industrial sector.

To differentiate classic IT systems to the IT systems used to control industrial equipment the term “Operational Technology” (OT) was forged.
The key difference between OT and IT in respect of security is the different prioritization of key security goals:

- IT: Confidentiality > Integrity > Availability
- OT: Availability > Integrity > Confidentiality

This means that in the IT industry confidentiality of information is the highest goal and availability requirements can be afflicted if necessary (e.g. if the loss of critical information is at hand).
In the OT industry availability is the highest goal and cannot be restricted by other security needs or measures (e.g. even if an unauthorized access to the system is detected, critical operation functions must still be available). This is especially the case if the health, safety or environment (HSE) may be at harm.

In conclusion, to ensure the information security of such industrial control systems, the automation industry will have to take measures in developing these systems in a more secure way. An outlook on how to do that will be provided in the next chapter.

2. Improving System Security of Industrial Products

2.1. Attack Vector Taxonomy

It is important to prioritize the effort to improve information security.

According to [26], the main attack vectors for embedded systems are in a descending order:

- Internet facing devices
- Local or remote network access to the device
- Direct physical access to the device
- Physical proximity of the device
- Misc/Other (specific configurations, etc.)

This is in accordance with the Common Vulnerability Scoring System (CVSS), which rates the severity of the attack vector in the following descending order:

- Network
- Adjacent (which refers to a attack vector limitation, either to a shared physical or logical network, but not across the OSI layer 3 boundary, e.g. a router in the internet)
- Local
- Physical

The consequence is to focus the first efforts on logical attacks and on improving the security measures in the software parts of the underlying products (as they are mainly affected by logical attacks).
More specifically the main threat vector is the interface connecting to the Ethernet network or (if applicable) directly connecting to the internet.
To narrow the focus of this thesis we will look specifically at ensuring the information security of the software products.
Ensuring physical security and security from side-channel attacks could be the focus of further work in this field.

2.2. Security Standards

Worldwide security experts are working on standards, guidelines, regulations and best practices to ensure the protection and security of technology products. Led by the IT industry, these publications help information security staff find the correct approach to solve problems in their respective fields.
In this overview, an introduction to the most popular and best suitable standards is given.

2.2.1. ISA/IEC 62443

"ISA is an American National Standards Institute (ANSI) accredited organization. ISA administers United States Technical Advisory Groups (USTAGs) and provides secretariat support for International Electrotechnical Commission (IEC) and International Organization for Standardization (ISO) committees that develop process measurement and control standards."[27]

ISA/IEC 62443 (former ISA99) is a series of technical reports seeking information security in industrial automation and control systems. It is probably the best applicable standard for the automation industry and for a vendor like B&R. Three different vendor types are the target group of this collection:

- Asset owners / end users
- System integrators
- Product suppliers

The standard defines the different responsibilities of the three groups to ensure successful information security throughout the automation process.

The ISA/IEC 62443 series contains 15 elements fitted into 4 different topic groups as the following Figure shows:

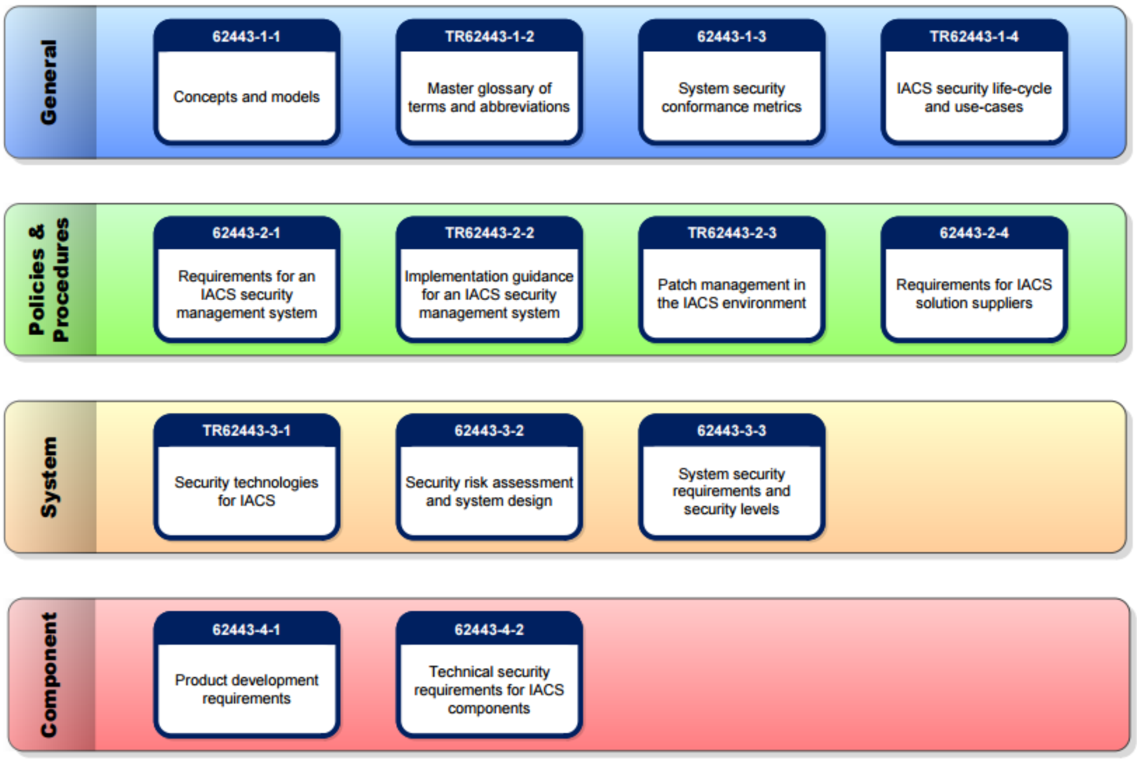

General
62443-1-1
Concepts and models
TR62443-1-2
Master glossary of terms and abbreviations
62443-1-3
System security conformance metrics
TR62443-1-4
IACS security life-cycle and use-cases
Policies & Procedures
62443-2-1
Requirements for an IACS security management system
TR62443-2-2
Implementation guidance for an IACS security management system
TR62443-2-3
Patch management in the IACS environment
62443-2-4
Requirements for IACS solution suppliers
System
TR62443-3-1
Security technologies for IACS
62443-3-2
Security risk assessment and system design
62443-3-3
System security requirements and security levels
Component
62443-4-1
Product development requirements
62443-4-2
Technical security requirements for IACS components

Figure 4 - ISO/IEC 62443 Elements[4]

With B&R as a product supplier, the focus is on ISA/IEC 62443-4-1 "Secure Product Development Lifecycle Requirements" and ISA/IEC 62443-4-2 "Technical Security Requirements for IACS Components".

The first document gives recommendations about how to make products secure by design and how to adapt the engineering process to implement security in every step. In the next chapter 2.3. "Information Security in the Development Lifecycle", a focus on this matter and concepts of ISA/IEC 62443-4-1 will be discussed.

In the second document, tangible security requirements are defined for products that are sold by product suppliers like industrial PCs and PLCs, embedded devices, network components, as well as the application software.
From its perspective 7 key requirements must be met:

- Identification and authentication control
- Use control
- System integrity
- Data confidentiality
- Restricted data flow
- Timely response to events
- Resource availability

Another aspect covered in the ISO/IEC 62443 is the concept of defining security levels for the target system. Asset owners can define the needed protection level of their environment and choose products and processes according to these requirements.
The defined security levels are shown in the following figure:

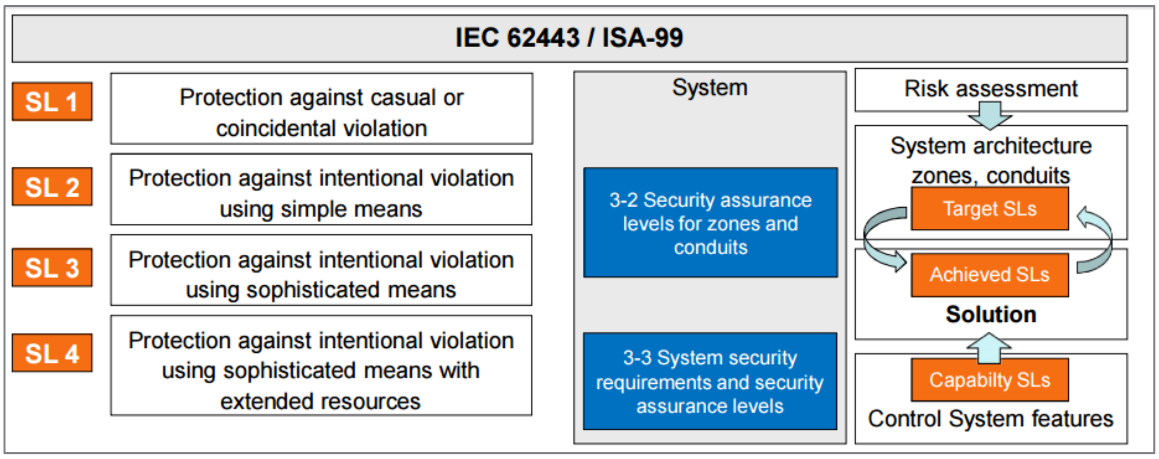

IEC 62443 / ISA-99
SL 1
Protection against casual or coincidental violation
SL 2
Protection against intentional violation using simple means
SL 3
Protection against intentional violation using sophisticated means
SL 4
Protection against intentional violation using sophisticated means with extended resources
System
3-2 Security assurance levels for zones and conduits
3-3 System security requirements and security assurance levels
Risk assessment
System architecture zones, conduits
Target SLs
Achieved SLs
Solution
Capabilty SLs
Control System features

Figure 5 – ISO/IEC 62443 Security Levels[5]

[4] Taken from ISA: The 62443 series of standards: Industrial Automation and Control System Security, 2016, available at http://isa99.isa.org/Public/Information/The-62443-Series-Overview.pdf
[5] Taken from: Kobes P.: Security Levels in ISA-99 / IEC 62443, 2012, available at http://isa99.isa.org/Documents/Committee_Meeting/(2012-05)%20Gaithersburg,%20MD/ISA-99-Security_Levels_Proposal.pdf

Security Level 1 defines a protection level that is suited against unintentional or casual misuse of a given system. Examples for this level can range from inexperienced users entering wrong or just unexpected data into user input interfaces to impatient or power users who are stressing a system with too many requests.

Security Level 2 adds bad intent to the user's actions with the system. The purpose of the interaction with the system is to tamper with the confidentiality, integrity or availability of the system and/or the processed data. At this level the attacker uses only simple means. Simple in this case means that the attacker does not possess deep knowledge of the system or how to attack such a system. The attacker maybe tries to tamper with the system by sending many requests or by tampering with the systems input data. It can be assumed that the attacker can use standard tools like text editors, for example, to edit configuration parameters, but not specific tools in the field of information security.

Security Level 3 gives potential attackers the possibility to use sophisticated methods and special software or hardware to tamper with the given system, assuming that the attacker has experience in analyzing and penetrating these types of systems. Specific software in the field of vulnerability scanning, forensics, stress testing, reverse engineering, cracking and exploitation can be used by the attacker.

Security Level 4 defines that an attacker not only is able to use sophisticated measures as explained with SL3, but also has access to extended resources. These extended resources can be monetary assets, access to vast amounts of computing power or simply enough time to attack the given system. This level also specifically targets the power of domestic intelligence agencies, like the NSA. Since Snowden's revelations in 2013, it is known that such organizations have a wide variety of possibilities to attack any type of system. One example is the possibility to break weak cryptographic ciphers, where even sophisticated attackers are limited due to lack of computing time to break a certain type of hash. Other possibilities are side channel attacks, which need special types of equipment, like probing devices or sophisticated measuring equipment. It can be stated that it is very hard to protect a system against this wide variety of logical, physical and side-channel attacks.

For the European market an ISO/IEC 62443 certification is available from the TÜV SÜD for product suppliers and system integrators. As the standard is developing into a best-practice approach for companies in the automation industry, the certification of such products could lead to an advantage in the market.

2.2.2. NIST Special Publications

The National Institute of Standards and Technology (NIST) provides a vast variety of technical research documents concerning information security.

The three lines of special publications that deal with this matter are:

- SP 800 Computer Security
- SP 1800 Cybersecurity Practice Guides
- SP 500 Computer Systems Technology

Especially in the SP 800 line of publications, some are very fitting to the automation industry, for example NIST SP 800-82r2 Guide to Industrial Control Systems (ICS) Security.

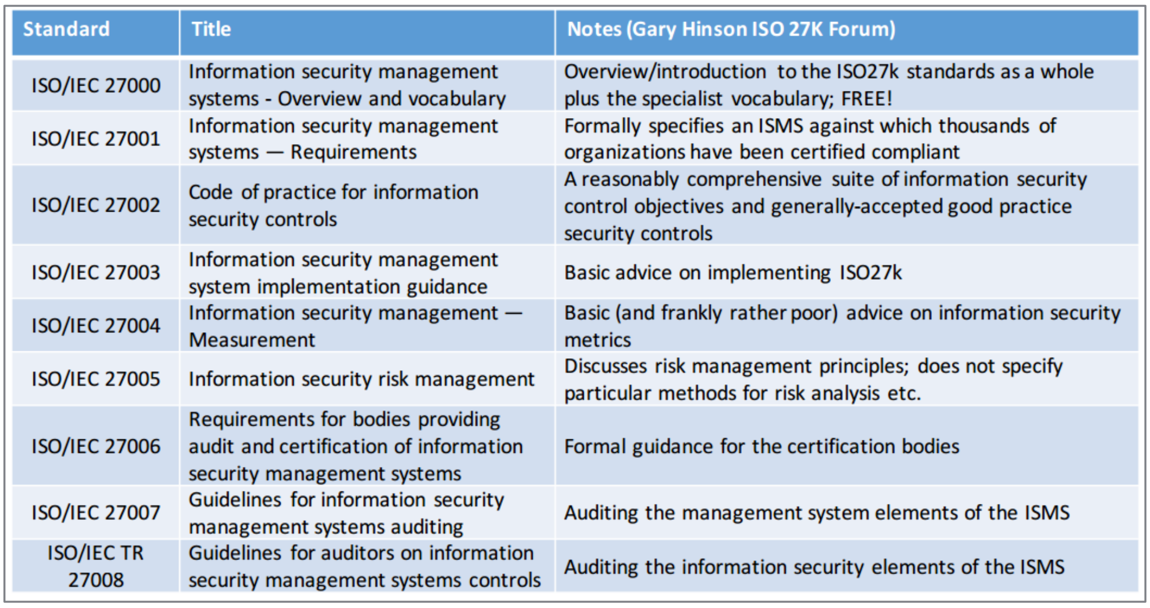
NIST SP 800-53r4 provides an overview of an information security control framework made up of 285 controls in 19 control families. It is mandatory for US federal agencies and their contractors but also applicable to almost every organization.

However, also basic approaches to information security like how to make an awareness training program can be researched in detail (SP 800-50 Building an Information Technology Security Awareness and Training Program). Notable is also NIST SP800-30 Guide for Conducting Risk Assessment as this is still the standard paper regarding information security risk management. This will be discussed in a later topic regarding information security in existing products.

2.2.3. **ISO 27001**

ISO 27001:2013 is an international standard adopted by many organizations worldwide.
It provides guidance on how to implement an Information Security Management System and the necessary security controls.

The following figure shows an overview of the ISO 27000 family:



Standard	Title	Notes (Gary Hinson ISO 27K Forum)
ISO/IEC 27000	Information security management systems - Overview and vocabulary	Overview/introduction to the ISO27k standards as a whole plus the specialist vocabulary; FREE!
ISO/IEC 27001	Information security management systems — Requirements	Formally specifies an ISMS against which thousands of organizations have been certified compliant
ISO/IEC 27002	Code of practice for information security controls	A reasonably comprehensive suite of information security control objectives and generally-accepted good practice security controls
ISO/IEC 27003	Information security management system implementation guidance	Basic advice on implementing ISO27k
ISO/IEC 27004	Information security management — Measurement	Basic (and frankly rather poor) advice on information security metrics
ISO/IEC 27005	Information security risk management	Discusses risk management principles; does not specify particular methods for risk analysis etc.
ISO/IEC 27006	Requirements for bodies providing audit and certification of information security management systems	Formal guidance for the certification bodies
ISO/IEC 27007	Guidelines for information security management systems auditing	Auditing the management system elements of the ISMS
ISO/IEC TR 27008	Guidelines for auditors on information security management systems controls	Auditing the information security elements of the ISMS

Figure 6 - The ISO 27000 Family[6]

ISO/IEC 27002:2013 shows a code of practice for information security controls. This is a reasonably comprehensive suite of information security controls and works well as a best practice when searching for appropriate security controls. The overview is shown in the following figure:

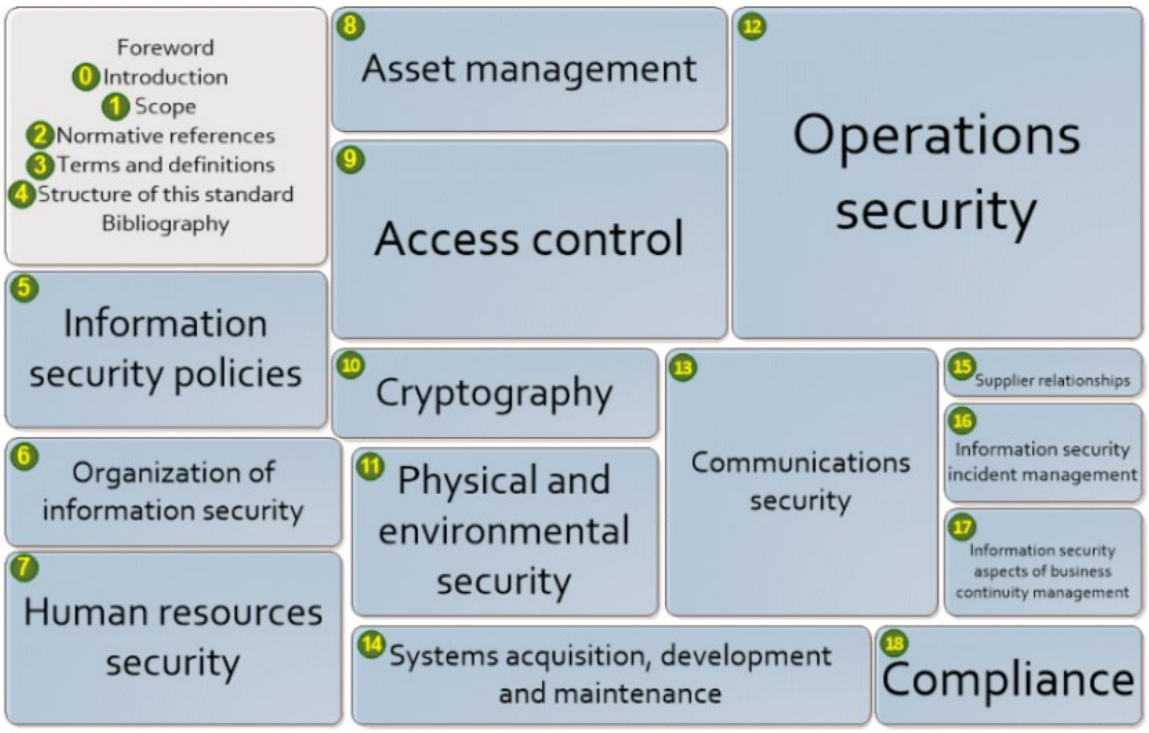

Foreword
0 Introduction
1 Scope
2 Normative references
3 Terms and definitions
4 Structure of this standard
Bibliography
5 Information security policies
6 Organization of information security
7 Human resources security
8 Asset management
9 Access control
10 Cryptography
11 Physical and environmental security
12 Operations security
13 Communications security
14 Systems acquisition, development and maintenance
15 Supplier relationships
16 Information security incident management
17 Information security aspects of business continuity management
18 Compliance

Figure 7 - ISO/IEC 27002:2013 Security Controls[7]

[6] Taken from: Hinson G.: The ISO27k Standards, March 2014
[7] Taken from: ISO 27k Security, 2013, available at http://www.iso27001security.com/html/27002.html

Although the ISO 27000 family is probably one of the most comprehensive resources for an information security program, it must be used with caution. To focus on the essential parts a definition of the objectives scope is one of the most important tasks when applying security controls from this framework.

2.2.4. BSI 100 „IT-Grundschutz“

Since 1994, the German federal office of information security (BSI) has released its own technical manifest called the "IT-Grundschutz" or IT baseline protection. Like the aforementioned frameworks, its goal is to achieve sufficient levels of information security in information technology systems. It can be seen as a shortened, more precise approach to the topic of information security management than the more comprehensive ISO standards.
Nevertheless, the BSI offers a certification for ISO/IEC 27001 based on the IT baseline protection. Therefore, a mapping table of the ISO/IEC 27001 standard to the IT baseline protection is offered.

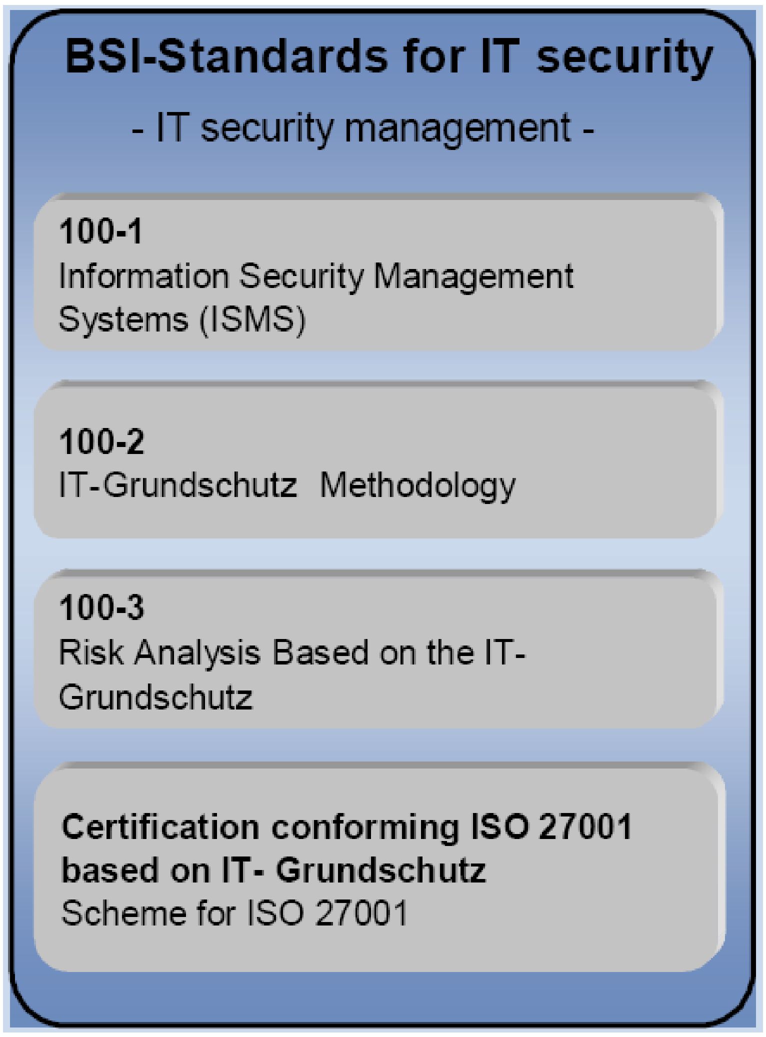

BSI-Standards for IT security
- IT security management -
100-1
Information Security Management Systems (ISMS)
100-2
IT-Grundschutz Methodology
100-3
Risk Analysis Based on the IT-Grundschutz
Certification conforming ISO 27001 based on IT- Grundschutz
Scheme for ISO 27001

Figure 8 - BSI IT Baseline Protection[8]

The BSI additionally released BSI 100-4 Business Continuity Management which focuses on specific actions to minimize damage and business interruption in an organization, if critical situations like attacks happen.

To conquer the rising requirements of information security in the field due to new technologies like cloud computing, Internet of Things and others the BSI is trying to renew the IT baseline protection documentation. These new versions are already available as community drafts and will be released in the near future.

2.3. Information Security in the Development Lifecycle

To ensure that software, which is developed in the future, does not fall into the same pattern of security issues as systems do worldwide today, the commitment to various measures must be agreed upon. The process to produce new software with a better focus on the information security issues will be called "Secure Software Development Lifecycle", in short SSDLC. This is in alignment with current literature and best practices in leading development companies (e.g. Microsoft).

Modern software development can be categorized into different phases starting with the basic product idea to a successfully deployed product. These phases can be executed consecutively, in a waterfall approach or also in an iterative-incremental process.
To look into the different approaches of increasing the security of newly developed software, the sequential process will be used. Afterwards, the adaption of the approaches to the more modern, iterative or agile way of developing software will be discussed.
Many of the applied measures are a general way of ensuring better quality in software products and are not directly linked to information security (though the later follows by the correct measure implementation).

The basic goal is to minimize security-related problems in the design and implementation of a new product as early as possible in its lifecycle.
This principle is already known from the field of software quality engineering. Already back in 1981, Barry Boehm stated in his presentation "Case study: Finding defects earlier yields enormous savings" [28] that the relative cost of fixing software errors increases rapidly the later these errors are found. Subsequent studies [29][30] compared the costs of issues found during an early phase to those of a later phase. And even though these ratios may be exaggerated to some research and some

[8] Taken from: BSI-Standard 100 – Managementsystems for Information Security, 2008, available at https://www.bsi.bund.de/EN/BSI/Publikationen/ITGrundschutzstandards/BSI-Standard_1001.pdf

implications like “requirement errors are the hardest to fix” [31] are not reproducible, the basic rule seems to uphold more than 35 years later.
Therefore, the credo of nearly every modern approach on improving software quality is to find defects as early in the lifecycle of a product as possible.
As many security-related problems have their origin in software defects and many security-related requirements yield in basic design principles, it is only conclusive that the same rules apply to them.

A renowned practice approach is provided by Microsoft with their version of the SSDLC which is simply called Security Development Lifecycle (SDL). [32] Since its first occurrence in 2004 the SDL has evolved into a well-defined methodology. Although in the definitions and practical examples of the SDL the focus lies on Microsoft-specific technology and tools, it can be used as a guideline for other areas as well.

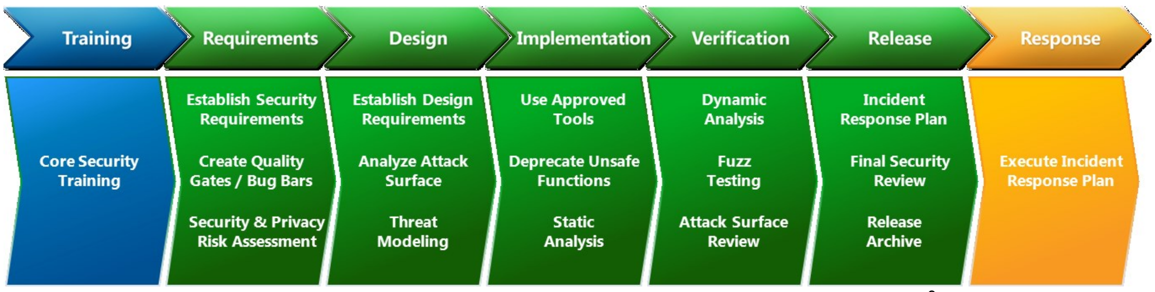


Figure 9 - The Microsoft Security Development Lifecycle - Simplified[9]

Newer standards focusing on securing the developing processes, like the industrial standard ISA/IEC 62443-4-1 “Secure Product Development Lifecycle Requirements”, are mostly in accordance with these recommendations and are heavily influenced by its advices.
There are more general approaches which focus on assessing and evolving secure software initiatives in a given environment like BSIMM [33] (Building Security in Maturity Model) or the OpenSAMM [34] (Open Software Assurance Maturity Model) framework.
Other companies are also following Microsoft in publishing their independent version of secure development lifecycles. One example is Cisco with their “Cisco Secure Development Lifecycle”. [35]
Cisco focuses more on the different software assurance activities and does not categorize them in lifecycle phases like Microsoft, but the activities are roughly comparable.

In the following chapters these possible security measures will be discussed in each phase of the development lifecycle of a product. It is obvious that this is not a replacement for an approved software development process that is already in use, but rather an enhancement to existing development steps.

[9] Taken from: Microsoft Corporation: Simplified Implementation of the Microsoft SDL, 2010, available at https://www.microsoft.com/en-us/download/details.aspx?id=12379

Microsoft puts it this way: “This document describes both required and recommended changes to software development tools and processes. These changes should be integrated into existing software development processes to facilitate best practices and achieve measurably improved security and privacy.” [36]

2.3.1. Training Phase

Basic training is the key to ensure that the necessary awareness of the problem areas, concerning the products’ security, is spread among the key stakeholders. Furthermore, the fundamental concepts for the various activities in the different process phases must be obtained by the different team members.

According to NIST SP800-50, four critical steps are needed to enroll an Awareness and Training Program in a company:[37]

- **“Awareness and Training Program Design:**
  In this step, an agency wide needs assessment is conducted and a training strategy is developed and approved. This strategic planning document identifies implementation tasks to be performed in support of established agency security training goals.
- **Awareness and Training Material Development:**
  This step focuses on available training sources, scope, content and development of training material, including solicitation of contractor assistance if needed.
- **Program Implementation:**
  This step addresses effective communication and roll-out of the awareness and training program. It also addresses options for delivery of awareness and training material (web-based, distance learning, video, on-site, etc.).
- **Post-Implementation:**
  This step gives guidance on keeping the program current and monitoring its effectiveness. Effective feedback methods are described (surveys, focus groups, benchmarking, etc.).”

It is described that the learning process itself is a continuum and, therefore, must be embraced in such a way. The continuum starts with the basic awareness, builds into a training program and matures into education as the following figure points out:

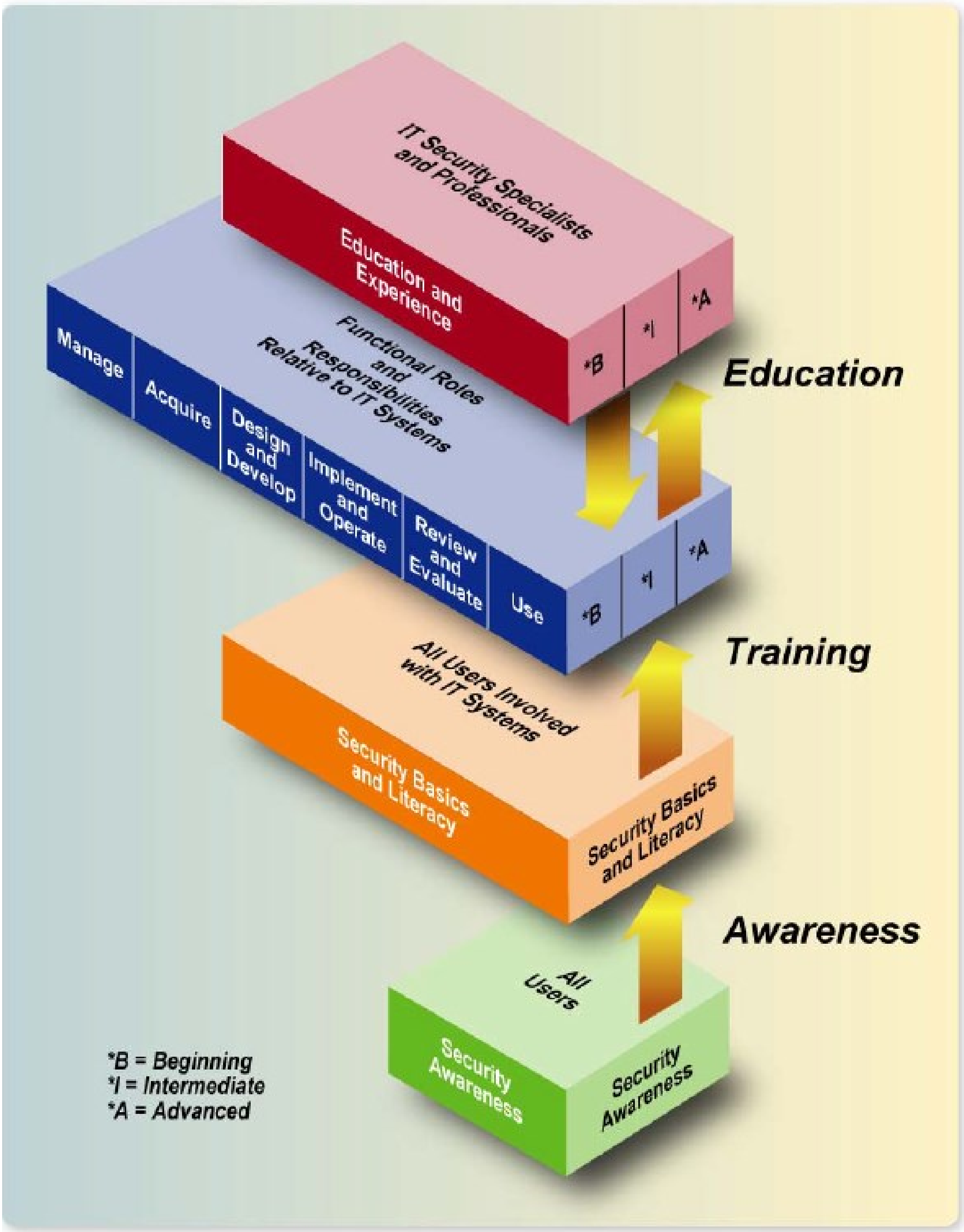


Figure 10 - The IT Security Learning Continuum[10]

Basic topics of a security focused initial training can be:

- **Security Awareness / Motivation Training**
  Developers often lack the proper motivation or principal awareness in security-related problems. Even in modern development processes like

[10] Taken from: Wilson M and Hash J.: NIST SP800-50 Building an Information Technology Security Awareness and Training Program, October 2003, available at http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-50.pdf

Scrum, the performance of developers is measured in how quick a certain feature can be achieved. If security is not directly addressed as a goal in these features it is evident that the developers won’t focus on such issues.
A cultural change in developing companies is often needed and can only be started by addressing this with ensuring the awareness of all development stakeholders.

- **Secure Architecture and Design (e.g. Security Patterns)**

Software engineers and architects need to apply secure design principles when drafting the foundations of new products. This can include topics like defense in depth, secure defaults, attack surface reduction, least privilege principles, etc. and will be discussed in the “Design Phase” chapter.

- **Threat Modelling**

An explicit training should focus on the topic of threat modelling, which will be introduced in the “Design Phase” chapter. The training itself should not only be attended by the engineer(s) who will conduct the threat modelling process, but also by programmers and testers, because it is also necessary to know how to code or test to a threat model.

- **Secure Coding**

Developers should be introduced to the principle of writing secure code. This can include topics like buffer overruns, heap overflows, arithmetic number errors, SQL injection, pointer errors, etc. and will be discussed in detail in the “Implementation Phase” chapter. There are also many well-known books in this sector which can be recommended. [38][39]

- **Basics of Cryptography**

Cryptography is the key element to protect the confidentiality and integrity of data in an untrusted environment. All engineers in the development process should know the basic principles of secret-key, public-key and hash functions and in which use cases to apply those different types of cryptography.

- **Security Testing**

Various methodologies exist to test and verify a program and the underlying code against security requirements. The responsible stakeholders must understand the special characteristics of security testing and the difference to functional testing. This also includes the ethnical factor as in particular penetration testing often requires a “blackhat” approach, where the tester seeks to apply destructive measures as a malicious adversary would do.
This will be discussed in the “Verification Phase” chapter.

- **Risk Management**

Risk management in general and risk assessment in detail are topics which have to be considered when trying to achieve a certain security goal. As no modern product with all kinds of interfaces (regardless of hardware or software) can achieve “full” security, it is vital to know how to categorize and manage possible security risks. As important as this subject is while developing a new project, it gets even more crucial when looking at existing products. Therefore, it will be discussed in detail how to manage this risk in chapter 2.4. “Handling Information Security in Existing Software”.

- **Privacy**
  Privacy should be a dedicated topic when looking at information security. The concepts of different types of privacy data and practices on how to handle it should be discussed.
- **Usage of Tools for the Different Process Phases**
  There are many programs and tools (software and hardware) which can help in the different phases to achieve the proposed security goal or measure. Showing the different stakeholders that these tools can help them in their work is a key element to help accepting security measures as a necessary and crucial element in the development process. Examples of these tools will be given in the different phases, especially in the implementation phase.

Though the training phase is located at the start of the development lifecycle, it is clear that new events, like e.g. changes in the security of cryptographic algorithms, can always happen. It is recommended to provide up-to-date information on an ongoing basis to all stakeholders and renew parts of the training program even during other phases of the lifecycle if necessary.

2.3.2. Requirement Phase

When trying to implement security measures, the academic community has a different understanding of the requirement phase. Literature for requirement engineers, business analysts, product owners, product managers, etc. often states that the focus of requirements engineering has to be the product itself or that the approach should be solely feature-based and that no attention should be given to technicalities. When confronted with information security requirements, the professionals responsible for the requirements engineering process usually argue that security is a technical topic and, therefore, has to be processed in a later phase.
However, the overwhelming majority of research topics add dedicated security requirements in the requirements phase when introducing a secure software development lifecycle.

2.3.2.1. Establish Security Requirements

Typically (but not necessarily), security requirements are non-functional requirements (NFRs) and can be distinguished into two different classes: one-time and continuous.
While one-time requirements can be treated like other requirements, continuous requirements are best implemented in a “Definition of Done” or a Quality-Gate to ensure that no implemented feature lacks the continuous requirement. Security requirements can also be focused on privacy issues of the end user.

There are many models in the field which can help defining security requirements. These modelling approaches are already introducing elements from the design phase, but help, of course, with finding the underlying security requirements.

A few examples are:

- Information Security and Safety Models by Firesmith [40]
- Attack Trees [41]
- Fault Trees [42]
- CORAS Risk Modelling Notation [43]
- Secure Tropos [44]
- The SQUARE Methodology

The last example (SQUARE) will be explained at the end of this chapter. More design focused approaches will be discussed in the following chapter 2.3.3. “Design Phase”.

A comprehensive categorization list of Information Security Requirements was published by Firesmith: [45]

- Identification Requirements
- Authentication Requirements
- Authorization Requirements
- Immunity Requirements
- Integrity Requirements
- Intrusion Detection Requirements
- Nonrepudiation Requirements
- Privacy Requirements
- Security Auditing Requirements
- Survivability Requirements
- Physical Protection Requirements
- System Maintenance Security Requirements

Security requirements are essential security in the development process. If the security requirements are well-defined in this early stage of a product, the chances of achieving the necessary security for the product are significantly higher.

2.3.2.2. Establish Quality Gates / Bug Bars

At the start of a project, a minimum level of quality must be defined for each release (of components, features or the product itself). Typically this quality gate centers around which type of bug must be closed (e.g. all above a certain level) and what conventions (e.g. code criteria) must be fulfilled.
Potential examples are:

- Every requested requirement must be met and tested
- Architectural design is documented
- Unit Test Code Coverage above percentage value (e.g. branch coverage 90%)

- Static code analysis values are met (e.g. maximum cyclomatic code complexity is not higher than 10)
- Build must be warning free
- User Documentation is updated

While it is vital that these quality-centered requirements are pursued, additional security-related topics can be added. A few examples are:

- Threat model documented
- Debug interfaces closed (at least in the Release Build)
- Minimum levels of privacy and security requirements are met
- No known vulnerabilities

2.3.2.3. Security / Privacy Risk Assessments

In the requirement phase it is mandatory to find out which privacy and security risks are to be expected when looking at the requirements of the product.
The security risk assessment is the primary foundation to find out where more comprehensive threat modeling or security design analysis has to be done in the design phase or which part needs specific fuzz testing requirements.
The privacy risk assessment shows what privacy impact rating is applicable. Microsoft SDL provides a classification in three categories: [36]

- P1 High Privacy Risk. The feature, product, or service stores or transfers PII, changes settings or file type associations, or installs software.
- P2 Moderate Privacy Risk. The sole behavior that affects privacy in the feature, product, or service is a one-time, user-initiated, anonymous data transfer (for example, the user clicks on a link and the software goes out to a Web site).
- P3 Low Privacy Risk. No behaviors exist within the feature, product, or service that affect privacy. No anonymous or personal data is transferred, no PII is stored on the machine, no settings are changed on the user's behalf, and no software is installed.

2.3.2.4. SQUARE

A more structural approach to the process of security requirements engineering is proposed by the Software Engineering Institute’s Networked Systems Survivability (NSS) Program at Carnegie Mellon University.
There the Security Quality Requirements Engineering (SQUARE) methodology was developed. It consists of nine steps to ensure a comprehensive method when focusing on security requirements:

- Agree on Definitions
- Identify Security Goals
- Develop Artifacts
- Perform Risk Assessment

- Select Elicitation Technique
- Elicit Security Requirements
- Categorize Requirements
- Prioritize Requirements
- Requirements Inspections

Each step of the methodology has its defined input and output, the techniques to be used and the participants who have to perform the step. This approach may sound labor-intensive at the requirement phase, though it generally makes sense to invest resources at the beginning of a project. Nevertheless, a lighter version called SQUARE-Lite was also developed.
The US-Cert lists the SQUARE technique as their recommendation for requirements engineering and also (in a special form) as a method for acquisition where identical problems occur. [46]

2.3.3. Design Phase

The design phase is a very crucial step for the outcome of a systems security. Important decisions have to be made about the architecture and best practices on which design principles to apply. Analyzing a design with threat modeling and attack surface analysis can show possible problems, like exposed modules which may need special design treatments.

2.3.3.1. Threat Modelling

Threat modelling is one of the most common ways to improve the security of a system before or while implementing it.
It is implicitly done in many engineering disciplines. For example, when an engineer is planning to build a new house he has to think of the problems that may occur, like natural events (rain, snow, flooding) or burglaries. Depending on the threats, appropriate measures like a stronger roof, drainage equipment or special locks have to be installed.

Threat modelling gives a development team a structured guide on how to discuss possible security weaknesses of the modelled design.
Within this chapter a short introduction on how threat modelling works is given, largely based on Adam Shostacks book “Threat Modelling: Designing for Security”. [47]

As the following figure shows, there are 4 consecutive tasks when trying to do threat modelling with a software system:

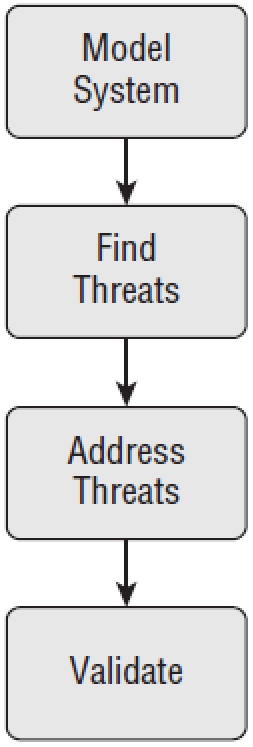


Figure 11 - The Threat Modelling Process[11]

In each of the process steps one fundamental question can be focused on:

- Model System: What are you building?
- Find Threats: What can go wrong?
- Address Threats: What should you do about those things that can go wrong?
- Validate: Did you do a decent job of analysis?

Model System:
Modelling the underlying system is the first step of the threat analysis. Engineers have to find an abstract way of thinking about the system they are trying to design and implement. The easiest way for humans to achieve this is drawing diagrams. There are different types of diagrams suitable for this task. First off and most frequently used are Data Flow Diagrams (DFDs). Vulnerabilities tend to open where data is flowing, especially when it is data input by the user.
Also very common are UML diagrams, mainly structure diagrams, state diagrams, interaction diagrams or behavior diagrams. UML is the de facto standard for modelling in software projects, although the complexity behind it is fairly high. There are even specific notations like UMLsec or SecureUML for noting information security advisories within regular UML diagrams or for model-driven development of secure systems, respectively.
The third common diagram type is the Swim Lane Diagram. SLDs also show data flow between communicating parties.

[11] Taken from: Shostack A. – Threat Modelling: Designing for Security, 2014, John Wiley & Sons.

In the end, what matters is not which type of diagram is used, if all the contributing people receive a better understanding of the models behind the project's design.

As a general rule on what to include in a diagram, the following comprehensive list from [38] can be followed as a guideline:

- Show the events that drive the system.
- Show the processes that are driven.
- Determine what responses each process will generate and send.
- Identify data sources for each request and response.
- Identify the recipient of each response.
- Ignore the inner workings, focus on scope.
- Ask if something will help you think about what goes wrong, or what will help you find threats.

Finding Threats:
When the threat modelling participants have a clear understanding of the underlying model, the actual search for threats can begin in the second part of the process. There are four different methodologies on how to find threats, each with its own merits and drawbacks:

- The STRIDE mnemonic
- Attack Trees
- Attack Libraries
- Privacy Tools

STRIDE is a mnemonic for modelling security threats. It divides the threats in six categories:

- Spoofing of user identity
- Tampering
- Repudiation
- Information disclosure (privacy breach or data leak)
- Denial of service (D.o.S)
- Elevation of privilege

The framework helps in finding threats by thinking about how the system design is susceptible to each of the threat types. When checking the model, each STRIDE category is evaluated and if there is a threat possibility, it is written down. It is imperative that the categorization itself is not important, since it is very easy to find threats that are hard to classify into one specific STRIDE category. In essence, it is important that threats are found, but not that they are classified.

Modelled tree diagrams, so called attack trees, are another possibility of detecting threats. The root represents the goal (e.g. a malicious attack) and the branches representing the way of attacking the system, or as Schneier puts it:
"Attack trees provide a formal, methodical way of describing the security of systems, based on varying attacks. Basically, you represent attacks against a

system in a tree structure, with the goal as the root node and different ways of achieving that goal as leaf nodes." [41]
An example of an attack tree is shown in the following figure:

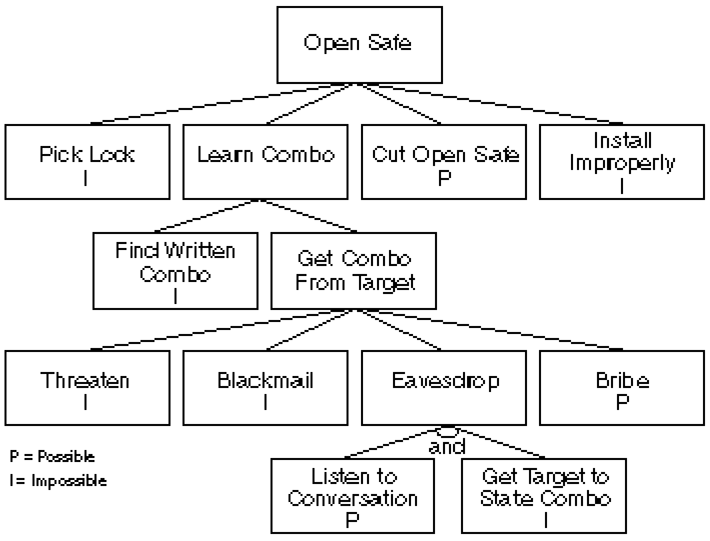


Figure 12 – Example Attack Tree[12]

Attack libraries consist of detailed lists of practical attacks or tools for attack. Threats are then derived from thinking of how these practical attacks could work on the given system. Other threat libraries work the same as STRIDE, providing only a short list of security risks for the application. A good example is the OWASP Top Ten List [48], showing the ten most common security risks for web applications:

- A1-Injection
- A2-Broken Authentication and Session Management
- A3-Cross-Site Scripting (XSS)
- A4-Insecure Direct Object References
- A5-Security Misconfiguration
- A6-Sensitive Data Exposure
- A7-Missing Function Level Access Control
- A8-Cross-Site Request Forgery (CSRF)
- A9-Using Components with Known Vulnerabilities
- A10-Unvalidated Redirects and Forwards

Privacy tools focus explicitly on finding privacy risks in the underlying models. Like with the STRIDE-mnemonic there is a mnemonic for privacy threat modelling:
LINDDUN, which stands for the following privacy issues:

- Linkability
- Identifiability
- Non-Repudiation
- Detectability

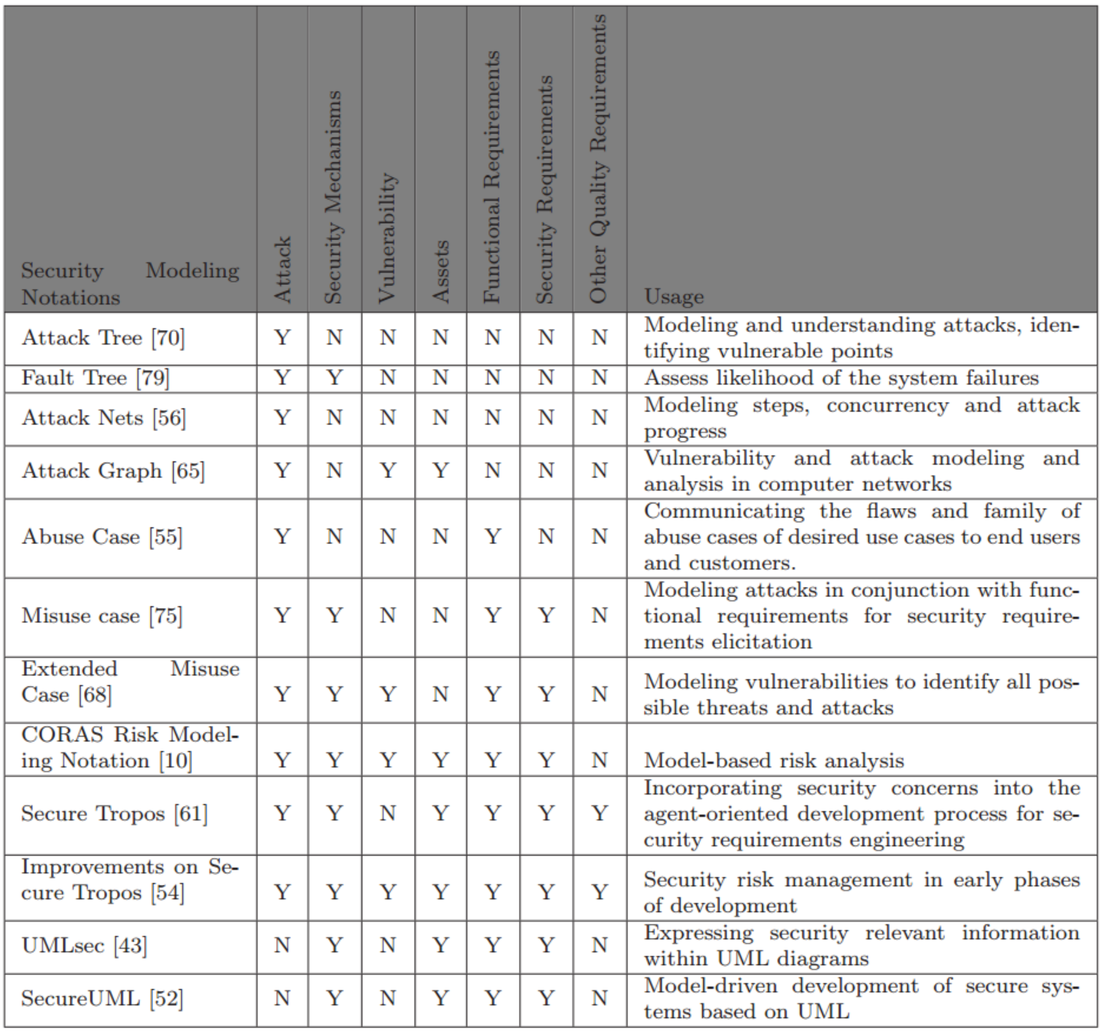
[12] Taken from: Schneier B. – Dr. Dobb's Journal: Attack trees, December 1999, available at https://www.schneier.com/academic/archives/1999/12/attack_trees.html

- Disclosure of information
- Content unawareness
- Policy and consent noncompliance

Address Threats:
In the third part of the process, the found threats are analyzed and mitigation possibilities are researched.
Most of the time, it will be technical implementation of a defensive pattern or features like encryption, authentication, integrity checking, etc., to address a threat. It could also be a simple configuration or usage of another protocol, or tunneling through a secure protocol to make a threat entirely disappear or minimalize it to a certain level.
Keep in mind that often the best way to address threats is to use standardized, security audited products, as self-made solutions naturally tend to be unstable in first iterations.
The business perspective of addressing threats is risk management and will be discussed in a later topic.

Validate:
After addressing the threats with various new designs, better security technology and special features, the most important thing is to validate the found solutions. It would not be the first time that a minimal problem was fixed with a code-heavy feature leading to other threats and vulnerabilities. In essence, like every other step in the development process, some sort of quality assurance has to be applied to the changes made by threat modelling.

There are many other security modelling notations out in the field which can help to find suitable threats and vulnerabilities in systems. The following figure from Elahi shows a comprehensive list of notation with the indication of whether a security concept is considered or not: (Y indicates that the concept is considered, N that it is not)



Security Modeling Notations	Attack	Security Mechanisms	Vulnerability	Assets	Functional Requirements	Security Requirements	Other Quality Requirements	Usage
Attack Tree [70]	Y	N	N	N	N	N	N	Modeling and understanding attacks, identifying vulnerable points
Fault Tree [79]	Y	Y	N	N	N	N	N	Assess likelihood of the system failures
Attack Nets [56]	Y	N	N	N	N	N	N	Modeling steps, concurrency and attack progress
Attack Graph [65]	Y	N	Y	Y	N	N	N	Vulnerability and attack modeling and analysis in computer networks
Abuse Case [55]	Y	N	N	N	Y	N	N	Communicating the flaws and family of abuse cases of desired use cases to end users and customers.
Misuse case [75]	Y	Y	N	N	Y	Y	N	Modeling attacks in conjunction with functional requirements for security requirements elicitation
Extended Misuse Case [68]	Y	Y	Y	N	Y	Y	N	Modeling vulnerabilities to identify all possible threats and attacks
CORAS Risk Modeling Notation [10]	Y	Y	Y	Y	Y	Y	N	Model-based risk analysis
Secure Tropos [61]	Y	Y	N	Y	Y	Y	Y	Incorporating security concerns into the agent-oriented development process for security requirements engineering
Improvements on Secure Tropos [54]	Y	Y	Y	Y	Y	Y	Y	Security risk management in early phases of development
UMLsec [43]	N	Y	N	Y	Y	Y	N	Expressing security relevant information within UML diagrams
SecureUML [52]	N	Y	N	Y	Y	Y	N	Model-driven development of secure systems based on UML

Figure 13 - Comparison of Security Modeling Notations[13]

“Existing modeling notations for security address different security concepts and take different viewpoints to security. Each modeling approach is able to express certain aspects and lacks conceptual modeling constructs to represent some other.” [49]
Therefore, when searching for a security modelling notation, one has to consider which concepts are important to apply in the specific use case. [50]

2.3.3.2. **Security Design Requirements**

When designing the architecture of a new system, fundamental security design principles have to be applied.

[13] Taken from: Elahi G.: Security Requirements Engineering: State of the Art and Practice and Challenges, available at http://www.cs.toronto.edu/~gelahi/DepthPaper.pdf

Many of the design patterns, which are explained in the following chapters, go back to Saltzer and Schroeder's original 1975 design principles. [51] Apparently, most of them have stood the test of time and are either merely adapted or enhanced to fit the modern world of software development.

An example list of these patterns could be: [52][53][54]

- **Economy of mechanism**
  Maybe the most important lesson a security architect can teach is to keep things simple. The more complex a system or software application gets, the harder it is to keep it secure. Many other principles are based on this assumption or are aimed in the same direction. An example would be KISS (Keep It Simple, Stupid) or YAGNI (You Aren't Gonna Need It). Another characteristic of economy of mechanism is that the security functionality itself should also follow this rule of being as simple as possible. Otherwise the security functionality itself could become a security problem.
- **Least privilege principle**
  A program, user or other entity should only be given the privileges it needs for fulfilling the task it was created for. If privileges have to be raised for a special purpose, this should only be temporary. Example resources where privileges could be applied are user rights, file system permissions or hardware resources like CPU power, memory usage or network bandwidth.
- **Separation of privilege**
  The separation of privilege or also sometimes called separation of duty pattern states that not only one condition should enable access to a resource. If multiple conditions (which may even differ in the type of the condition) have to be granted it is harder for an attacker to circumvent the access system. A modern example of this is multi-factor authentication, where a user has to present at least two separate types of evidence (knowledge, possession or inherence) to an authentication system.
- **Fail-safe defaults**
  Every access to an important resource should be restricted by default. To open access to a resource the authorization system has to identify explicit permissions (rather the opposite way with exclusions). In more modern systems this is sometimes referred to as "Deny by Default" and a modern configuration of fail-safe defaults is often implemented as a white-list (in contrary to a blacklist).
- **Complete mediation**
  This principle states that every access to a critical resource has to be validated by the underlying system, e.g. by an authorization mechanism. These access controls should not be weakened by caching, as access rights could be revoked. Of course, this can have a direct impact on the performance of the system.
- **Open design**
  The well-known term "security through obscurity" follows the basic idea that the security of a system is ensured by keeping its functionality and architecture secret, however in the modern world this is no longer a valid option. The open design principle follows exactly the opposite goal by making the design publicly available.

Therefore, different kinds of engineers can peer-review this public data and help in finding security problems. In the field of cryptography this concept goes back to the 19th century with Kerckhoffs's principle. It states that "a cryptosystem should be secure even if everything about the system, except the key, is public knowledge". [55]

- **Least common mechanism**

  This security principle states that functions that enable sharing resources should be minimized or even eliminated, if possible. This reduces deadlock scenarios and also the risk that information paths are crossed between users.
- **Psychological acceptability**

  A security mechanism is often a direct rival of usability. For example, users do not want to be asked for a password every time they access a system. It would be more convenient for them to never see a security-related prompt like a login-field. Therefore, it is very hard to design a security mechanism in a way that users with low awareness of security-necessities are going to use it properly. This is the goal of the psychological acceptability secure design principle. Some modern approaches restate this mechanism as "Least astonishment principle", where the user should not be astonished by the behavior of the system.
- **Isolation**

  The basic idea of isolation is to divide critical data, programs or systems into small pieces and separate them from each other. If one system is compromised then the other pieces are still unaffected. In practice this design pattern is very hard to fulfill, as modern systems tend to be interconnected more and more. A good example is the virtualization technology, where thin layers of control programs called hypervisors are used to isolate other components of the running environment.
- **Encapsulation**

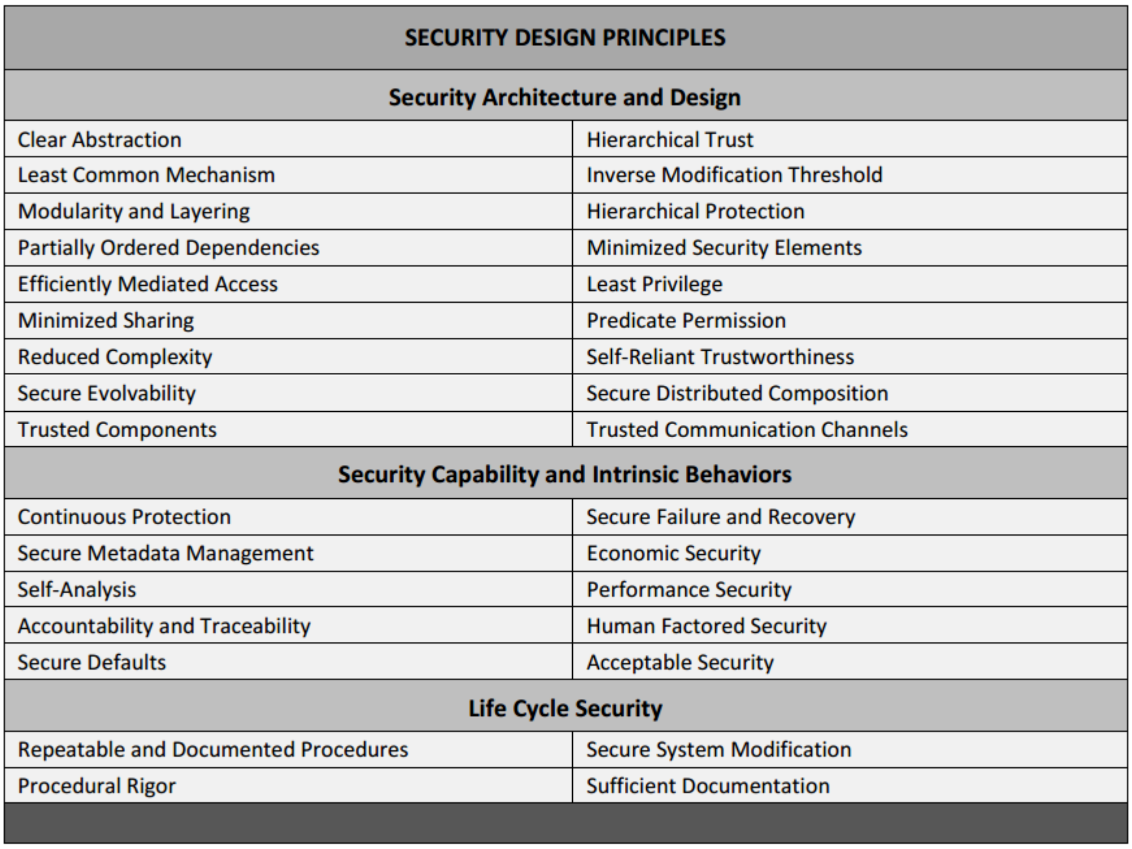
  Encapsulation, a reference to the object-oriented programming language mechanism of restricting direct access to a member of an object is also used as a secure design principle. In general data should be hidden and not be made directly accessible, but only with functions or procedures returning the value.
- **Modularity**

  This design pattern follows the "separation of concerns" approach and ensures that a program or system is designed in a modular way. Therefore, also security functions can be separated into protected modules and used by many other parts of the system. A good example could be cryptographic libraries embedded into a security module, so that all other components can use the cryptographic functions of these libraries.
- **Defense in Depth**

  Defense in depth is also often called layering. It uses overlapping security principles which make it exponentially harder for attackers to penetrate a system, as the attacker needs to accomplish multiple types of successful attacks.

NIST SP800-160 "Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy SFecure Systems" [56] provides a more conclusive approach, listing 32 design principles in the three categories "Security

Architecture and Design", "Security Capability and Intrinsic Behaviors" and "Life Cycle Security". The following figure shows an overview:



SECURITY DESIGN PRINCIPLES	
Security Architecture and Design	
Clear Abstraction	Hierarchical Trust
Least Common Mechanism	Inverse Modification Threshold
Modularity and Layering	Hierarchical Protection
Partially Ordered Dependencies	Minimized Security Elements
Efficiently Mediated Access	Least Privilege
Minimized Sharing	Predicate Permission
Reduced Complexity	Self-Reliant Trustworthiness
Secure Evolvability	Secure Distributed Composition
Trusted Components	Trusted Communication Channels
Security Capability and Intrinsic Behaviors	
Continuous Protection	Secure Failure and Recovery
Secure Metadata Management	Economic Security
Self-Analysis	Performance Security
Accountability and Traceability	Human Factored Security
Secure Defaults	Acceptable Security
Life Cycle Security	
Repeatable and Documented Procedures	Secure System Modification
Procedural Rigor	Sufficient Documentation

Figure 14 - Security Design Principles[14]

It is very hard to apply all these secure design principles to their full extent in a system and not every system needs the usage of all design principles. Security design requirements often directly oppose performance or usability requirements. System architects have to balance these requirements to ensure an acceptable user experience as well as a secure system.

2.3.3.3. Attack Surface Analysis

Attack surface analysis is closely related to threat modelling according to OWASP:
"There is a recursive relationship between Attack Surface Analysis and Application Threat Modeling: changes to the Attack Surface should trigger threat modeling, and threat modeling helps you to understand the Attack Surface of the application." [57]
In the attack surface analysis the observed design model is examined to find opportunities for potential attackers to exploit vulnerabilities.

The attack surface of an application is in short:

- "the sum of all paths for data/commands into and out of the application, and

[14] Taken from: Ross R. et al: NIST SP800-160: Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems, November 2016, available at: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160.pdf

- the code that protects these paths (including resource connection and authentication, authorization, activity logging, data validation and encoding), and
- all valuable data used in the application, including secrets and keys, intellectual property, critical business data, personal data and PII, and
- the code that protects these data (including encryption and checksums, access auditing, and data integrity and operational security controls)" [58]

The goal is to reduce the attack surface by design. This is, for example, possible by restricting access or shutting off specific (maybe unneeded) services. If a general shutdown is not possible, turning certain features off in a default configuration could be viable. In specific use cases these services can be activated again, but the attack surface will be reduced for a majority of users. If the attack surface analysis finds exposed parts, which cannot be changed by design, a defense-in-depth approach can help in mitigating possible attacks.
There are theoretical models to measure the attack surfaces of programs, like the RASQ approach, as indicated by Howard et al in [59].
More practical analysis is done by tools like Attack Surface Analyzer, but as these tools measure the attack surface of finalized software rather than the potential attack surface of a design, this topic will be discussed in the Verification Phase chapter.

2.3.3.4. Usage of Cryptography

Cryptography is the art of ensuring privacy while communicating via an untrusted medium. Four primary goals can be achieved with cryptography:
- Confidentiality: it is ensured that only the intended receiver of a sent message can read the message
- Integrity: a receiver can verify if a sent message has been altered in transit
- Authenticity: the identity of the receiver or sender can be proved
- Non-repudiation: the sender cannot deny sending the message

The tricky part is, that system designers have to decide in which use case which type of cryptography should be applied. The three types of cryptography are: [60]
- **Secret-key cryptography:**
  The same key is used for encryption and decryption of a message. Therefore, the key has to be kept secret and transferred securely to the receiver. Also a unique key is required for every message partner. This is often called symmetric-key cryptography.
- **Public-key cryptography:**
  Two different keys are used for encrypting and decrypting the message. For example a public key can be used for encrypting a message. Only the holder of the matching private key can decrypt the message. Therefore, the public key can be published via insecure channels, only the private key has to be kept secure. The key-pair can be used for infinite communication partners. This is also called asymmetric-key cryptography.

- **Hash functions**
  Hash functions are algorithms which map data of arbitrary size into a bit string of fixed size. They are designed as a one-way function, which are infeasible to revert. These algorithms are often used to ensure data integrity.

A multitude of good practices when designing and implementing cryptographic functions can be found. For reasons of consistency, the implementation examples will also be discussed in this chapter: [61]

- For symmetric-key cryptography the Advanced Encryption Standard (AES) cipher should be used. This is due to the large effort of reviews this cipher has endured since its status as a standard cipher without significantly weakening it. Hardware acceleration is also available on many platforms when this cipher is used and the library supports it.
- A minimum of 128bit of key length has to be used when applying a symmetric cipher.
- RSA is recommended for asymmetric cryptography in a finite field. More modern elliptic curve ciphers can be used, especially in low-performance environments.
- A minimum of 2048bit of key length has to be used when applying an asymmetric cipher in a finite field. For elliptic curve cryptography a minimum of 256bit of key length should be selected.
- When using elliptic curve cryptography appropriate curves have to be selected. There are many standards which try to help in this matter. A good online resource which summarizes the academic coverage in this field is: https://safecurves.cr.yp.to/.
- For hashing purposes, the Secure Hashing Algorithm (SHA) version 2 or 3 should be used. (e.g. SHA-256)
- When implementing SSL/TLS a cryptographically secure version of the protocol has to be used. Currently SSLv2 and SSLv3 have to be omitted, TLSv1.2 is recommended. Ciphers supporting forward secrecy should be used (ECDHE or DHE).
- When using X.509 certificates always check the validity period, verify the hostname or IP address against the Subject Alternative Name or Common Name field as a client, use certificate revocation mechanisms like Certificate Revocation Lists (CRLs), the Online Certificate Status Protocol (OCSP) and OCSP stapling.
- Never implement a cryptographic algorithm on your own, except for educational purposes.

2.3.4. Implementation Phase

During the implementation phase the architecture and plan of the design phase have to be developed into a working product. Best practices in secure coding and knowledge of supporting tools are the key factor in guaranteeing a high software quality and lack of issues that could lead to security or privacy concerns.

2.3.4.1. Secure Coding Practices

To minimize the chance of developing code with security issues, there are, in addition to the more general design principles, several practical coding practices, which should be followed by the software developers.

In the following a few examples are provided: [38][62]

- **Managed code**
  In general it is advisable to use managed code whenever possible. The advantages in e.g. handling memory management, thread management or garbage collection are helping to reduce errors like pointer mess-ups. Type concepts like delegates improve type safety by guaranteeing a reference to a method with a correct signature and thus should be used instead of function pointers. [63]
- **Exception handling**
  Exception handling is a powerful construct to handle run-time errors in a program. Many security attacks have the direct goal to reach a failure-state in a program. Exception handling gives the programmer the chance to separate the error handling code from the productive code, which can also result in security problems if the error-handling succumbs to "catch-all"-handling.
  There is also the possibility to differ the type of error which occurred. This is very important, as programmers should not use global exception handlers, as this could mask attacks. Also programmers need to be careful when exceptions leave objects in inconsistent states, as this could be exploitable by attackers.
  It is also vital to handle exception or error messages in a way that an attacker cannot gain advantage by reading them. Detailed information, like version numbering, stack trace information or other diagnostic information can lead to new attack vectors for a malicious attacker.
- **Validate user input**
  User input is a key attack vector when analyzing attacks in the past years. Attacks like SQL Injection or Cross-Site Scripting are very popular and could be mitigated quite easily. A key prevention measure is to validate the input based on a white-list (accept only known input) or other form of input sanitization (e.g. data range, length, types, character-set, etc.). For standardized query languages like SQL more abstract features like prepared statements or stored procedures can also mitigate this type of attack. [64]
  To test if the implemented mitigations work it is common to use fuzz testers generating random, invalid or unexpected data for the user input. This will be discussed in the following chapter "Verification Phase".
- **Use validated parsers**
  When using standard types of markup language (e.g. XML or JSON) to transfer or store data it is vital to use security-validated parsers. Parsing complex file types like XML is very error-prone and security issues are likely to occur. [65] This applies also to other forms of parsers (e.g. other file-formats like images or regular

expressions). Self-implementation of any form of parser should be omitted. If this is not possible, the parsing algorithm should be extensively tested, e.g. with a fuzz tester.

- **Use existing libraries and APIs**
  It is always better to use an existing library or functionality than to implement the same feature again. Standard libraries are in most cases better tested and reviewed than code which is only implemented for a specific product.
- **Integer arithmetic**
  Overflows or Underflows in numeric variable types are very common mistakes (in non-managed code) and often lead to security issues. A general mechanism to check boundaries should be in place.
- **Explicit initialization**
  Variables and other data storing types should always be initialized explicitly. In most languages this can be done directly when declaring the variable, otherwise it has to be done before the first use of the variable.
- **Secure library loading**
  When loading dynamically linked libraries a program has to be sure to load the correct file. If an attacker gets access to a particular folder where the library is stored or manipulates the path to the library the program could load potentially malicious code. [66]

These practices are often accompanied by special tools, which can check or verify, whether the specific practice is followed correctly. A few examples will be given in the following chapter “Tool Usage”.

A more comprehensive list of secure coding guidelines, with the focus on web application development is provided by the OWASP Foundation. The list contains over 100 practical advices, divided into 14 categories: [67]

- Input Validation
- Output Encoding
- Authentication and Password Management
- Session Management
- Access Control
- Cryptographic Practices
- Error Handling and Logging
- Data Protection
- Communication Security
- System Configuration
- Database Security
- File Management
- Memory Management
- General Coding Practices

The advices often focus on secure design guidelines and the practical implementation of the design pattern, e.g. how to implement the least privilege principle or fail-safe defaults into a user access control mechanism. A detailed explanation of every advice will be omitted, as this is not within the scope of this thesis.

2.3.4.2. Tool Usage

Developers should use a list of approved tools and special settings to ensure that their code follows security regulations. This can be specialized programs which can find buffer overflows in unmanaged code or simple compiler/linker flags to do security checks. A style checker should be used, so no deprecated functions are used. These tools also often help with analyzing the code which leads to the following topic of static code analysis.
An example set could be:

- When using a compiler like GCC, turn on all warnings, enable buffer overflow protection (fstack), Adress Space Layout Randomization (ASLR), overflow traps and read-only relocation. Compiling for windows may be more suitable with visual studio to enable windows-specific protections not available in GCC (for example Structured Exception Handling Overwrite Protection – SEHOP).
- Linkers should also be configured to use features like ASLR or Data Execution Prevention (NX-bit)
- For windows environments BinSkim is a binary scanner that scans windows executables for correct security settings (e.g. if the above compiler and linker settings were made)
- Tools concerning the analysis of source code will be discussed in the following chapter.

2.3.4.3. Static Code Analysis

Static source code analysis is a standard approach in improving code quality, thus also helping to reduce problems which cause security issues. It is performed (contrary to dynamic code analysis) without executing the code and only based on analyzing the text sources or compiler output.
Static code analyzers are able to detect severe coding errors like buffer overflows, different kinds of memory corruption or null pointer dereferencing. Other possibilities are to check if certain coding conventions were followed, if there are code duplicates, if the code includes comments and documentation or if it is complex (which is measureable by different attributes).
Therefore, static code analysis includes the calculation of code metrics like Lines of Code, Code Complexity (Halstead, McCabe, Fat, Maintainability Index, etc.) and object-oriented metrics (Depth of Inheritance, Coupling, Cohesion, etc.). These metrics can be used to statistically analyze the product as it is being developed and as a quality bar that has to be reached for production code.
The analysis should be automatically done by a toolset (depending on the specific implementing technology) and the metrics enforced at the code check-in to the repository. This is a very scalable solution to perform automated code reviews.

A more theoretical approach is the usage of mathematical methods to specify, verify and proof the written code. This specialized field is called “formal methods” and can be categorized in three levels: [68]

- Level 0: Formal specification
- Level 1: Formal development and verification
- Level 2: Theorem proofing

Each level differs in how extensive formal methods are used. While the specification made in level 0 can help as a description for system designers or developers, a level 2 proof of a system guarantees its correctness in respect to the specification.

Of course proofing a program comes with a significant cost in effort. Finding the necessary specification (e.g. pre-conditions, post-conditions and invariants) and mathematically proofing a complex program is still very hard.

2.3.4.4. Security Code Review

When engineering a complex system human error is a common thing. According to McConnell in [69] the industry average of errors per 1000 lines of code is about 15 to 50.

Reviewing the implemented code is a key element of finding issues. It is a white-box assessment method, allowing the reviewer to see every detail of the implemented functionality. In later phases this converts to more and more of a black-box approach, where the insight of the product gets smaller, and issues are getting harder to find.

The greater benefit may be that the review partners can talk about their best practices and share experiences with each other. This contributes to an educational environment, as described earlier in 2.3.1. “Training Phase”.

A security code review focuses especially on finding vulnerabilities that lead to security issues, as well as reviewing the implemented security controls. It is also the only viable option to review specific implementations, where no automated tool or fuzzing approach can help in finding bugs.

OWASP provides numerous guides [70] for review topics.

A few examples are:

- Reviewing Code for Data Validation
- Reviewing Code for Error Handling
- Reviewing Code for Logging Issues
- Reviewing Code for OS Injection
- Reviewing Code for Race Conditions
- Reviewing Code for Session Integrity issues
- Reviewing Code for SQL Injection

These guides help reviewers in finding specific vulnerabilities, especially in the field of web application security (which is the focus of OWASP).

2.3.5. Verification Phase

When a product is implemented it has to be tested against the security requirements formed earlier to guarantee that all the security goals were achieved in previous phases. The testing methods can be performed in different phases by different team members (e.g. by the implementation team) to verify if security requirements are met. It is common that the implemented test schemes and test environments are reused (e.g. when maintaining or updating the product) until the decommissioning of the product.

2.3.5.1. Dynamic Code Analysis

Dynamic code analysis is performed by executing programs and analyzing them during runtime. This can range from simple unit testing to live memory checking. Measures like code coverage should be used to know if the analysis was made with sufficient paths and/or lines of the program executed.
This way, runtime errors like race conditions, resource leaks, user privilege issues, code injection possibilities, null pointers, etc., can be found.
Many different tools for different platforms are available, for example resource leak detectors like AddressSanitizer, Purify or dmalloc. For the special case of multithreaded error analysis tools like Intel Thread Checker, ThreadSanitizer or the mighty valgrind suite are capable to dynamically find e.g. race conditions.

2.3.5.2. Fuzz Testing

Fuzz Testing or „Fuzzing“ is a widely-used test strategy to find security vulnerabilities (and also general program errors) in modern application software. This approach is to “fuzz” random (or malicious) data to the application’s interfaces. This should simulate the real-life usage of the software, where not only plausible data input will happen.
The basic problem with this approach is the need for millions or billions of iterations to reach a sufficient coverage of all code paths of a more complex program. Modern fuzzers take this into account and are using techniques like code instrumentation to mitigate these problems.
Fuzzing is particularly helpful, the more nested and complex operations a program is performing. A human code review is naturally limited in reliably verifying a program as it gets more complex. This is widely seen in the open source community, where even in heavily audited programs, fuzzing yields impressive results (e.g. with OpenVPN, documented by [71]).
Additionally, once a fuzz test environment is established for a certain program, it can be reused and included into a continuous-integration process. Contrary to code reviews, which are looking into a particular state of a program, the fuzz test environment can then help in finding issues continuously.

2.3.5.3. Attack Surface Review

A review of the attack surface given by the implemented program should be made after code completion. The result can be compared to the attack surface analysis and the threat modelling results made in the design phase.
The result should determine if the implementation phase contributed to an increase of the attack surface or if the design or implementation has to be changed due to new attack vectors.
A common testing method to determine the given attack surface is a black box penetration test while running the program with a specialized toolset.
Examples of these tools are Attack Surface Analyzer or AppVerifier, which are specialized in searching for attack surface issues in products related to the Microsoft environment. In a Linux environment tools like ntop, nmap, watch, strace, etc. can also be used to analyze attack surfaces depending on the type of created product.
For a common type of application, like a web application more specialized tools like OWASP Zed Attack Proxy, Burp Suite, Skipfish or w3af are available to view potential web vulnerabilities.
For a more general approach a vulnerability scanner like OpenVAS, Nessus or Nexpose can be used. These scanners assess applications, whole computers or even networks against an internal database with thousands of known exploits.
The practical usage of such tools will be shown in chapter 3. “Security Analysis of an Industrial Automation Product”.

2.3.6. Release Phase

If the developed product is getting ready to be deployed to the public, the release phase focuses on a final security and privacy review. Additionally the product can be certified with a security related certification.

2.3.6.1. Final Security and Privacy Review

Before the release of the new product, an examination of all security-related activities during the project should be performed. A best practice approach is to test the product against the given security requirements, threat models and quality gate rules. It should not be a last ditch-effort to patch found vulnerabilities.
Example activities in this final security review are: [32]

- The threat model and attack surface analysis should be reviewed and cross-checked with the final product, if they are still up-to-date. Additionally, all found threats should be examined if they were mitigated.
- All found security issues in the development process should be verified against the final product. Security issues that were suspended or declined should also be reviewed, if they apply for the final product.
- The output of all security related tools should be reviewed and in case of inconsistencies the tool should be rerun.

2.3.6.2. **Certification**

Proper certification of products (and also the development process) ensures that the intended level of security was reached. It also reassures the potential customers that the product meets given standards in information security.
Certifications in the field can be organized in two groups - product certifications and organizational certifications, but many focus only on information security in the IT infrastructure.

Below is a sample list of applicable information security related product certifications:

- Common Criteria (ISO 15408), by ITSEC Joint interpretation library
- TÜV product certifications ("proved software","proved app", "proved privacy" etc.)
- CIS Certified Security Software Products (Center for Internet Security)
- ICSA Labs Certification

These certifications focus on the certification of a specific product. Often this comes only in combination with a process-oriented company certification. Applicable examples are as follows:

- ISO 27001 Information Security Management
- IEC 62443 Conformity Assessment Program
- ISO 21827 Systems Security Engineering Capability Maturity Model (SSE-CMM) - ISSEA

2.3.6.3. **Archiving**

The archiving of all data, including source code, binaries, requirements, documentation, licenses, etc., should be checked at the end of the project. This is equally important for any third party software.

2.3.7. **Response Phase**

Creating and executing an incident response plan is the final act for a product in its secure development lifecycle. Incident response deals with threats that occur while the product is deployed in the field. Incidents can come from internal or external resources. An emergency contact must be provided to handle the information properly.
Incident response also involves the handling of inherited code from third parties or other development teams in the company. It is also vital to observe used third party components in worldwide CVE repositories to find advisories about used versions of these products.
It depends on the organization of the company whether the responsibilities of the product stays with the same team or whether a dedicated engineering team is formed and commissioned with the task of incident response.
To publicly announce the found vulnerabilities within such incidents is called a security advisory. This will be discussed in the following chapter.

2.3.7.1. Security Advisory

When vulnerabilities are found in a product that is publicly released, it is the obligation of the selling company to disclose it as an advisory in public. A best-practice approach for companies today is to disclose these security advisories on the homepage of the company in a separate section for information security concerns.

A security advisory should implement the following sections:

Notice:

In the notice section, legal information like copyright claims, warranty notices, responsibilities etc. should be clarified.

Affected Products:

All of the company's products that are affected by the specific vulnerability should be declared. Different hardware and software releases and versioning must be considered. It should be stated if the company also includes advisories on end of life products or if they are omitted.

Vulnerability Summary:

A thorough explanation of the vulnerability must be provided. Scenarios in which the vulnerability can or could be exploited should be given. If exploits were witnessed or implemented in the incident response process and mitigations are available, they can be released for testing purposes.

Naming and Severity Rating:

The vulnerability should be given a Common Vulnerability and Exposure (CVE) Identification for public referencing.

Additionally the vulnerability should be rated according to the Common Vulnerability Scoring System (CVSS) which states:

"The Common Vulnerability Scoring System (CVSS) is an open framework for communicating the characteristics and severity of software vulnerabilities. CVSS consists of three metric groups: Base, Temporal, and Environmental. The Base group represents the intrinsic qualities of a vulnerability, the Temporal group reflects the characteristics of a vulnerability that change over time, and the Environmental group - represents the characteristics of a vulnerability that are unique to a user's environment. The Base metrics produce a score ranging from 0.0 to 10.0, which can then be modified by scoring the Temporal and Environmental metrics." [72]

For some companies it could be suitable to apply for a CVE Numbering Authority (CAN). This authorizes an organization "to assign CVEs to vulnerabilities affecting products within their distinct, agreed upon scope, for inclusion in first-time public announcements of new vulnerabilities." [73]

Mitigations:

In this section, corrective actions or resolutions of the vulnerability and how to implement them should be presented. As an example the company could provide a software update which closes the vulnerability and further documentation on how to apply the update for the corresponding devices.

If no direct mitigation is available the advisory could also point out workarounds or recommendation of usage until a corrective measure exists.

Support:
A section for contact information regarding the vulnerability should be provided. Also references to additional information can be included here.
Credit:
Credits to the reporting party (if external) should be included, if the reporting party wants to be publicly credited.

2.3.8. Agile Adaption

To use different methods of improving the product security also when they are developed in an agile process, we have to align them to the agile methodology.
It is obvious that not all measures can be completed in one sprint with the duration of two weeks.
Therefore, the activities have to be divided in three categories: [32]

- Every Sprint: measures to be done in every sprint
- One Time: measures to be done only one time (mostly at the beginning or at the end of a project)
- Bucket: measures to be done over time but finished within a limit (e.g. six months).
  This can mean that the task at hand is sliced into 12 pieces and within every sprint one chooses one of the pieces (but not necessarily).

The figure below shows a visual representation of the Microsoft SDL adapted for an agile, sprint-based process.

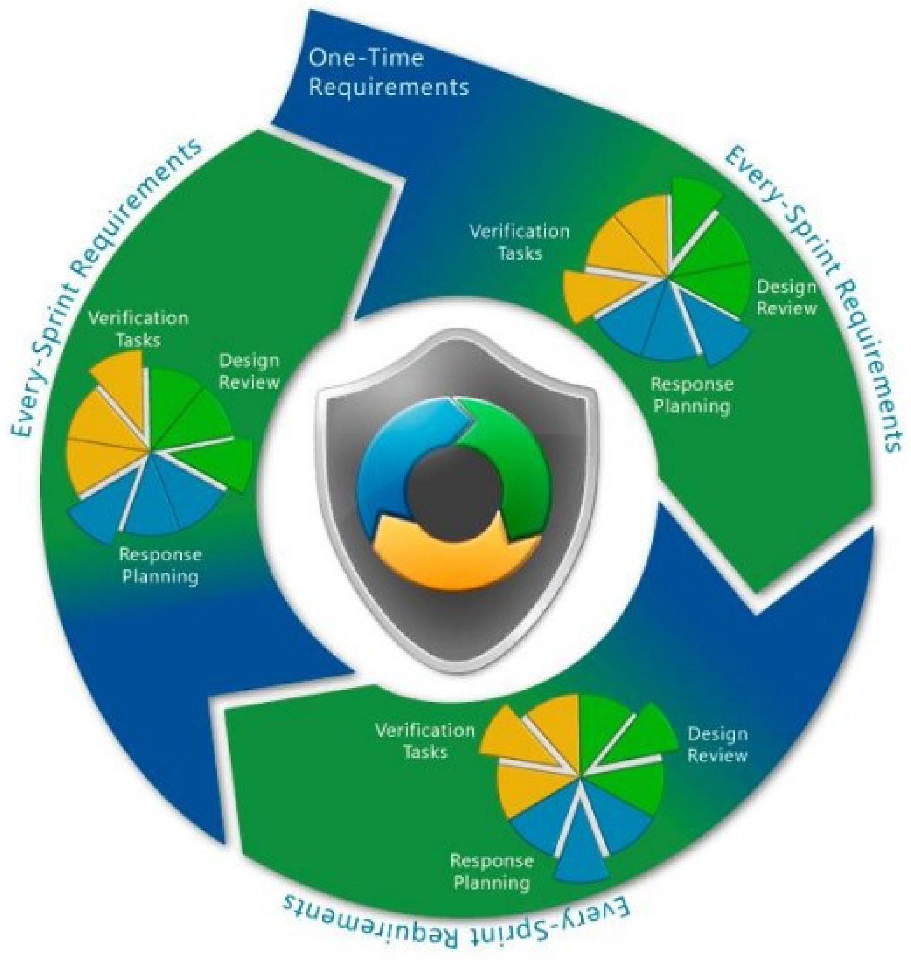


Figure 15 – Agile Model for Microsft SSDLC[15]

The figure shows the split of the security activities into the three categories. The one-time activities are indicated by the dark blue color, which are most likely to be done at the start or at the end of the lifecycle. A good example is the training phase, where a good start would be to train the team members in topics like security requirements or otherwise, it would be hard to start the requirements phase.
Bucket tasks are displayed in the figure as small buckets with slices in every sprint. Good examples are security verification tasks, like fuzz tests for attack surface analysis. They have to be performed, but not necessarily in every sprint.
And as the final category, the figure shows three sprints where certain activities have to be performed in every sprint. Good examples are the tasks from the Implementation phase, where every piece of written code has to undergo static code analysis to ensure a certain software quality.

[15] Taken from: Microsoft Corporation: Simplified Implementation of the Microsoft SDL, 2010, available at https://www.microsoft.com/en-us/download/details.aspx?id=12379

2.4. Handling Information Security in Existing Software

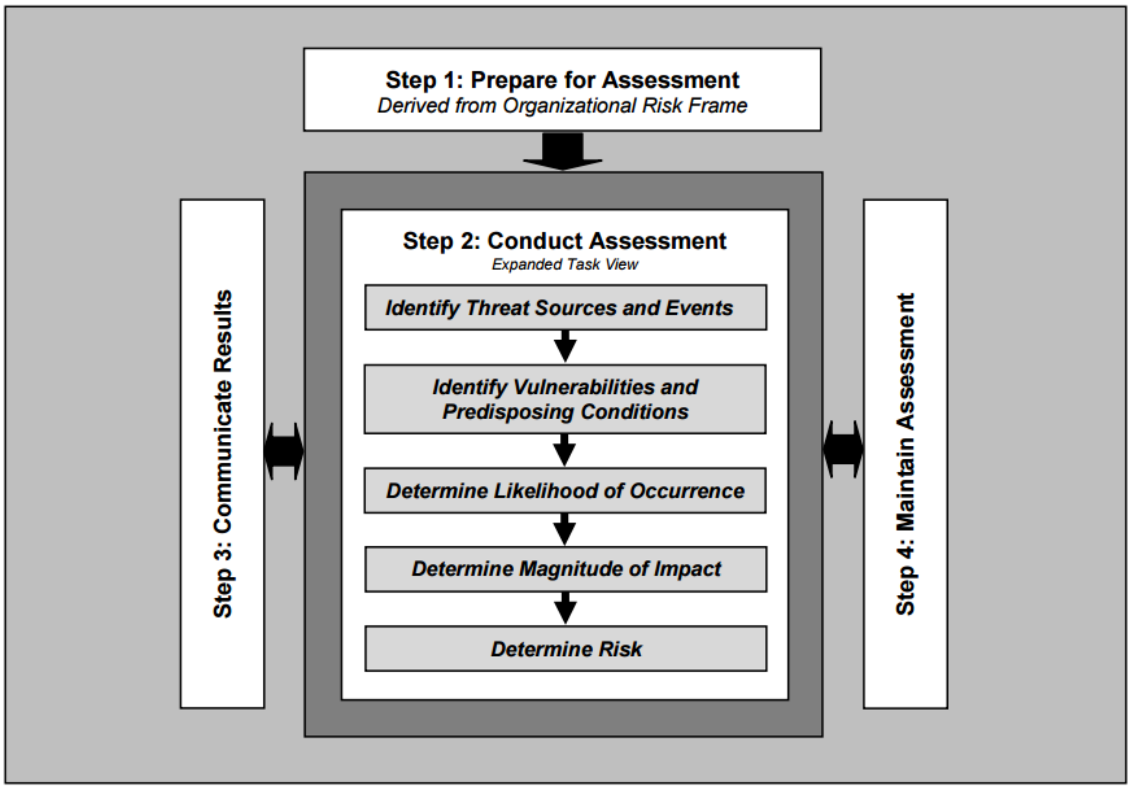
As described in the chapter 2.3 “Information Security in the Development Lifecycle”, we have found numerous ways to improve the security of newly developed software. The apparent problem is what to do with already existing products with an older code base. To apply the measures of the introduced SSDLC, a review (and potentially refactoring) of every existing piece of software created and still in operation (whether in the company or at customers), could be necessary.
Since this is not a realistic scenario, other measures to mitigate the security risk have to be applied in existing software.

2.4.1. Risk Management

From a business perspective, applying the case of risk management is the only option to approach information security in existing products. It is not possible to ensure “complete” secureness in a product. A middle ground between the investment cost and residual risk from security issues has to be found. A possible method for achieving this is presented in this chapter.

2.4.1.1. Risk Assessment

To perform a risk assessment is the first step to analyze the risk given by a certain product. The risks can then be estimated and prioritized, which enables the responsible persons to consciously make the correct decisions on what to do with these risks.
The NIST Special Publication 800-30r1 “Guide for Conducting Risk Assessment” [74] shows a comprehensive way to execute such an assessment:



Step 1: Prepare for Assessment
Derived from Organizational Risk Frame
Step 2: Conduct Assessment
Expanded Task View
Identify Threat Sources and Events
Identify Vulnerabilities and Predisposing Conditions
Determine Likelihood of Occurrence
Determine Magnitude of Impact
Determine Risk
Step 3: Communicate Results
Step 4: Maintain Assessment

Figure 16 - NIST SP80030r1 Risk assessment process[16]

Assessment Preparation

In preparation for the risk assessment, the goal, purpose and scope of the assessment itself have to be initially defined. Then the practical method of conducting the risk assessment has to be chosen.

Identify Threat Sources, Threat Events, Vulnerabilities and Predisposed Conditions

To find threats and vulnerabilities, various techniques can be applied. The NIST SP800-30 framework explores the possible sources and events with a collection of examples. These lists are analyzed and a threat, if applicable, is noted.
As we have seen in chapter 2.3. “Information Security in the Development Lifecycle” more promising approaches for software threats can be applied here. For example, threat modelling with its different finding techniques could lead to more specific threats.

Additionally, as we already have working software, a security audit of the product will directly determine vulnerabilities or hints for bad design (and, therefore, the possibility of future threats) of the software. An example of a practical security analysis will be shown in the following chapter 3. “Security Analysis of an Industrial Automation Product”.

[16] Taken from: Stoneburner, G. el al.: NIST SP800-30r1 Guide for Conducting Risk Assessments, September 2012, available at http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf

Determine Likelihood

As a next step, the threat's likelihood of occurrence has to be defined. This is one of the two crucial metrics which will allow calculating the risk of the threat. The properties of the threats found from the previous task have to be taken into consideration.
NIST SP800-30 puts it this way:
"Determine the likelihood that threat events of concern result in adverse impacts, considering: (i) the characteristics of the threat sources that could initiate the events; (ii) the vulnerabilities/predisposing conditions identified; and (iii) the organizational susceptibility reflecting the safeguards/countermeasures planned or implemented to impede such events."

Determine Impact

Assessing the impact of the given threat will define the second metric for calculating the risk factor of an attack. Again the possible impact can be cross-referenced to tables with experienced impacts. STRIDE can also give a hint as to which security categories have been impacted by the given threat.

Calculate Risk

Calculating the risk of a given threat is now very easy given the impact and the likelihood of the risk by simply multiplying those factors.
Other frameworks give different approaches which include other or more factors. An example is the DREAD methodology, which gives a rating based on 5 factors: [75]

- Damage - How bad would an attack be?
- Reproducibility - How easy is it to reproduce the attack?
- Exploitability - How much work is it to launch the attack?
- Affected users - How many people will be impacted?
- Discoverability - How easy is it to discover the threat?

Given that the risks were discovered within a security analysis (which makes them vulnerabilities) the rating system "Common Vulnerability Scoring System" (CVSS) can be applied.

Communicate Results

The result of the risk assessment should be communicated to the effected business areas. Depending on the severity of the results, this could lead to directly informing the upper management to support risk responses.
Also, a disclosure of the found threats or vulnerabilities is possible. This has to be defined in the incident response plan as we have seen in the previous chapter 2.3.7. "Response Phase".
The different ways of handling the risk will be discussed in the following chapter.

Ongoing Assessment

Risk management is not a one-time process, but has to be ongoing to keep the risk knowledge of the underlying product up-to-date. The frequency of risk monitoring has to be defined and when changes to the product occur, it should lead to a rerun of the risk assessment process.

2.4.1.2. Handling Risk

Handling risk means investigating the prioritized list of risks from the risk assessment to select appropriate controls in descending order. It is important to fully understand the result of the risk assessment, as some risk treatments can address many different risks from the list and should be pursued. [76]
Different approaches (which are not mutually exclusive) can be made:

Risk Avoidance
If possible, the first attempt to conquer a certain risk should be to avoid it completely. Oftentimes this can be achieved by fixing a simple programming error, but in other cases functionality must be turned off completely. Risk avoidance by adding new functionality or by fixing existing problems must be considered carefully, as new issues could be introduced by the new code.

Risk Mitigation
Risk Mitigation is a mechanism to reduce either the impact or the likelihood of a known risk. This is often a viable strategy if risk avoidance cannot be achieved (i.e. turn off the feature completely).

Risk Transfer
This means to transfer the risk to a party that is better able to manage it. Often this includes third party companies like banks or insurance companies. Most of the time this sounds simpler than it is because of the quantification problem of the assessed risk.
A special form of risk transfer is also to transfer the risk to the customer. As with industrial products, it is common to sell them as an open platform, where the customers (e.g. machine builders) deploy their own programs and, therefore, are also able to customize the software configuration.
A special case of risk transfer is to share the risk with e.g. other organizations or customers. This can be liabilities or responsibilities for other adequate risk responses (like risk mitigation). Specialized companies use this as their business model, by providing expertise or certain type of resources which enables them to address these risks in a better way. [77]

Risk Acceptance
Risk Acceptance is the process of consciously accepting the known risk. Usually this means the avoidance, mitigation or transfer of the risk is likely to cost more than the cost if the corresponding incident occurs or the likelihood is minimal.

Residual Risk

Even if countermeasures are applied to every known risk, there is still the (very high) possibility of other unknown risks. It is vital to keep a balance between additional security assessments and the associated costs.

Though in this use case we investigated risk management in the context of handling information security in existing products, risk management must also be taken into account at the different stages of the (security) development lifecycle.
NIST SP 800-64 gives advice which support risk management activities can give in the different phases. [78]

2.5. Conclusion

At the beginning of this chapter, we have seen that many organizations and institutions are pursuing the goal to give proper advice on information security measures. Standards have been formed to focus on the special needs of industries, such as industrial automation.

Derived from these standards, the process of developing new products can be improved to fulfill the growing need for information security. More and more organizations in the IT industry have adopted security development lifecycles and their security measures in the past.
As industrial products are evolving from proprietary hardware and software to standard components and connecting to the IT world with known protocols and functionality, it is overdue for industrial companies to pay more attention to security assurance processes. The rising call for regulatory measures and increasing awareness of industrial asset owners and operators are additional triggers to get the automation industry to look into the matter of security.

It can be assumed that in the past industrial products were developed with not as many security practices in mind as today. We have derived that it is not possible to refactor security into these products, as when developing new products with adopting a security lifecycle. The risk management process gives a business perspective on how to treat such devices.
As a practice example and to underline the necessity of introducing security improvement measures in an industrial environment, a black-box penetration test of an industrial automation device will be executed in the following chapter.

3. Security Analysis of an Industrial Automation Product

3.1. Disclaimer

The following analysis methods, test cases and findings are intended only to help building better systems in the future. They are not intended as means to help breaking into or disrupting the activity of a system or doing anything else illegal, immoral or unethical.

3.2. Approach

This security analysis will be performed as a black box penetration test. This means that it examines the functionality of the device without looking into the internals (e.g. the source-code) or internal structure (e.g. debugging interfaces). It is performed as an exploratory test to get to know the different angles of penetration testing. It must not be misinterpreted as a fully-fledged security audit.
As we have derived from chapter 2.1. “Attack Vector Taxonomy”, the focus will be logical attacks on the Ethernet interface of the device, as this usually is the primary entry point of malicious attacks. A widened approach could investigate for logical attack possibilities on other interfaces like the industrial fieldbus or the serial interface. To complete the possible attack vectors also physical or side-channel attack possibilities could be investigated in a further analysis.

The penetration test will be the first part of a Security Risk Assessment process of existing B&R control products. The setup was chosen because of the high market share in the field. This setup is seen as a best practice solution for small to medium-sized machine control appliances.
The goal is to determine if the control system is vulnerable to attacks which are done intentionally, but with simple means (according to ISA/IEC 62443 Security Level 2).

3.3. Experiment Setup

3.3.1. Used B&R Hardware

3.3.1.1. B&R X20 CP1585 Industrial PLC

A standard B&R Industrial PLC is used as the primary target for the penetration test. The X20 series is the most common PLC from B&R and also the best-selling PLC range of the company with over two million applications worldwide (as of 2016).

The CPx5xx range features Intel Atom based X86-CPUs with a basic entry model clocked at 333 MHz and 128 MB RAM up to the top of the range model clocked at 1.6 GHz and 512MB RAM.
The provided CP1585 has a fan-free Intel Atom E640T clocked at 1.0 GHz with an additional I/O processor supported by 256MB DDR2-SDRAM. Compact Flash is used as a removable application memory, containing the operating system and the application-specific cyclic programs and configuration.
The PLC shows six different types of interfaces to connect to peripherals and other systems:

1) IF1: RS232, connection made using 12-pin X20TB12 terminal block with a max. transfer rate of 115.2 kbit/s
2) IF2: RJ45, Standard Ethernet capable of 10 BASE-T/100BASE-TX/1000 BASE-T.
3) IF3: RJ45, POWERLINK (V1/V2) managing or controlled mode, Type 4 with a max. transfer rate of 100Mbit/s
4) IF4: a standard Type A USB 2.0 Port
5) IF5: a second USB 2.0 Port
6) IF6: X2X Fieldbus Master

3.3.1.2. B&R PPC 2100 Industrial Terminal PC

The B&R PPC 2100 terminal is an integrated industrial PC with a display attached. It has an X86 architecture, featuring an Intel Atom (Baytrail) E3845 quad-core processor with 4GB RAM. For this analysis Windows 7 embedded is used as an operating system, though other systems (like Debian Linux) are available for this kind of terminal PC. The terminal itself is not a part of the technical analysis, but it can be part of a follow-up research project.
For the different purposes of client communication, programs like Google Chrome (Web visualization client), a VNC-Viewer (VC4 visualization client), an OPC-UA client, VC4 for Terminal mode and a proprietary INA/ANSL client are installed.

3.3.2. Used B&R Software

3.3.2.1. B&R Automation Studio 4.2.5.238

B&R Automation Studio is the integrated development environment (IDE) used to develop automation projects including B&R hardware products. It has an "all-in-one" approach and contains all tools necessary for the whole automation process, beginning with the start of the project to deploying and maintaining it.
For this test, it was only used to deploy the standard B&R training project "CoffeeMachine" to the PLC and for failure analysis. This project is generally selected as a starting point for many automation projects in the field, as it delivers a minimalistic cut-through approach to all features necessary in modern applications. Therefore, it is

assumed that the majority of the detected vulnerabilities and issues with this version will be seen in running applications worldwide.
The general approach in the field is to add features to the application until the requirements of the specific project are met and not to deactivate parts or even try to harden the application.

3.3.2.2. B&R Automation Runtime D4.2.5.2

B&R Automation Runtime is the architectural runtime system based on Windriver VxWorks operating on the PLCs. It contains the real-time application scheduler, B&R system components and also the project-specific application. This will be the main target for this penetration test, as the Automation Runtime handles all communication on the Ethernet interface.

3.3.3. Used Analysis Software

- Kali Linux Rolling 2016.1 and 2016.2
- Wireshark 2.2.3
- NMAP 7.12
- Chrome + Chrome DevTools 49.0.2623.112 m
- w3af 1.6.54
- OWASP WebScarab
- OWASP ZED Attack Proxy 2.4.3
- Burp Suite Free Edition v1.7.03
- AFL 2.10b
- TLSSLed 1.3
- sslyze 0.15.0
- Metasploit 4.12.23
- hashcat 3.10
- Hping3.0.0a2
- JohnTheRipper 1.8
- netcat v1.10-41
- curl 7.51.0
- packETH 1.8.1
- Ostinato 0.8
- OpenVAS-8
- Nessus Home Edition 6.5.2
- T50 5.4.1-rc1
- WFuzz 2.0
- VNCcrack 2.1
- THC Hydra 8.4
- Medusa 2.2
- Ncrack 0.5
- patator v0.5
- Unified Automation UaExpert 1.4.
- snmp-check v1.8

3.3.4. Topology

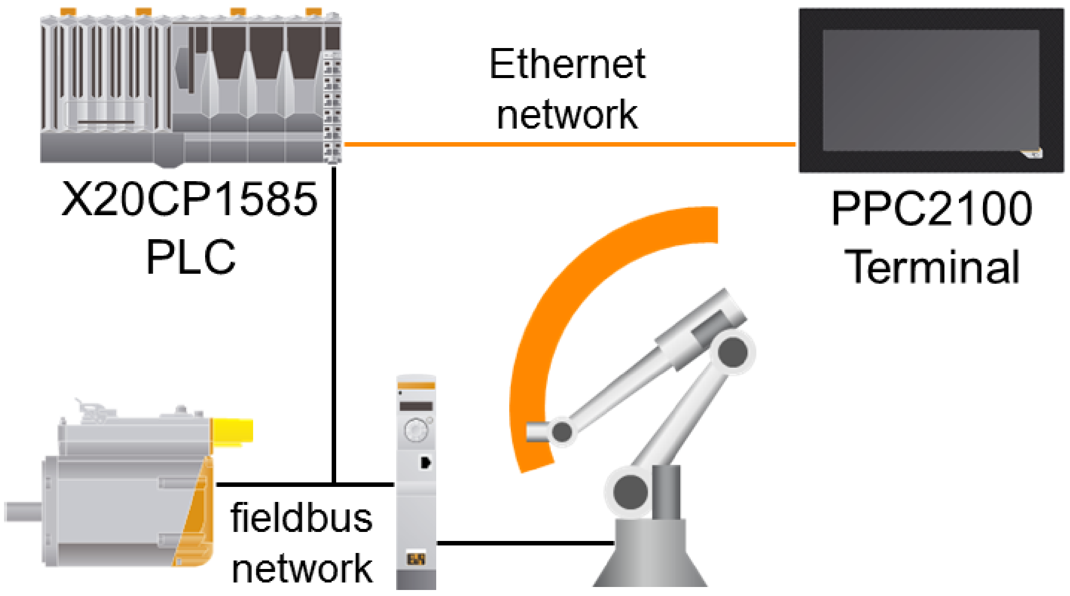


Figure 17 - Security Analysis Topology[17]

The topology used demonstrates the assumption that the PLC and the Terminal communicate over Ethernet network. Also we assume that there could be other means of communication over the Ethernet, such as remote maintenance (e.g. via VNC) or remote data extrusion (e.g. via OPC-UA). The PLC is then connected to the controlled machine via the fieldbus network.

3.4. Analysis

3.4.1. Information Gathering

At the start of a black box security analysis, the major goal is to collect as much information about the assessment target as possible.
"Information gathering and research can be of great value in penetration testing, particularly in the case of a penetration test in which we do not have inside knowledge of the targets or the environment. A number of data sources, and quite a few different tools, are available for us to use in the course of our efforts." [79]
Generally speaking, the target search can be divided into non-intrusive and intrusive searching. As in this case the means of non-intrusive searching, other than reading public material (like user manuals) provided by the B&R homepage, is rather limited. Therefore, the information gathering process starts with an intrusive target search, where the following has to be considered:

[17] [selfmade diagram by author]

"This is when you probe and explore the target network; consequently, ensure that you have the explicitly written permission to carry out this activity with you. Never perform an intrusive target search without permission as this written authorization is the only aspect which differentiates you from the malicious hacker." [80]

3.4.1.1. **Port Scan**

As a start, the basic attack surface of the PLC on the Ethernet interface was analyzed. The popular tool NMAP was used to get a list of open TCP and UDP network ports.
NMAP is a freely available network scanning tool licensed under GPLv2. It sends specially crafted network packets to selected hosts in a network and analyzes the responses.
With this approach, the following information can be obtained from the network or respectively the hosts:

- Identifying hosts on a network (e.g. by ICMP discovery)
- Scanning network ports on hosts
- Version detection of protocols or operating systems of the host

A full NMAP port scan yielded the following results:

Port number	**Service**	**Protocol**	**Description**
68	DHCPc	UDP	DHCP-Client
69	TFTP	UDP	TFTP-Server for System Image Uploads (only for specific terminals (T-Series)
80	HTTP	TCP	Port of the legacy webserver (explained in detail in the analysis section of the web server)
81	HTTP	TCP	Port of Webservers for the Web-Visualization. Port 80 is explicitly omitted here, for compatibility reasons to the existing older webserver (and webservice).
161	SNMP	UDP	SNMP Communication
443	HTTPS	TCP	Webserver Port for encrypted (SSL/TLS) communication
4840	OPC-UA	TCP	OPC-UA Server for machine data communication
5900	RFB	TCP	VNC service for displaying the legacy visualization VC4
11159	INA (prop)	UDP	Proprietary B&R protocol for communication between PLCs and to Engineering Environment (Automation Studio) - deprecated
11169	ANSL (prop)	TCP	Proprietary B&R protocol for communication between PLCs and to Engineering Environment (Automation Studio)

Based on these results, a closer look into a selection of these services to find out more about what data is transferred via these ports is taken.

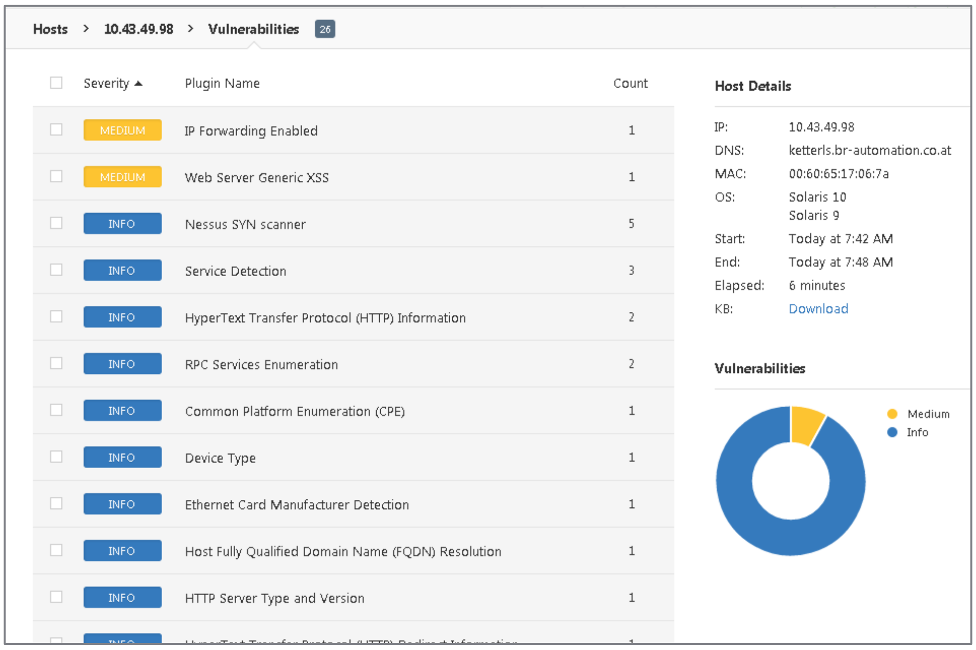
3.4.1.2. Vulnerability Scanners

To initially detect issues with the given system, vulnerability scanners were used. Vulnerability scanners test a wide range of known vulnerabilities and are designed for ease of usage. Only an IP address (or a range of IP addresses) and some type of attack scenario has to be provided, and the scanner then analyzes the targeted system and provides a report on the findings.
In this scenario, the test of the SSL/TLS communication on Port 443 was explicitly excluded, as a deeper analysis of the encryption will be shown in section 3.4.2. “Web Server”.

3.4.1.2.a Nessus

Nessus is a proprietary vulnerability scanner developed by Tenable Network Security. It started as free software under GPL license and was available for the public. Since 2005 with version 3 of Nessus, the source was closed and commercially licensed. It is the most popular commercial vulnerability scanner and Tenable claims to have over 20.000 business customers worldwide. The system implements a plugin infrastructure which consists of over 80.000 plugins, covering over 35.000 CVEs.
Only the very limited Nessus Home version is available free of charge, but only after a registration process. Advanced versions would also enable special SCADA plugins, which could yield more interesting results.

After an advanced network scan with Nessus Home, the following report was generated:



Severity ▲	Plugin Name	Count
MEDIUM	IP Forwarding Enabled	1
MEDIUM	Web Server Generic XSS	1
INFO	Nessus SYN scanner	5
INFO	Service Detection	3
INFO	HyperText Transfer Protocol (HTTP) Information	2
INFO	RPC Services Enumeration	2
INFO	Common Platform Enumeration (CPE)	1
INFO	Device Type	1
INFO	Ethernet Card Manufacturer Detection	1
INFO	Host Fully Qualified Domain Name (FQDN) Resolution	1
INFO	HTTP Server Type and Version	1
INFO	HyperText Transfer Protocol (HTTP) Redirect Information	1

Host Details

IP:	10.43.49.98
DNS:	ketterls.br-automation.co.at
MAC:	00:60:65:17:06:7a
OS:	Solaris 10 Solaris 9
Start:	Today at 7:42 AM
End:	Today at 7:48 AM
Elapsed:	6 minutes
KB:	Download

Vulnerabilities

Figure 18 - Nessus Scan Report Screenshot

As can be seen, the scanner revealed basic reports about open ports of the target and which services are running.

The only two vulnerabilities found were a potential Cross-Site-Scripting issue (this matter will be discussed later in the topic “Web Server”) and an IP forwarding service.

3.4.1.2.b OpenVAS

The second vulnerability scanner used is OpenVAS (Open Vulnerability Assessment System). OpenVAS is a fork of Nessus Version 2.2 (the last open source version) and is licensed under General Public License (GPL). As its commercial rival, it features a vast set of plugins (over 47.000) called Network Vulnerability Tests (NVTs).
The architecture of OpenVas is shown in the following figure:

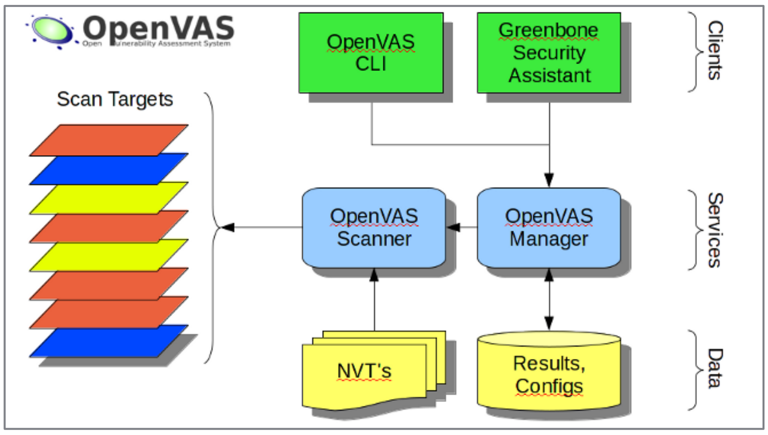
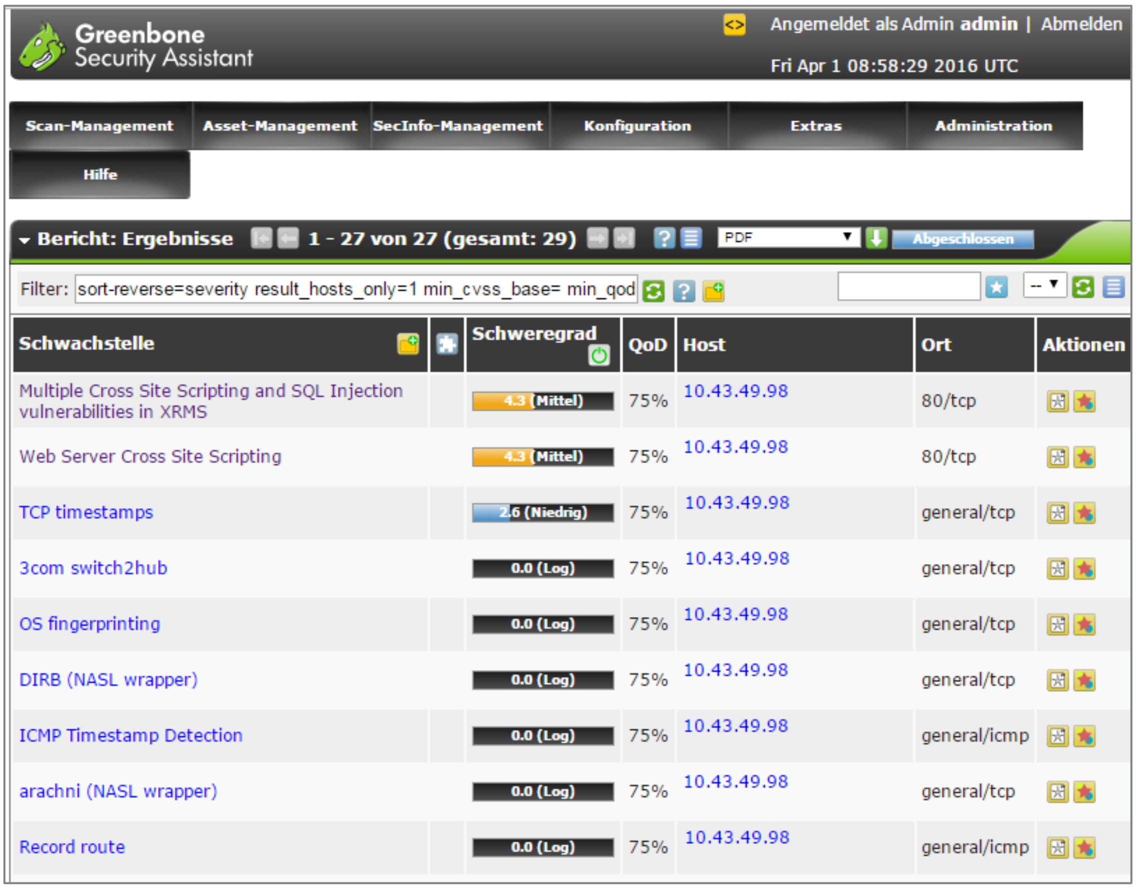


Figure 19 - OpenVAS Architecture[18]

The core module to do the actual security scanning is the OpenVAS Scanner. Thereby, it is fed by the thousands of NVT plugins to run through during the scan. The scanner is managed by the OpenVAS Manager which receives the user inputs from a CLI and a web-interface called the Greenbone Security Assistant. It saves the configuration and the results from the scanning in a separate archive.

The results of the OpenVAS Scan can be seen in the report below. They are very similar to the results of the Nessus Scan, showing the potential XSS vulnerabilities. As previously stated, these will be discussed in the next chapter.

[18] Taken from Greenbone Networks GmbH, available at http://www.openvas.org/software.html#feature_overview



Filter: sort-reverse=severity result_hosts_only=1 min_cvss_base= min_qod

Schwachstelle	Schweregrad	QoD	Host	Ort
Multiple Cross Site Scripting and SQL Injection vulnerabilities in XRMS	4.3 (Mittel)	75%	10.43.49.98	80/tcp
Web Server Cross Site Scripting	4.3 (Mittel)	75%	10.43.49.98	80/tcp
TCP timestamps	2.6 (Niedrig)	75%	10.43.49.98	general/tcp
3com switch2hub	0.0 (Log)	75%	10.43.49.98	general/tcp
OS fingerprinting	0.0 (Log)	75%	10.43.49.98	general/tcp
DIRB (NASL wrapper)	0.0 (Log)	75%	10.43.49.98	general/tcp
ICMP Timestamp Detection	0.0 (Log)	75%	10.43.49.98	general/icmp
arachni (NASL wrapper)	0.0 (Log)	75%	10.43.49.98	general/tcp
Record route	0.0 (Log)	75%	10.43.49.98	general/icmp

Figure 20 - OpenVAS Scan Report Screenshot

3.4.2. Web Server

3.4.2.1. Introduction

B&R uses the internal web server for its new web-based industrial visualization, called mapp View. An embedded web server is used to deliver the visualization pages to the visualization terminal, which in essence means, that it supplies a client with a state of the art web browser).
There is no usage of plugins or browser applications on the client side. The content delivered consists of plain HTML, CSS and Javascript files and additional media assets (e.g. pictures, PDFs, videos, etc.).
The communication of process variables to the displaying web widgets is done via a websocket in a JSON format. This is more effective than a classic HTTP request/response system to poll the variables.

Per default, the webserver listens to HTTP requests on port 81 and to HTTPS requests on port 443. The default port for HTTP-based communication (80) is not used due to compatibility reasons with a legacy webserver. This legacy web server is not activated in the tested version of the coffee machine example application. The port is open but no listening service is attached.

Some of the analysis steps shown in the following chapter would also be applicable for the legacy web server, if the service had been enabled.

3.4.2.2. Web Security Scanner

As shown before, the vulnerability scanners have already found some issues with the underlying web server of the visualization. There are specialized security scanners explicitly used for scanning web services. In this analysis, OWASPs ZED Attack Proxy (ZAP) was used. ZAP is a very popular open source, web-security tool with multiple functionalities: [81]

- Intercepting Proxy
- Traditional and AJAX spiders
- Automated scanner
- Passive scanner
- Forced browsing
- Fuzzer
- Dynamic SSL certificates
- Smartcard and Client Digital Certificates support
- Web sockets support
- Support for a wide range of scripting languages
- Plug-n-Hack support
- Authentication and session support
- Powerful REST based API
- Automatic updating option
- Integrated and growing marketplace of add-ons

3.4.2.3. HTTP Response Header Flags

The quick analysis feature of ZAP showed similar issues which the vulnerability scanners had already found. These are not set options in the HTTP response header:

- To deny Clickjacking-attacks, the web server must set the X-Frame-Option of the HTTP response header to “DENY” if the page is not embeddable within a frame. Otherwise the option “SAMEORIGIN” is possible, or if there are trusted pages it can be set to “ALLOW-FROM” with the indicated pages as a parameter. If this flag is set the browser can deny pages to embed the visualization service. More information is available in RFC7034 or in the MSDN. [82]
- To allow the browser to activate its anti-cross-site-scripting functionality the X-XSS-Protection Flag of the HTTP response header must be set. [83]
- To deny MIME-type sniffing the X-Content-Type-Options flag in the HTTP response header should be set to “nosniff” [84]

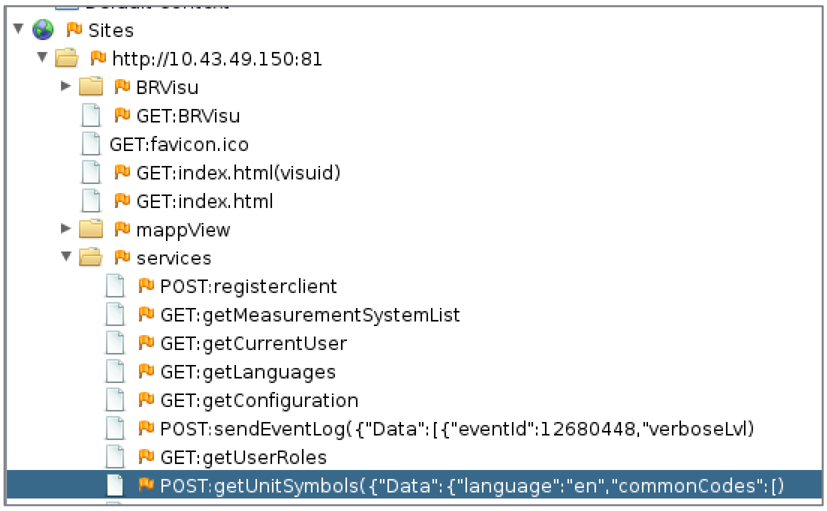
Within the typical use cases of the web visualization, these issues are not too relevant (as no embedding of the sites are planned) but should be fixed nonetheless.

3.4.2.4. Proxy Analysis

With ZAP it is possible to use a man-in-the-middle proxy for deeper analysis. It includes a browser extension for Firefox, which tunnels the web traffic through the ZAP scanner. ZAP will create a sitemap in the background when the user visits different pages.
With the deep scan possibility, ZAP will investigate the sitemap even further with an add-on called spider (a so-called web crawler). ZAP also retries every HTTP request it has learned and also alters the payloads.

Parsing Error in POST - Function

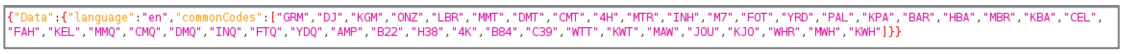
While trying to call the requests ZAP found the POST functions “getUnitSymbols”:



```
▼ Sites
  ▼ http://10.43.49.150:81
    ▶ BRVisu
      GET:BRVisu
      GET:favicon.ico
      GET:index.html(visuid)
      GET:index.html
    ▶ mappView
    ▼ services
        POST:registerclient
        GET:getMeasurementSystemList
        GET:getCurrentUser
        GET:getLanguages
        GET:getConfiguration
        POST:sendEventLog({"Data":[{"eventId":12680448,"verboseLvl)
        GET:getUserRoles
        POST:getUnitSymbols({"Data":{"language":"en","commonCodes":[)
```

Figure 21 - ZAP Sitemap and Vulnerable Function Screenshot

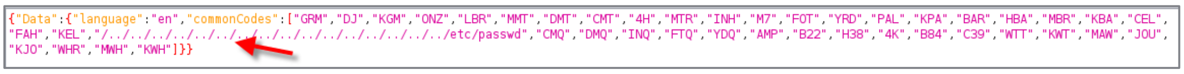
The payload of the function when called by the user looks as following:



```
{"Data":{"language":"en","commonCodes":["GRM","DJ","KGM","ONZ","LBR","MMT","DMT","CMT","4H","MTR","INH","M7","FOT","YRD","PAL","KPA","BAR","HBA","MBR","KBA","CEL",
"FAH","KEL","MMQ","CMQ","DMQ","INQ","FTQ","YDQ","AMP","B22","H38","4K","B84","C39","WTT","KWT","MAW","JOU","KJO","WHR","MWH","KWH"]}}
```

Figure 22 - Unmodified Request Payload Screenshot

ZAP automatically calls this function many times and tries to modify its payload as in the following example:



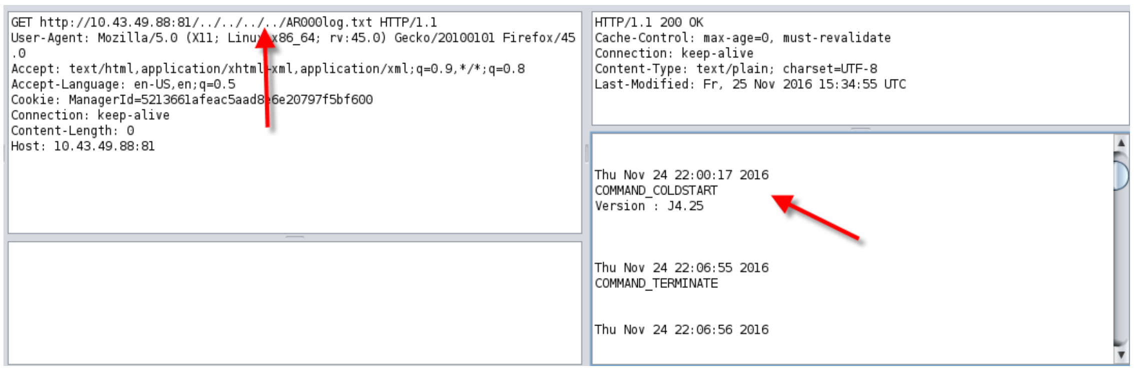
```
{"Data":{"language":"en","commonCodes":["GRM","DJ","KGM","ONZ","LBR","MMT","DMT","CMT","4H","MTR","INH","M7","FOT","YRD","PAL","KPA","BAR","HBA","MBR","KBA","CEL",
"FAH","KEL","/../../../../../../../../../../../../../../../../etc/passwd","CMQ","DMQ","INQ","FTQ","YDQ","AMP","B22","H38","4K","B84","C39","WTT","KWT","MAW","JOU",
"KJO","WHR","MWH","KWH"]}}
```

Figure 23 - Modified Request Payload by ZAP Screenshot

This results in a crash of the visualization server and, therefore, a denial-of-service of the PLC (as it goes into service mode).

HTTP Get Path Traversal

While manually inspecting the requests, the HTTP Get request was forged to try to break out of the webroot of the web server. [85] Since modern browsers are not allowing this malicious input, ZAP was also used to demonstrate this:



```
GET http://10.43.49.88:81/../../../../AR000log.txt HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45
.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Cookie: ManagerId=5213661afeac5aad8e6e20797f5bf600
Connection: keep-alive
Content-Length: 0
Host: 10.43.49.88:81
```

```
HTTP/1.1 200 OK
Cache-Control: max-age=0, must-revalidate
Connection: keep-alive
Content-Type: text/plain; charset=UTF-8
Last-Modified: Fr, 25 Nov 2016 15:34:55 UTC
```

```
Thu Nov 24 22:00:17 2016
COMMAND_COLDSTART
Version : J4.25


Thu Nov 24 22:06:55 2016
COMMAND_TERMINATE


Thu Nov 24 22:06:56 2016
```

Figure 24 - HTTP Get Traversal Exploitation ZAP Screenshot

As seen here, the web server does not validate the input parameters given in the get request. A malicious attacker can go through the folders and read random files. This could be used, for instance, to read private keys for the encryption of communication.

3.4.2.5. **Traffic Analysis**

In this case we assume that the attacker can sniff the network traffic between the web server and the client communication. In the following figure an example topology is shown:

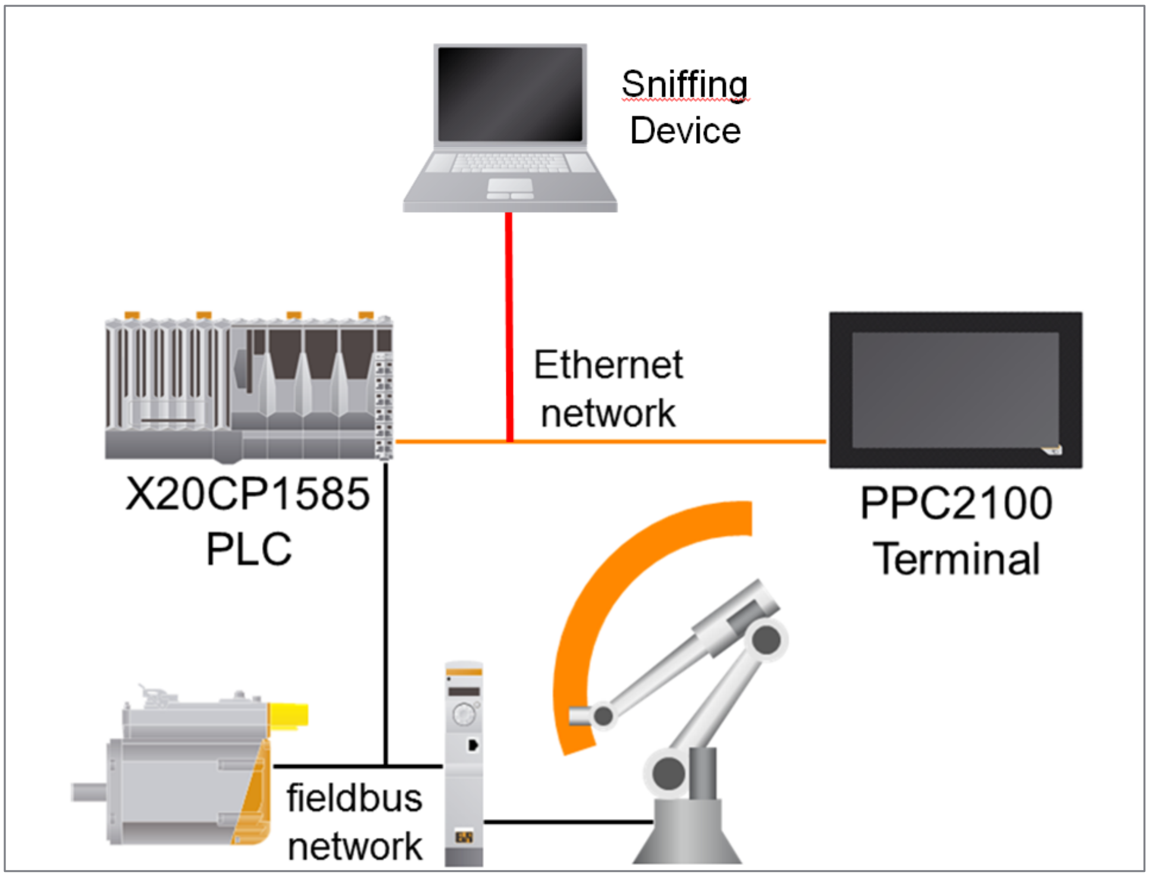


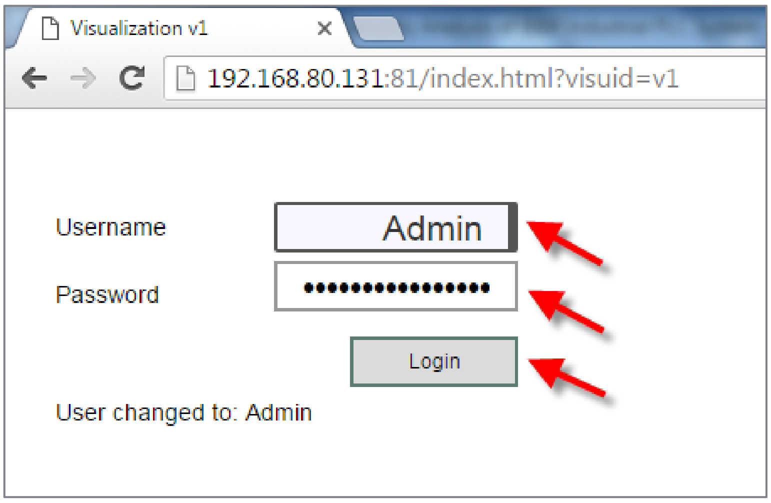
Figure 25 – Sniffing Topology[19]

This can be achieved with techniques like MAC/ARP spoofing or also with special hardware devices. In managed switching environments, the switch itself could also be attacked and then configured to mirror the communication between the client and the server to the attacker. Furthermore, it is possible that in industrial environments hubs instead of switches may still be used to transmit Ethernet communication between devices. With hubs, the Ethernet traffic of any attached device is mirrored to every other device, which makes sniffing very easy.

Authentication Sniffing

To get extended rights for the visualization environment B&R has prefabricated widgets which enable logging into the B&R Automation Runtime Role Based Access Control (RBAC). An example of the login widget can be seen in the following figure:

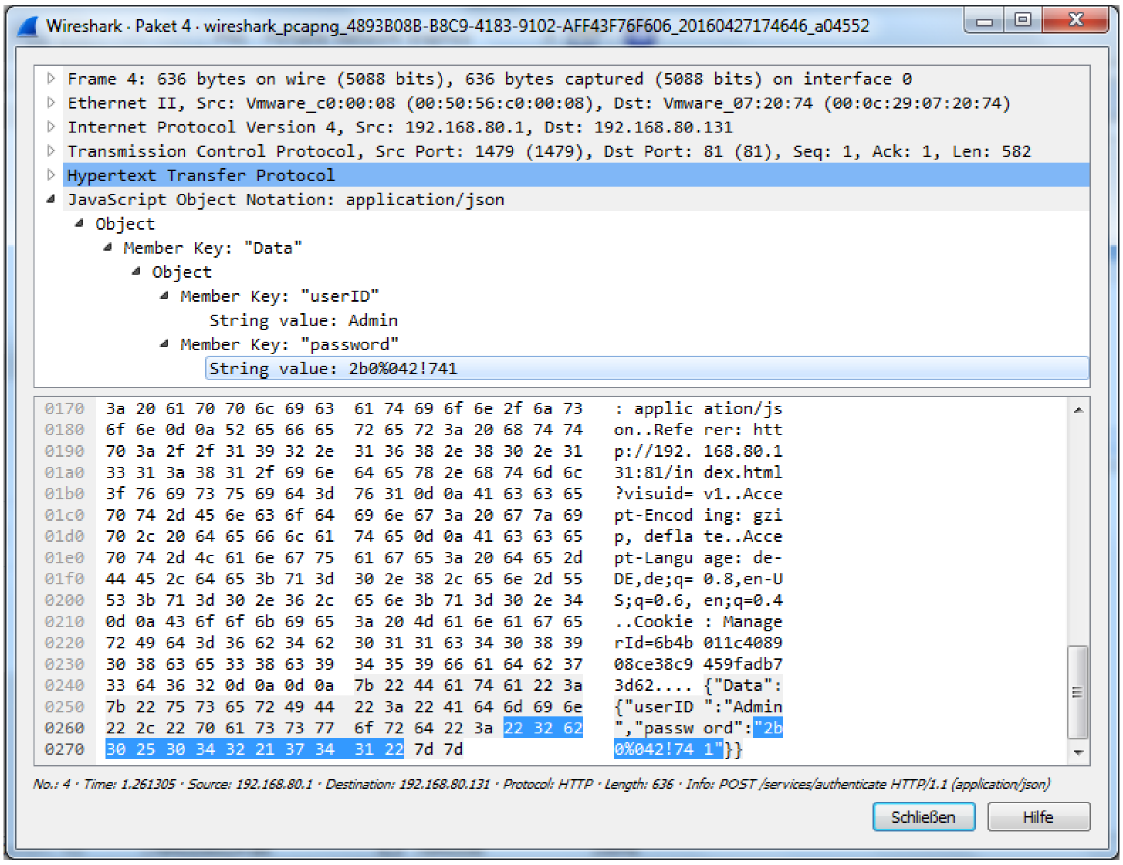
[19] [selfmade diagram by author]



User changed to: Admin

Figure 26 – Mapp View Login Widget Screenshot

By sniffing the corresponding request the attacker is able to see the password in plaintext. No hashing with adding a nonce value is done on the client side as the following figure indicates:



Wireshark · Paket 4 · wireshark_pcapng_4893B08B-B8C9-4183-9102-AFF43F76F606_20160427174646_a04552

```
▷ Frame 4: 636 bytes on wire (5088 bits), 636 bytes captured (5088 bits) on interface 0
▷ Ethernet II, Src: Vmware_c0:00:08 (00:50:56:c0:00:08), Dst: Vmware_07:20:74 (00:0c:29:07:20:74)
▷ Internet Protocol Version 4, Src: 192.168.80.1, Dst: 192.168.80.131
▷ Transmission Control Protocol, Src Port: 1479 (1479), Dst Port: 81 (81), Seq: 1, Ack: 1, Len: 582
▷ Hypertext Transfer Protocol
◢ JavaScript Object Notation: application/json
  ◢ Object
    ◢ Member Key: "Data"
      ◢ Object
        ◢ Member Key: "userID"
            String value: Admin
        ◢ Member Key: "password"
            String value: 2b0%042!741
```

```
0170  3a 20 61 70 70 6c 69 63  61 74 69 6f 6e 2f 6a 73   : applic ation/js
0180  6f 6e 0d 0a 52 65 66 65  72 65 72 3a 20 68 74 74   on..Refe rer: htt
0190  70 3a 2f 2f 31 39 32 2e  31 36 38 2e 38 30 2e 31   p://192. 168.80.1
01a0  33 31 3a 38 31 2f 69 6e  64 65 78 2e 68 74 6d 6c   31:81/in dex.html
01b0  3f 76 69 73 75 69 64 3d  76 31 0d 0a 41 63 63 65   ?visuid= v1..Acce
01c0  70 74 2d 45 6e 63 6f 64  69 6e 67 3a 20 67 7a 69   pt-Encod ing: gzi
01d0  70 2c 20 64 65 66 6c 61  74 65 0d 0a 41 63 63 65   p, defla te..Acce
01e0  70 74 2d 4c 61 6e 67 75  61 67 65 3a 20 64 65 2d   pt-Langu age: de-
01f0  44 45 2c 64 65 3b 71 3d  30 2e 38 2c 65 6e 2d 55   DE,de;q= 0.8,en-U
0200  53 3b 71 3d 30 2e 36 2c  65 6e 3b 71 3d 30 2e 34   S;q=0.6, en;q=0.4
0210  0d 0a 43 6f 6f 6b 69 65  3a 20 4d 61 6e 61 67 65   ..Cookie : Manage
0220  72 49 64 3d 36 62 34 62  30 31 31 63 34 30 38 39   rId=6b4b 011c4089
0230  30 38 63 65 33 38 63 39  34 35 39 66 61 64 62 37   08ce38c9 459fadb7
0240  33 64 36 32 0d 0a 0d 0a  7b 22 44 61 74 61 22 3a   3d62.... {"Data":
0250  7b 22 75 73 65 72 49 44  22 3a 22 41 64 6d 69 6e   {"userID ":"Admin
0260  22 2c 22 70 61 73 73 77  6f 72 64 22 3a 22 32 62   ","passw ord":"2b
0270  30 25 30 34 32 21 37 34  31 22 7d 7d               0%042!74 1"}}
```

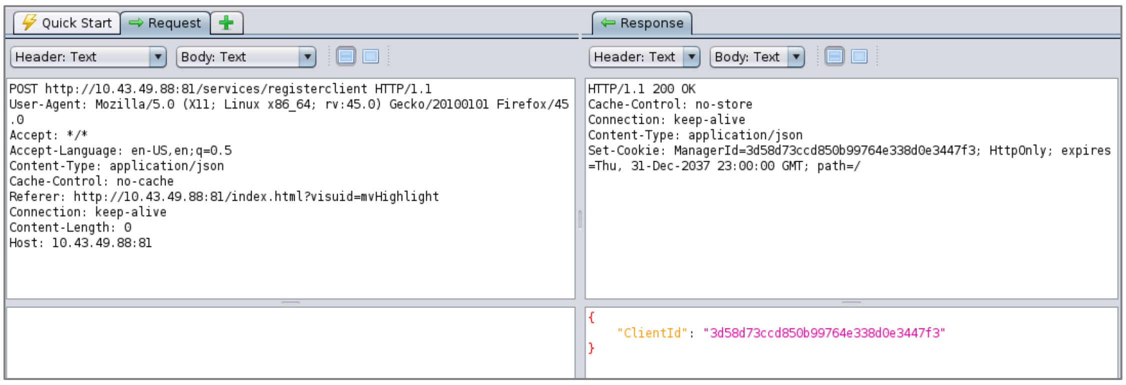
No.: 4 · Time: 1.261305 · Source: 192.168.80.1 · Destination: 192.168.80.131 · Protocol: HTTP · Length: 636 · Info: POST /services/authenticate HTTP/1.1 (application/json)

Figure 27 – Wireshark Capture of a Visualization Login Screenshot

Client Registration

On closer examination of the different HTTP requests, the POST method "registerclient" seemed particularly interesting.

The client requests and gets a cookie with a managerID. This ID is also sent in the message body.


Quick Start | Request | Response
Header: Text | Body: Text

```
POST http://10.43.49.88:81/services/registerclient HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45
.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Content-Type: application/json
Cache-Control: no-cache
Referer: http://10.43.49.88:81/index.html?visuid=mvHighlight
Connection: keep-alive
Content-Length: 0
Host: 10.43.49.88:81
```

```
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: keep-alive
Content-Type: application/json
Set-Cookie: ManagerId=3d58d73ccd850b99764e338d0e3447f3; HttpOnly; expires
=Thu, 31-Dec-2037 23:00:00 GMT; path=/
```

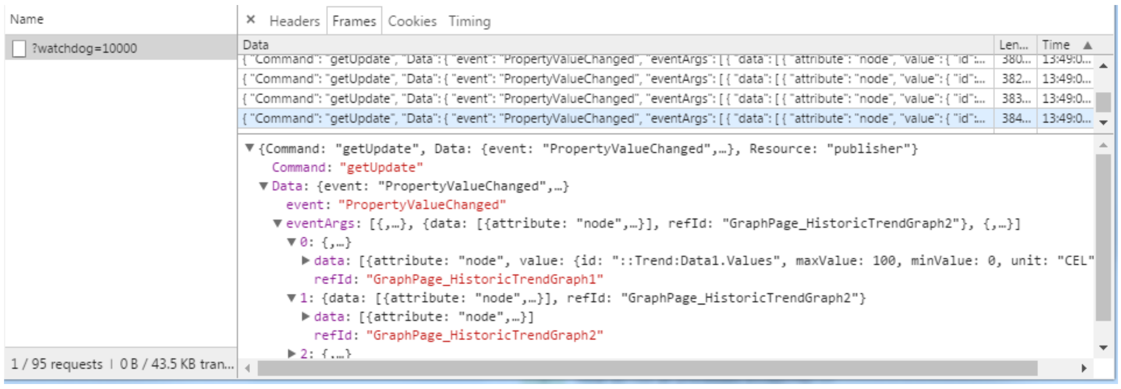
```
{
    "ClientId": "3d58d73ccd850b99764e338d0e3447f3"
}
```

Figure 28 - Registerclient Request ZAP Screenshot

Sending the request more often, the cookies always gets renewed until the maximum number of clients is reached. It has no impact which request (GET or POST) is used. When reaching the maximum number of clients, the visualization is not loaded and a message indicating “Max. number of clients reached!” appears. There was no timeout specified either. The sessions of the malicious clients run indefinitely.

3.4.2.6. **Fuzzing**

ZAP includes an integrated fuzzer and already fuzzes GET and POST request parameters in the deep scanning run with an integrated dictionary. We focus, therefore, on fuzzing the websocket communication with JSON data:


Name: ?watchdog=10000
1 / 95 requests | 0 B / 43.5 KB tran...
Headers | Frames | Cookies | Timing

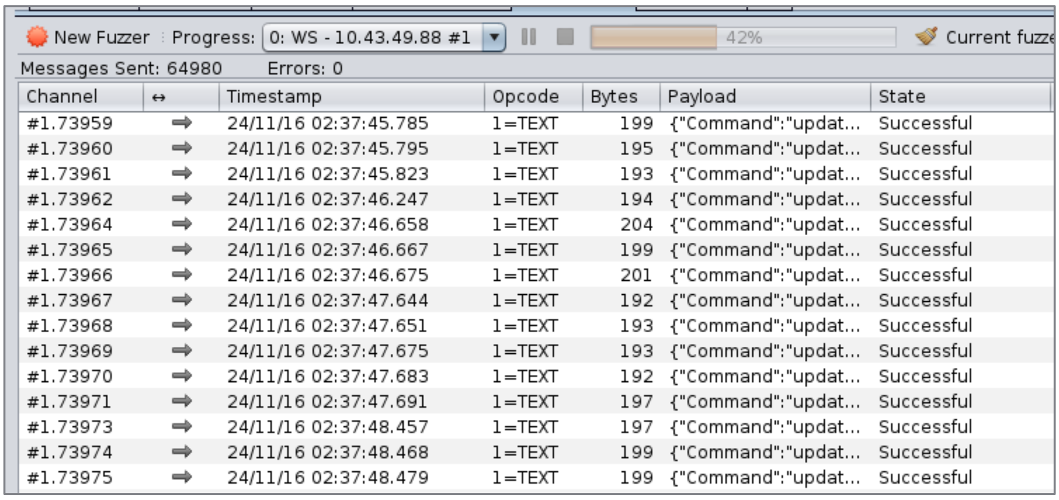
Data	Len...	Time
{ "Command": "getUpdate", "Data": { "event": "PropertyValueChanged", "eventArgs": [{ "data": [{ "attribute": "node", "value": { "id":...	380...	13:49:0...
{ "Command": "getUpdate", "Data": { "event": "PropertyValueChanged", "eventArgs": [{ "data": [{ "attribute": "node", "value": { "id":...	382...	13:49:0...
{ "Command": "getUpdate", "Data": { "event": "PropertyValueChanged", "eventArgs": [{ "data": [{ "attribute": "node", "value": { "id":...	383...	13:49:0...
{ "Command": "getUpdate", "Data": { "event": "PropertyValueChanged", "eventArgs": [{ "data": [{ "attribute": "node", "value": { "id":...	384...	13:49:0...

```
▼{Command: "getUpdate", Data: {event: "PropertyValueChanged",…}, Resource: "publisher"}
   Command: "getUpdate"
  ▼Data: {event: "PropertyValueChanged",…}
     event: "PropertyValueChanged"
    ▼eventArgs: [{,…}, {data: [{attribute: "node",…}], refId: "GraphPage_HistoricTrendGraph2"}, {,…}]
      ▼0: {,…}
        ▶data: [{attribute: "node", value: {id: "::Trend:Data1.Values", maxValue: 100, minValue: 0, unit: "CEL"
         refId: "GraphPage_HistoricTrendGraph1"
      ▼1: {data: [{attribute: "node",…}], refId: "GraphPage_HistoricTrendGraph2"}
        ▶data: [{attribute: "node",…}]
         refId: "GraphPage_HistoricTrendGraph2"
      ▶2: {,…}
```

Figure 29 - JSON Websocket Communication of the Visualization Screenshot

ZAP does not provide a JSON-FileFuzzer per default. In addition to the vanilla ZAP project, the community provides a ZAP extensions project where a JSON fuzzing dictionary was available. [86]

These library test runs with fuzzing the whole JSON message and, with fuzzying only, certain parameters were started. Unfortunately, no deterministic error was found, but the PLC crashes after a few minutes of fuzzing requests (the following figure shows a stopped test run after 64980 requests, when the PLC crashed). It is assumed that there is some buffer overflow in the JSON parser or websocket stack as there is no dedicated logger entry for the failure.



New Fuzzer Progress: 0: WS - 10.43.49.88 #1 42% Current fuzz
Messages Sent: 64980 Errors: 0

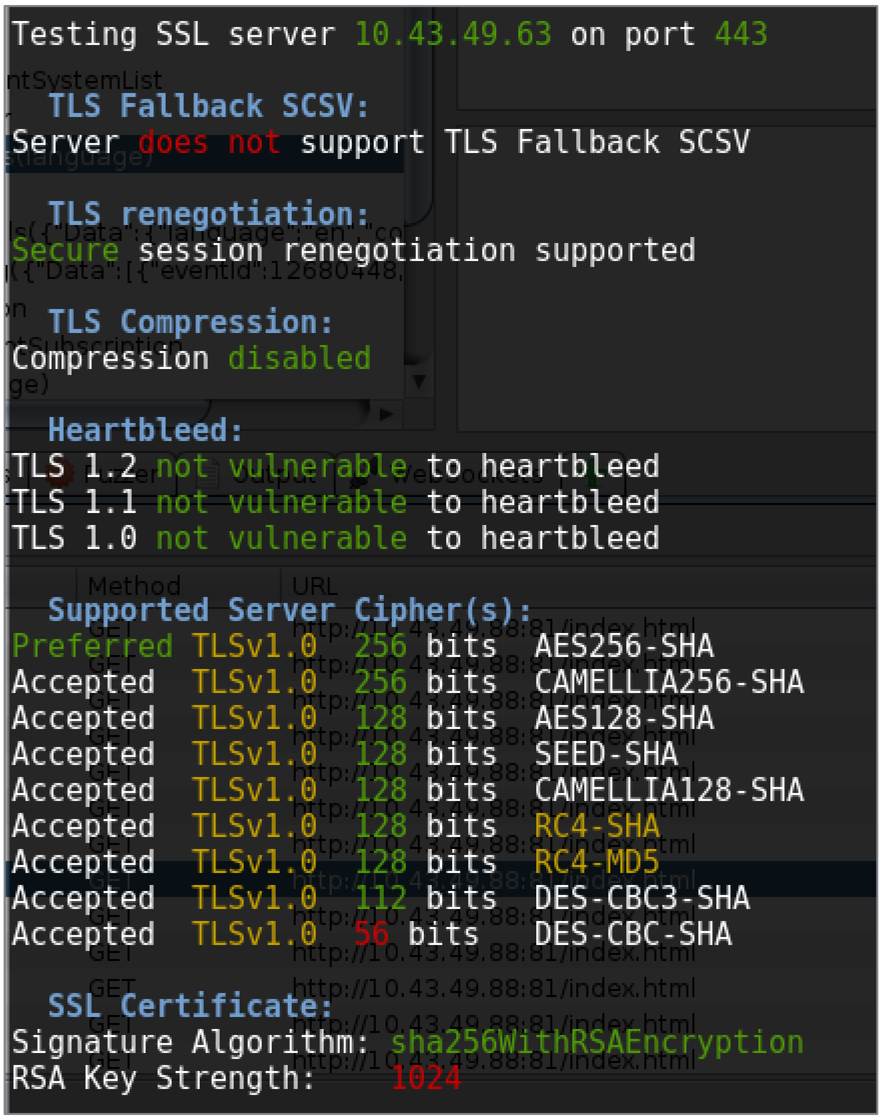
Channel	↔	Timestamp	Opcode	Bytes	Payload	State
#1.73959	⇒	24/11/16 02:37:45.785	1=TEXT	199	{"Command":"updat...	Successful
#1.73960	⇒	24/11/16 02:37:45.795	1=TEXT	195	{"Command":"updat...	Successful
#1.73961	⇒	24/11/16 02:37:45.823	1=TEXT	193	{"Command":"updat...	Successful
#1.73962	⇒	24/11/16 02:37:46.247	1=TEXT	194	{"Command":"updat...	Successful
#1.73964	⇒	24/11/16 02:37:46.658	1=TEXT	204	{"Command":"updat...	Successful
#1.73965	⇒	24/11/16 02:37:46.667	1=TEXT	199	{"Command":"updat...	Successful
#1.73966	⇒	24/11/16 02:37:46.675	1=TEXT	201	{"Command":"updat...	Successful
#1.73967	⇒	24/11/16 02:37:47.644	1=TEXT	192	{"Command":"updat...	Successful
#1.73968	⇒	24/11/16 02:37:47.651	1=TEXT	193	{"Command":"updat...	Successful
#1.73969	⇒	24/11/16 02:37:47.675	1=TEXT	193	{"Command":"updat...	Successful
#1.73970	⇒	24/11/16 02:37:47.683	1=TEXT	192	{"Command":"updat...	Successful
#1.73971	⇒	24/11/16 02:37:47.691	1=TEXT	197	{"Command":"updat...	Successful
#1.73973	⇒	24/11/16 02:37:48.457	1=TEXT	197	{"Command":"updat...	Successful
#1.73974	⇒	24/11/16 02:37:48.468	1=TEXT	199	{"Command":"updat...	Successful
#1.73975	⇒	24/11/16 02:37:48.479	1=TEXT	199	{"Command":"updat...	Successful

Figure 30 - Crash after JSON Fuzzing Requests Screenshot

3.4.2.7. Encryption Analysis

For the web visualization, the communication from the web server to the clients can be encrypted. X509 certificates can be created via the Automation Studio engineering environment and transferred to the PLC. SSL/TLS is then used to encrypt all the data in transit.

Cipher Analysis
The Automation Studio configuration only allows SSL 3.0 or TLS 1.0. In our case TLS 1.0 was active and will be analyzed.
The following figure shows the output of a common SSL/TLS-analyzing tool (TLSSLed)



```
Testing SSL server 10.43.49.63 on port 443

  TLS Fallback SCSV:
Server does not support TLS Fallback SCSV

  TLS renegotiation:
Secure session renegotiation supported

  TLS Compression:
Compression disabled

  Heartbleed:
TLS 1.2 not vulnerable to heartbleed
TLS 1.1 not vulnerable to heartbleed
TLS 1.0 not vulnerable to heartbleed

  Supported Server Cipher(s):
Preferred TLSv1.0  256 bits  AES256-SHA
Accepted  TLSv1.0  256 bits  CAMELLIA256-SHA
Accepted  TLSv1.0  128 bits  AES128-SHA
Accepted  TLSv1.0  128 bits  SEED-SHA
Accepted  TLSv1.0  128 bits  CAMELLIA128-SHA
Accepted  TLSv1.0  128 bits  RC4-SHA
Accepted  TLSv1.0  128 bits  RC4-MD5
Accepted  TLSv1.0  112 bits  DES-CBC3-SHA
Accepted  TLSv1.0   56 bits  DES-CBC-SHA

  SSL Certificate:
Signature Algorithm: sha256WithRSAEncryption
RSA Key Strength:    1024
```

Figure 31 - TLSSLed Report Output Screenshot

There are multiple issues with the supported ciphers, in descending severity:

- Most problematic is the supported usage of the 56bit Data Encryption Standard symmetric-key algorithm. It was developed in the early 1970s and is now, with a key length of 56bit, not up to standard with current available computing power. To run a brute-force method to exhaust the complete keyspace of the DES algorithm online services like http://crack.sh are available. The 48 Xilinx Virtex 6 LX240T FPGAs run through the keyspace in approximately 26 hours, which costs (for a low priority order) 20 US Dollar. Also the calculation on current high-end Graphical Processing Units (GPUs) is possible. The GPU cracker "hashcat" supports RAW DES (KPA) cracking. [87] In tests on a NVIDIA Geforce GTX 1080, the cracker delivers 20 Gk/s. Therefore, brute-force exhaustion of the DES-keyspace also tends to get viable in the consumer sector.
- The RC4 streamcipher has been suspect of cryptanalysis for some years now. Older attacks still would need years or ridiculous amount of sessions to make a

suitable attack. Newer approaches can lead to session-cookie decryption in less than 75 hours. [88]

- Ciphers with a block size of 64 bit (here: DES and 3DES) are prone to birthday attacks. [89] Although this kind of attack also needs a lot of data to be transferred with a current threshold of 785 Gigabytes.

Certificate Analysis

Since the used default 1024bit RSA certificate is too weak, it should be exchanged for a 2048bit or 4096bit certificate. Although no factorization of a 1024bit RSA certificate has been shown in public yet, it is within reach of current cryptanalysis and just a matter of time. [90]

With Automation Studio the generation of up to 4096bit RSA certificates is already possible. Therefore, to switch the provided default certificate to a 4096bit variant should be a simple solution for this issue.

Signature Algorithm Analysis

The usage of SHA-1 is deprecated and very skilled attackers are in able to force collisions of the hash function. [91] Regarding newer research [92], the necessary computing power to craft SHA-1 collisions is decreasing rapidly. The research team reached a first fully practical SHA-1 collision with an effort of 110 GPU years of calculation time. Thus, the usage of the SHA-1 successors SHA-2 or the newer SHA-3 is recommended.

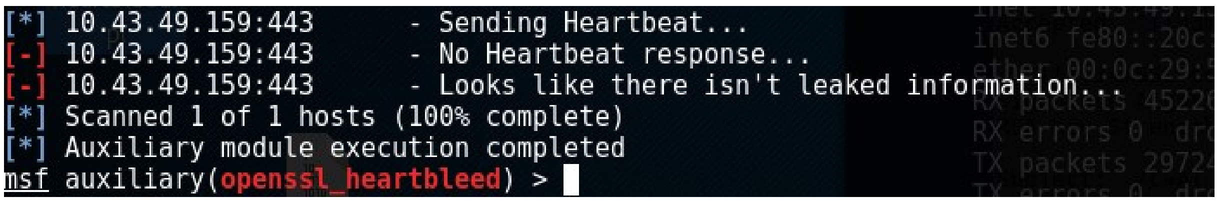
TLS Fallback SCSV

The server does not support the TLS_FALLBACK_SCSV mode, which is leading to a vulnerability that enables a downgrade attack [93] (e.g. from TLS 1.0 to SSL 3.0) on the client. In this case, even though this cannot be exploited as on the server, only one version of SSL/TLS is active and no downgrade is possible.

Heartbleed Attack

Because the library used for the implementation of SSL/TLS is OpenSSL, the vulnerability of the well-known Heartbleed attack was tested.

Though different test-tools resulted in heterogeneous results (sslyze and TLSSLed: no heartbleed vulnerability, Nmap with the script “ssl-heartbleed”: vulnerable to heartbleed), the test with metasploit and the corresponding exploit “openssl-heartbleed” resulted in no possibility to extract data from the server. The failed scan is shown in the figure below:



```
[*] 10.43.49.159:443      - Sending Heartbeat...
[-] 10.43.49.159:443      - No Heartbeat response...
[-] 10.43.49.159:443      - Looks like there isn't leaked information...
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(openssl_heartbleed) >
```

Figure 32 - Metasploit with openssl_heartbleed Module Screenshot

Other Known SSL/TLS Attacks

Well-known SSL/TLS exploits like CRIME, BEAST, BREACH, FREAK or Poodle use downgrade options or are only possible with certain SSL/TLS versions. As they were not applicable further analysis was omitted.

3.4.2.8. Recommendation

Most of the shown security problems can be solved by fixing the underlying code issue. The analysis suggests that no standard off-the-shelf web server is used. Therefore, fuzzing tests on the interfaces are highly recommended to find security issues.
The SSL/TLS library should be updated to enable TLS 1.2 or the upcoming TLS 1.3. If the upgrade is not immediately available, the weak ciphers should be deleted from the available options to omit possible downgrade attacks. The usage of SHA1 as hash function should be omitted, its successor (SHA-2 or the newer SHA-3) should be used.
RSA certificates with 1024bit key length should no longer be created, as 2048bit and 4096bit certificates are already available.

3.4.3. VNC

3.4.3.1. Introduction to VNC

Virtual Network Computing (VNC) is a mechanism to remotely control desktop appliances over an IP network. It uses the Remote Frame Buffer (RFB) protocol to display the graphical user interface of the server system at the client. As the name suggests, it works on the framebuffer level and, therefore, is applicable to all modern windowing systems on various platforms. The automation runtime also includes one of this windowing systems (although an X20 PLC has no display at all) to display the B&R visualization VC4.

RFB basic operations are very simple by using commands like "Put a rectangle of pixel data at the specified X,Y position" [94] In its vanilla configuration this uses a lot of network bandwidth leading to various specified encoding types to improve this behavior:

- RAW (basic)
- Hextile
- Zlib
- Tight-1.0
- Tight-1.1
- Tight-1.1L

In this case VNC is used to extend the legacy visualization VC4 to industrial terminal PCs (like in our topology to the PPC 2100 where a common Windows VNC client is used).

Access to the visualization can lead to full control, especially on smaller machines because no additional user access control mechanism of the visualization is used in small-scale applications.
In some case this is very critical, as the VNC port is directly routed to the internet for remote maintenance purposes.

3.4.3.2. Online Authentication Brute-Force Attack

VNC has some built-in issues that make it vulnerable to certain types of attacks.
First of all, the VNC system does not include the authorization via a combination of username and password, but rather just with the password. Per default VNC distinguishes between two types of user input: a view mode with no possibility to send control information (like keyboard or mouse input) and a full-control mode where input signals are processed. Therefore, two types of passwords are configured on the VNC server and depending on which one is used to login, a user can either take control, or view the visualization.
This design makes the system very vulnerable to brute-force attacks, as the attacker does not have to guess a valid username before trying to find a corresponding password.
Different types of command-line tools are available to brute-force VNC server connection attempts with either dictionary attacks or string generators.

A sample collection of these command-line tools are:

- medusa
- thc-hydra
- patator
- ncrack

The best results were made with medusa, parallelizing authentication attempts and minimalizing challenge-response delays with over 219 keys per second. Lower wait times or more threads lead to PLC errors (which is also a valid Denial-of-Service attack).

With a four character password (which is default in many example applications for convenience reasons) containing upper and lowercase characters, this attempt would brute-force the entire keyspace (with 7454980 possible combinations) in approximately 9.5 hours. Adding numeric values would add up to approximately 19 hours (with 15018570 possible combinations) of brute-force attempts until the keyspace is exhausted. Please note: on average, half the time is needed as going through the whole keyspace to find one password for brute-force attacks. It is assumed that we want to find the password to control the visualization (to find one of the two passwords used will take less time in average).

It has to be taken into account that this is an idealized experiment, as the brute forcing device was directly attached to the same network as the PLC with full bandwidth and the character-set was minimalized. However, with enough time, a weak password and no countermeasures in place, this is a very potent attack to gain access to the VNC visualization.

To aggravate this matter, the used VNC version only allows a maximum character set of eight. Longer passwords are truncated. This is due to a limitation in the RFB protocol and will not be changed for compatibility reasons.
Additionally, the automation studio configuration limits the possible inputs to alphanumeric characters. This leads to a maximum keyspace of 221919451578090 passwords. For the online attack this should be sufficient, as it would take more than 32111 years, to exhaust it with the given rate of 219 keys per second. In an offline attack scenario this is problematic, as the following issue explains.

3.4.3.3. Authentication Sniffing and Offline Brute-Force Attack

The RFB protocol was not designed with any security aspects in mind. Although it does not use plaintext-transportation for the authentication credentials the process works in a very simple manner:

1) The server sends a random 16-byte challenge
2) The client encrypts the challenge with DES, using a password supplied by the user as the key, and sends the resulting 16-byte response

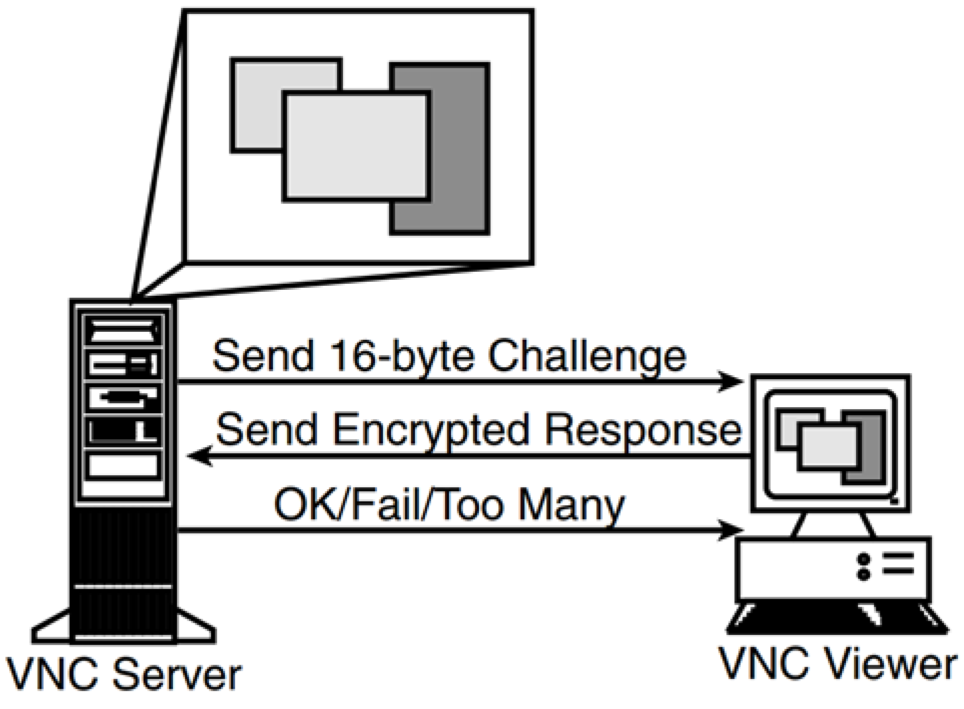


Figure 33 - The VNC Authentication Challenge-Response[20]

If a malicious attacker is able to sniff the handshake of a client logging into the VNC server and gets a valid pair of the challenge and the encrypted response, an offline attack can be made.
The challenge is only bit-flipped and encrypted with a single round of DES (Data Encryption Standard). A brute-force tool can do the same calculation with passwords from a dictionary or a password generator leading to much faster exhaustion of the keyspace than in the seen online attack. Depending on the given hardware and calculation on CPU or GPU (as modern crackers calculate passwords on GPU shader units), it is possible to calculate millions of keys per second and drastically reduce the time needed to exhaust the keyspace or to go through dictionaries.
In a tryout of the attack, it was shown that with a 4 character password and a system capable of calculating 400000 keys per second, the brute-force attempt lasted only seconds until the correct password was ascertained.

3.4.3.4. Other VNC Weaknesses

VNC Challenge Randomness:
As Tipton and Krause described, the real randomness of the provided challenges is questionable:
"The random challenge is generated using the rand(3) function in the C programming language to generate random numbers. The random number generator is initialized using the system clock and the current system time. However, the 16-byte challenge is created by successive calls to the random number generator, decreasing the level of randomness on each call. (Each call returns 1 byte or 8 bits of data.) This makes the challenge predictable and increases the chance an attacker could establish a session by storing all captured responses and their associated challenges. Keeping track of each challenge-response pair can be difficult and […] not necessary."[21]

Man-in-the-Middle Attack:
Due to the lack of timeouts and integrity checking of the RFB protocol, there is an easy approach to a man-in-the-middle attack if an attacker has the possibility to get between client and server in the switched network (e.g. by mac-spoofing). If an attacker initiates a connection to a VNC server, the 16-byte answer challenge is sent as seen in the figure below.

[20] Taken from: Tipton F. and Krause M.: Information Security Management Handbook, Fifth Edition, 2004, CRC Press
[21] Tipton F. and Krause M.: Information Security Management Handbook, Fifth Edition, 2004, CRC Press

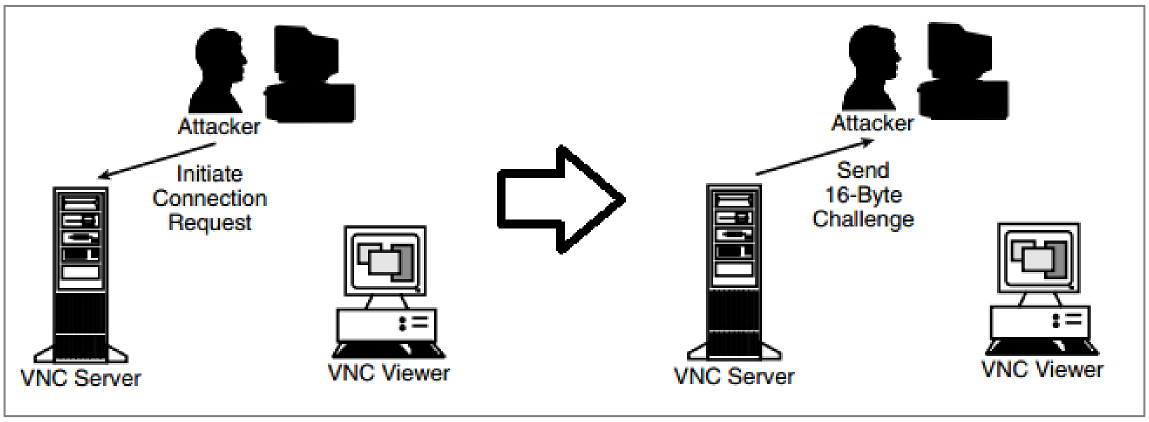


Figure 34 - VNC Man-in-the-Middle Attack - Preparation[22]

The attacker now has a session pending at the server and there is no timeout specified in the protocol standard. The attacker now waits until another VNC client wants to connect to the VNC server.
As seen in the figure below, the attacker can then intercept the server response with the challenge and send the challenge it received earlier. The attacker then only has to sniff the encrypted response by the legitimate client and resend it to the server.
The VNC server will only authenticate the attacker, because he/she sent the correct response to the challenge provided. The legitimate client will not be granted access, as the sent response does not fit to the challenge the server sent to this client earlier.

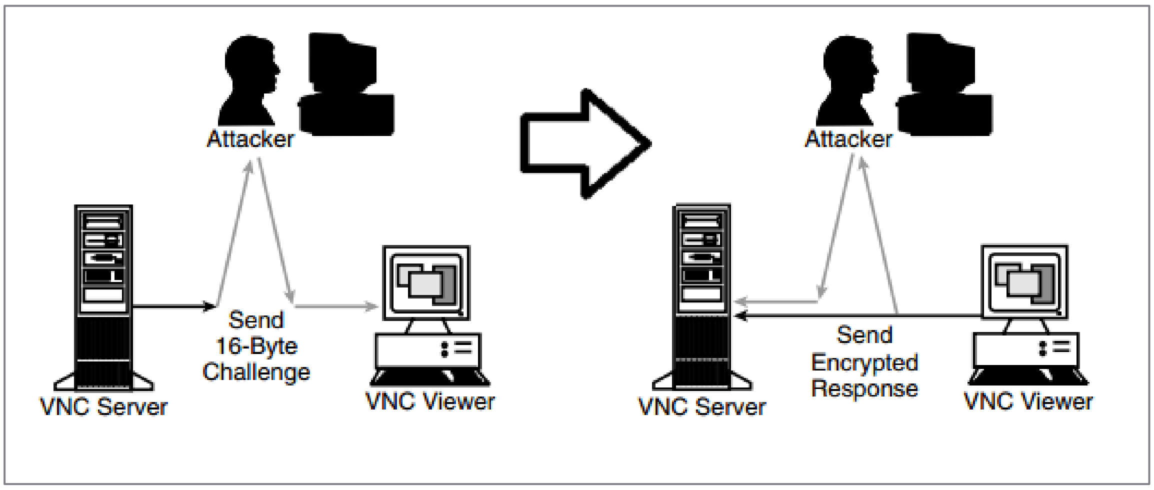


Figure 35 - VNC Man-in-the-Middle Attack - Execution[23]

RFB Extensions:
Also further analysis showed that additional data is transported via the RFB protocol. These RFB extensions are used to control hardware peripheral located on the terminal. By using a man-in-the-middle attack and tampering with the values of, for example,

[22] Taken and adapted from: Tipton F. and Krause M.: Information Security Management Handbook, Fifth Edition, 2004, CRC Press
[23] Taken and adapted from: Tipton F. and Krause M.: Information Security Management Handbook, Fifth Edition, 2004, CRC Press

hardware buttons, a malicious user could gain direct access to machine controls. A deeper analysis of this problem could be investigated in follow-up research.

3.4.3.5. Recommendation

As the design of VNC and the RFB protocol suffers from security weaknesses which cannot be addressed without breaking compatibility, it is hard to give proper recommendations for this topic.
There are specialized versions of VNC clients and servers which support advanced authentication and encryption options without being compatible to other vendors. B&R could implement the server extensions of these versions (e.g. UltraVNC or RealVNC) and use the according clients.
Another approach would be to tunnel the insecure VNC connection through an extra security layer, implementing stronger authentication and encryption. An example could be an SSH tunnel or a VPN implementation like IPsec.

3.4.4. OPC-UA

3.4.4.1. Introduction to OPC-UA

OPC-UA is a new industrial interoperability standard for communication in automation environments.
The organization behind the technology, the OPC Foundation defines it as following:
“The OPC Unified Architecture (UA), released in 2008, is a platform independent service-oriented architecture that integrates all the functionality of the individual OPC Classic specifications into one extensible framework.
This multi-layered approach accomplishes the original design specification goals of:

- Functional equivalence: all COM OPC Classic specifications are mapped to UA
- Platform independence: from an embedded micro-controller to cloud-based infrastructure
- Secure: encryption, authentication, and auditing
- Extensible: ability to add new features without affecting existing applications
- Comprehensive information modeling: for defining complex information” [95]

B&R uses this technology as an interface for the data of the PLC. The automation runtime implements a standard OPC-UA Server and publishes information like the state of the PLC, the process variables, etc.
A very popular use case is to connect PLCs of different vendors by OPC-UA, so there is no need for one vendor to implement the proprietary protocols of another vendor.

Another use case here is that the new web visualization of B&R (mapp View) internally communicates only with the OPC-UA server. So from an architectural point of view, it would be easy to separate the visualization server from the rest of the PLC.

3.4.4.2. OPC-UA Security Analysis

The OPC Foundation summarizes the security aspects of OPC-UA as following:
“One of the most important considerations in choosing a technology is security. OPC UA is firewall-friendly while addressing security concerns by providing a suite of controls:

- Transport: numerous protocols are defined providing options such as the ultra-fast OPC-binary transport or the more universally compatible SOAP-HTTPS, for example
- Session Encryption: messages are transmitted securely at 128 or 256 bit encryption levels
- Message Signing: messages are received exactly as they were sent
- Sequenced Packets: exposure to message replay attacks is eliminated with sequencing
- Authentication: each UA client and server is identified through OpenSSL certificates providing control over which applications and systems are permitted to connect with each other
- User Control: applications can require users to authenticate (login credentials, certificate, etc.) and can further restrict and enhance their capabilities with access rights and address-space “views”
- Auditing: activities by user and/or system are logged providing an access audit trail” [96]

This security review focuses on the binary based transport mode, because the B&R Automation Runtime only implements this mode to transfer data.

3.4.4.2.a Excerpt: Configuration

The B&R OPC-UA implementation supports three different modes:

- securityMode: None
- securityMode: Sign
- securityMode: SignAndEncrypt

With the first mode “None”, no additional security measures are taken and the protocol delivers its information as plaintext. Minimal security measures are taken to prevent simple exploitation like replay attacks (e.g. a sequence number).
The other modes include confirmation of the communication partner’s identity and guarantee the integrity of the sent data. The difference between Sign and SignAndEncrypt is the added confidentiality of the data. With SignAndEnrypt every communication is additionally encrypted to add security against eavesdropping.

The B&R OPC-UA server supports the following modes:

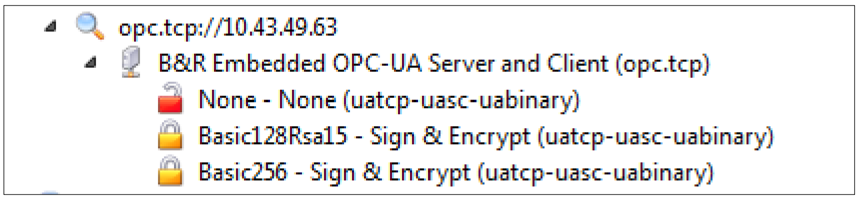

opc.tcp://10.43.49.63
B&R Embedded OPC-UA Server and Client (opc.tcp)
None - None (uatcp-uasc-uabinary)
Basic128Rsa15 - Sign & Encrypt (uatcp-uasc-uabinary)
Basic256 - Sign & Encrypt (uatcp-uasc-uabinary)

Figure 36 - B&R OPC-UA Server Modes Screenshot

The most secure mode “Basic256Sha256” defined in the OPC-UA standard is not supported.

3.4.4.2.b Encryption

The Basic128Rsa15 setting uses AES 128 bit in CBC mode for symmetric encryption. Though AES 256 would be better and a theoretical bicycle attack for AES 128 reduces the brute-force complexity to $2^{126.0}$, the ability to brute-force AES 128 is still not possible even for very powerful adversaries. [97]
The Basic256 setting uses AES in 256 bit with CBC mode for the symmetric encryption of the communication.
As a signing function in both configuration settings, SHA1 is used, which is not inherently bad, however SHA256 is recommended, as the cost to form a collision with SHA1 is becoming affordable for very powerful attackers, as mentioned earlier in the encryption analysis of the web server. [91][92]

3.4.4.3. Recommendation

To ensure a secure communication via the OPC-UA channel, it is vital to set the securityMode to Sign or SignAndEncrypt mode. If there is a need for confidentiality, then only the SignAndEncrypt mode is best, because it encrypts all the sent data.

Regarding the cryptographic algorithms the usage of SHA-1 should be omitted. The security policy “Basic256Sha256” is the most secure algorithm used and should be favored if technically possible.

Anonymous authentication should not be used as there is no option to identify an attacker login with anonymous credentials. If anonymous access is needed, only readable access rights should be given. Even if no securityMode option is possible, authentication methods with credentials should always be used.

3.4.5. Proprietary Protocols

3.4.5.1. Introduction to INA/ANSL

INA and ANSL are the two basic proprietary protocols used by B&R to communicate machine data (e.g. process variables) and machine control information to a PLC.

The first scenario is the communication between PLCs to exchange information regarding the automation process. This could be to operate different parts of a very complex machine using more than one PLC to control it.
Another use case would be to use redundant PLCs in a high-safety demanding environment.
In a special transport mode, this could also be used to tunnel communication not directly targeted to the PLC, but to the next PLC which is directly connected. In essence, this is used in environments where no additional cabling to the offset PLC is available and the fieldbus (e.g. powerlink) has to be used.

Also a common use case of PLC to PLC communication is the usage of a terminal with the legacy visualization VC4. This is still widely used because the web-based visualization has just entered the market.

The second possibility is the communication between the engineering environment (Automation Studio) and the PLC. Here the protocols are not primarily used to transfer machine data (though possible for monitoring or debugging purposes), but to control the PLC, e.g. to apply configuration settings, download images and load programs.

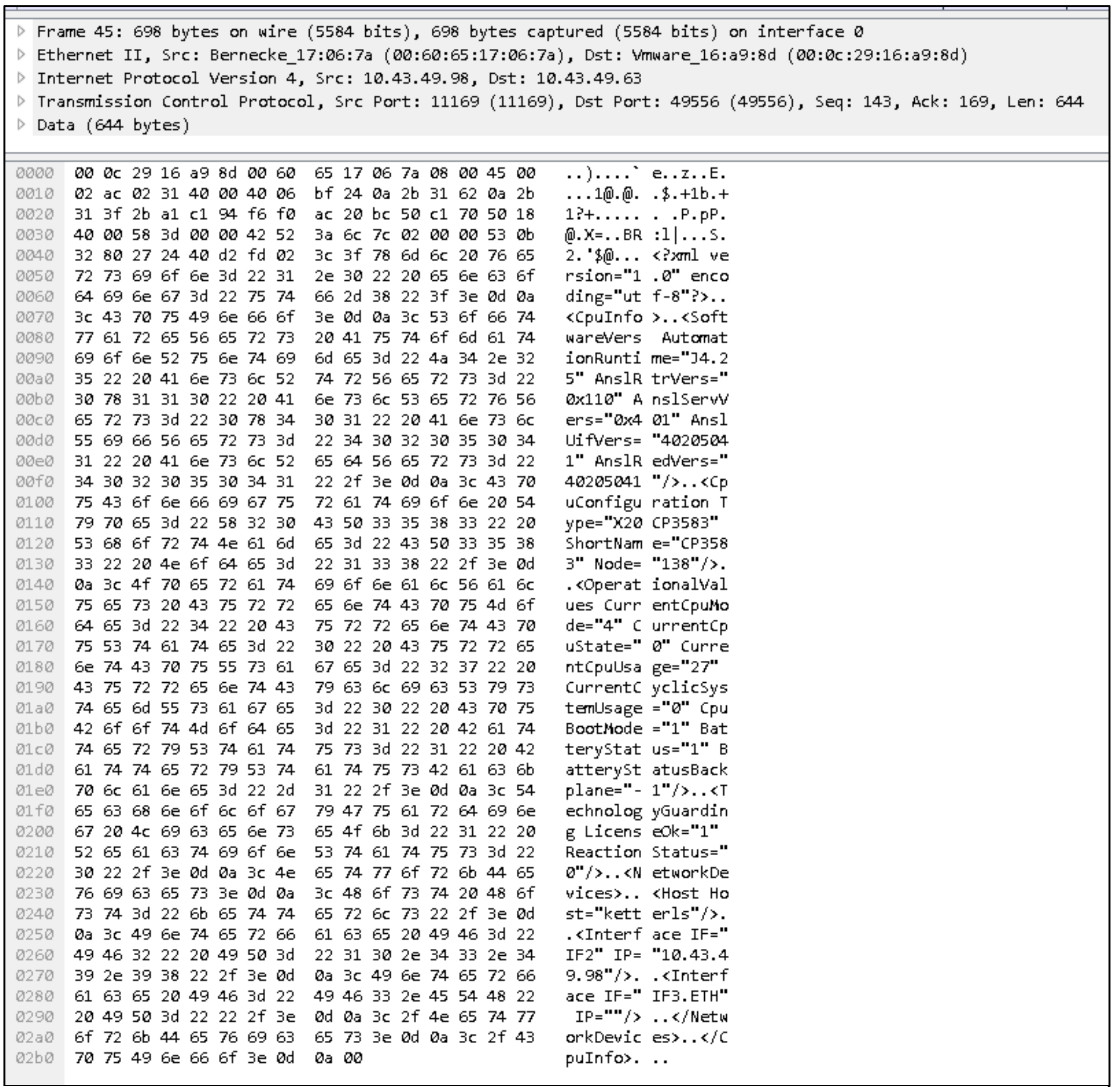
The third option is the communication of PLCs with third party devices. B&R provides libraries for common frameworks like .NET to enable communications with a wide variety of platforms. A popular use case is to connect a centralized monitoring station of a SCADA environment to the PLC. The process variables can then be monitored and stored in a database on the monitoring device for further data operations.

3.4.5.2. Security Analysis

By investigation of the packets sent on port 11159 and 11169 on the network in various operating modes (PLC to PLC, PLC to Automation Studio), the complete lack of authentication is visible.
Any malicious operator that can connect to the stated ports on the PLC is able to send the various possible commands. The only immediate problem is the format of the protocol, since the commands are not directly visible and also no browsing possibility is available. This approach could be identified as security by obscurity.
With little knowledge of the protocol or by observation of the changed data in the packets, it would be easily possible to inject malicious data into process variables possibly even leading to damage of the controlled machine.

Two examples of problematic scenarios are illustrated below.

A captured package with plaintext information of the PLC:

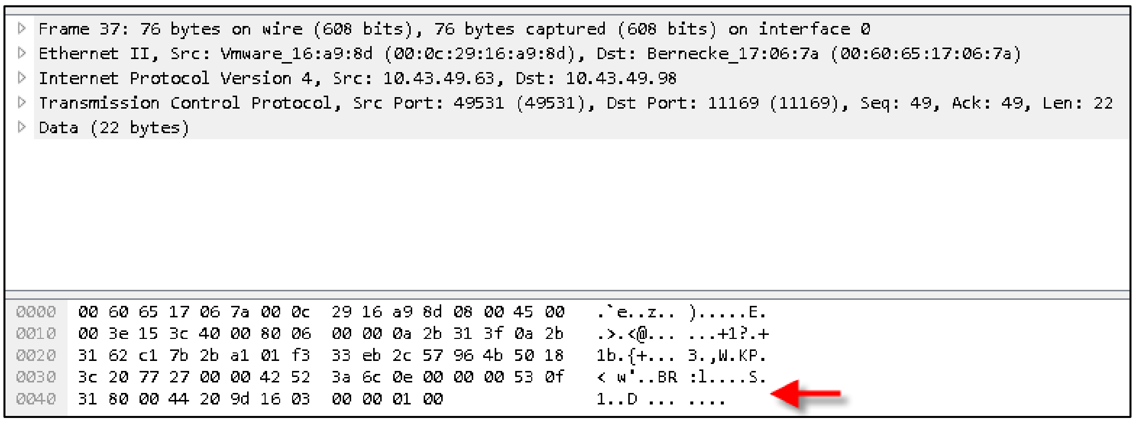


```
▷ Frame 45: 698 bytes on wire (5584 bits), 698 bytes captured (5584 bits) on interface 0
▷ Ethernet II, Src: Bernecke_17:06:7a (00:60:65:17:06:7a), Dst: Vmware_16:a9:8d (00:0c:29:16:a9:8d)
▷ Internet Protocol Version 4, Src: 10.43.49.98, Dst: 10.43.49.63
▷ Transmission Control Protocol, Src Port: 11169 (11169), Dst Port: 49556 (49556), Seq: 143, Ack: 169, Len: 644
▷ Data (644 bytes)

0000  00 0c 29 16 a9 8d 00 60  65 17 06 7a 08 00 45 00   ..)....` e..z..E.
0010  02 ac 02 31 40 00 40 06  bf 24 0a 2b 31 62 0a 2b   ...1@.@. .$.+1b.+
0020  31 3f 2b a1 c1 94 f6 f0  ac 20 bc 50 c1 70 50 18   1?+..... . .P.pP.
0030  40 00 58 3d 00 00 42 52  3a 6c 7c 02 00 00 53 0b   @.X=..BR :l|...S.
0040  32 80 27 24 40 d2 fd 02  3c 3f 78 6d 6c 20 76 65   2.'$@... <?xml ve
0050  72 73 69 6f 6e 3d 22 31  2e 30 22 20 65 6e 63 6f   rsion="1 .0" enco
0060  64 69 6e 67 3d 22 75 74  66 2d 38 22 3f 3e 0d 0a   ding="ut f-8"?>..
0070  3c 43 70 75 49 6e 66 6f  3e 0d 0a 3c 53 6f 66 74   <CpuInfo >..<Soft
0080  77 61 72 65 56 65 72 73  20 41 75 74 6f 6d 61 74   wareVers  Automat
0090  69 6f 6e 52 75 6e 74 69  6d 65 3d 22 4a 34 2e 32   ionRunti me="J4.2
00a0  35 22 20 41 6e 73 6c 52  74 72 56 65 72 73 3d 22   5" AnslR trVers="
00b0  30 78 31 31 30 22 20 41  6e 73 6c 53 65 72 76 56   0x110" A nslServV
00c0  65 72 73 3d 22 30 78 34  30 31 22 20 41 6e 73 6c   ers="0x4 01" Ansl
00d0  55 69 66 56 65 72 73 3d  22 34 30 32 30 35 30 34   UifVers= "4020504
00e0  31 22 20 41 6e 73 6c 52  65 64 56 65 72 73 3d 22   1" AnslR edVers="
00f0  34 30 32 30 35 30 34 31  22 2f 3e 0d 0a 3c 43 70   40205041 "/>..<Cp
0100  75 43 6f 6e 66 69 67 75  72 61 74 69 6f 6e 20 54   uConfigu ration T
0110  79 70 65 3d 22 58 32 30  43 50 33 35 38 33 22 20   ype="X20 CP3583"
0120  53 68 6f 72 74 4e 61 6d  65 3d 22 43 50 33 35 38   ShortNam e="CP358
0130  33 22 20 4e 6f 64 65 3d  22 31 33 38 22 2f 3e 0d   3" Node= "138"/>.
0140  0a 3c 4f 70 65 72 61 74  69 6f 6e 61 6c 56 61 6c   .<Operat ionalVal
0150  75 65 73 20 43 75 72 72  65 6e 74 43 70 75 4d 6f   ues Curr entCpuMo
0160  64 65 3d 22 34 22 20 43  75 72 72 65 6e 74 43 70   de="4" C urrentCp
0170  75 53 74 61 74 65 3d 22  30 22 20 43 75 72 72 65   uState=" 0" Curre
0180  6e 74 43 70 75 55 73 61  67 65 3d 22 32 37 22 20   ntCpuUsa ge="27"
0190  43 75 72 72 65 6e 74 43  79 63 6c 69 63 53 79 73   CurrentC yclicSys
01a0  74 65 6d 55 73 61 67 65  3d 22 30 22 20 43 70 75   temUsage ="0" Cpu
01b0  42 6f 6f 74 4d 6f 64 65  3d 22 31 22 20 42 61 74   BootMode ="1" Bat
01c0  74 65 72 79 53 74 61 74  75 73 3d 22 31 22 20 42   teryStat us="1" B
01d0  61 74 74 65 72 79 53 74  61 74 75 73 42 61 63 6b   atterySt atusBack
01e0  70 6c 61 6e 65 3d 22 2d  31 22 2f 3e 0d 0a 3c 54   plane="- 1"/>..<T
01f0  65 63 68 6e 6f 6c 6f 67  79 47 75 61 72 64 69 6e   echnolog yGuardin
0200  67 20 4c 69 63 65 6e 73  65 4f 6b 3d 22 31 22 20   g Licens eOk="1"
0210  52 65 61 63 74 69 6f 6e  53 74 61 74 75 73 3d 22   Reaction Status="
0220  30 22 2f 3e 0d 0a 3c 4e  65 74 77 6f 72 6b 44 65   0"/>..<N etworkDe
0230  76 69 63 65 73 3e 0d 0a  3c 48 6f 73 74 20 48 6f   vices>.. <Host Ho
0240  73 74 3d 22 6b 65 74 74  65 72 6c 73 22 2f 3e 0d   st="kett erls"/>.
0250  0a 3c 49 6e 74 65 72 66  61 63 65 20 49 46 3d 22   .<Interf ace IF="
0260  49 46 32 22 20 49 50 3d  22 31 30 2e 34 33 2e 34   IF2" IP= "10.43.4
0270  39 2e 39 38 22 2f 3e 0d  0a 3c 49 6e 74 65 72 66   9.98"/>. .<Interf
0280  61 63 65 20 49 46 3d 22  49 46 33 2e 45 54 48 22   ace IF=" IF3.ETH"
0290  20 49 50 3d 22 22 2f 3e  0d 0a 3c 2f 4e 65 74 77    IP=""/> ..</Netw
02a0  6f 72 6b 44 65 76 69 63  65 73 3e 0d 0a 3c 2f 43   orkDevic es>..</C
02b0  70 75 49 6e 66 6f 3e 0d  0a 00                     puInfo>. ..
```

Figure 37 - ANSL Plaintext Capture File Screenshot

Capture of a (with netcat) crafted package including a command to warm restart the PLC:



```
▷ Frame 37: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0
▷ Ethernet II, Src: Vmware_16:a9:8d (00:0c:29:16:a9:8d), Dst: Bernecke_17:06:7a (00:60:65:17:06:7a)
▷ Internet Protocol Version 4, Src: 10.43.49.63, Dst: 10.43.49.98
▷ Transmission Control Protocol, Src Port: 49531 (49531), Dst Port: 11169 (11169), Seq: 49, Ack: 49, Len: 22
▷ Data (22 bytes)

0000  00 60 65 17 06 7a 00 0c  29 16 a9 8d 08 00 45 00   .`e..z.. ).....E.
0010  00 3e 15 3c 40 00 80 06  00 00 0a 2b 31 3f 0a 2b   .>.<@... ...+1?.+
0020  31 62 c1 7b 2b a1 01 f3  33 eb 2c 57 96 4b 50 18   1b.{+... 3.,W.KP.
0030  3c 20 77 27 00 00 42 52  3a 6c 0e 00 00 00 53 0f   < w'..BR :l....S.
0040  31 80 00 44 20 9d 16 03  00 00 01 00               1..D ... ....
```

Figure 38 - ANSL Crafted Package Capture File Screenshot

3.4.5.3. Recommendation

The first measure should be to implement any kind of authentication into the proprietary protocols. A simple user and password authentication mechanism with usage of hashing and a nonce value would be a basic approach to contradict the mentioned attacks.
Additionally, the usage of Transport Layer Security (with SSL/TLS, or respectively DTLS for UDP) would ensure privacy and integrity of the connecting parties.

3.4.6. SNMP

3.4.6.1. Introduction to SNMP

SNMP or Simple Network Management Protocol is a standardized communication method of monitoring and configuring devices in an Ethernet network. The protocol communication is handled via UDP ports 161 and 162 on the IP layer.
The basic architecture behind SNMP is very simple. A management station collects all the necessary data of the network by polling (via port 161) so-called SNMP agents. In special cases (like if a critical error occurs), the agents can also self-trigger so-called SNMP traps (via port 162) to the management station without being polled.

A more comprehensive list of SNMP abilities to help managing a network is provided by [98]:

- "Provide Read/Write abilities – for example you could use it to reset passwords remotely, or re-configure IP addresses.
- Collect information on how much bandwidth is being used.
- Collect error reports into a log, useful for troubleshooting and identifying trends.
- Email an alert when your server is low on disk space.
- Monitor your servers' CPU and Memory use, alert when thresholds are exceeded.
- Page or send an SMS text-message when a device fails.
- Can perform active polling, i.e. Monitoring station asks devices for status every few minutes.
- Passive SNMP – devices can send alerts to a monitoring station on error conditions."

SNMP uses a MIB tree to publish information in the network. Each entry in the tree can be identified by an object identifier (OID) and are defined using a subset of the Abstract Syntax Notation One (ASN.1).

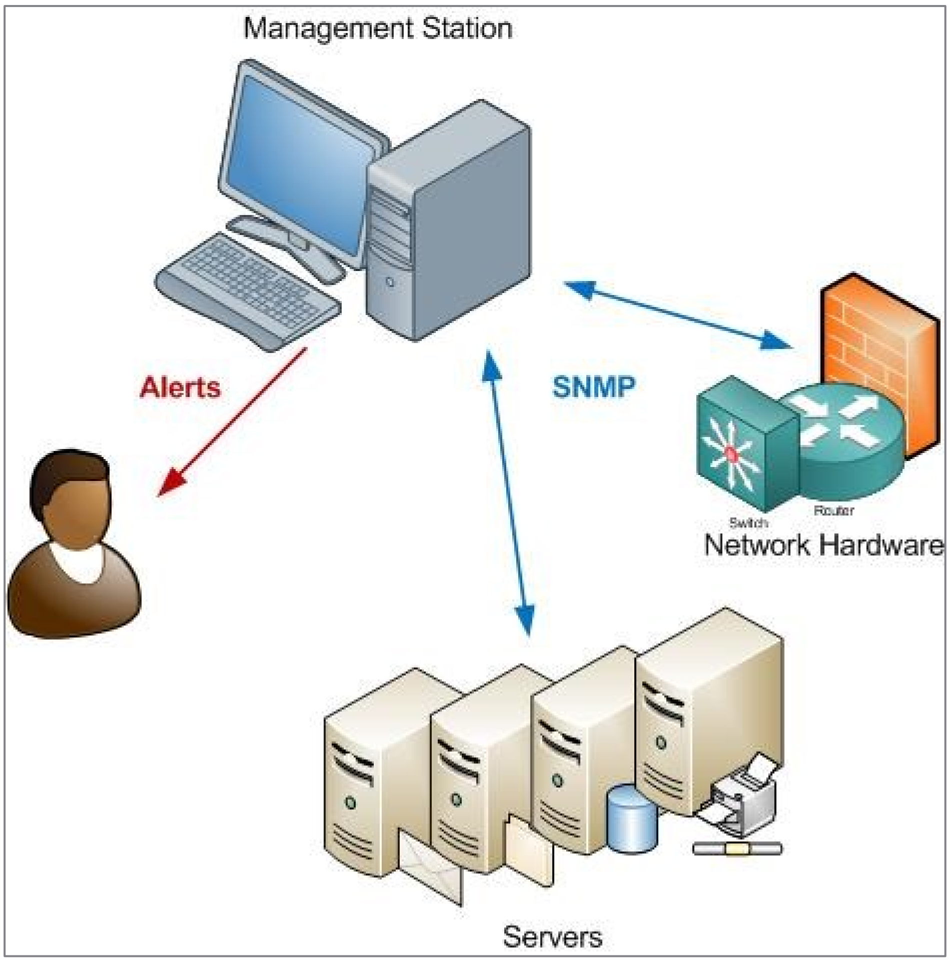


Figure 39 - SNMP Topology[24]

Different versions of SNMP are available:

- SNMP v1
- Secure SNMP
- SNMPv2p
- SNMPv2u
- SNMPv2c
- SNMP v3

Only three of these versions are common nowadays: v1, v2c (being the de facto v2) and v3.
Regarding security, only version 3 is still viable, though rarely used because of the complexity the security aspects demand (e.g. key-infrastructure).

3.4.6.2. Security Analysis

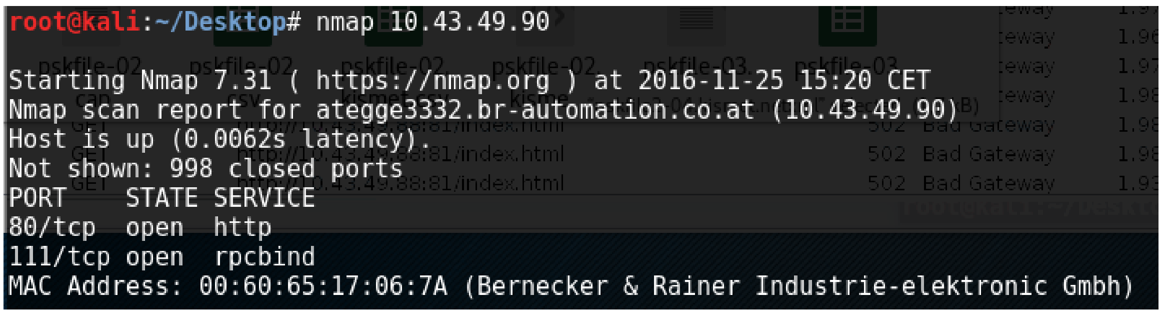
By starting the analysis of the SNMP agent of the PLC there was no communication possible on either UDP port 161 or 162 (although port 161 was open and listening on the PLC).

[24] Taken from: Leskiw A.: SNMP Basics: What is SNMP & How do I use it?, available at http://www.networkmanagementsoftware.com/snmp-tutorial/

A long-lasting traffic analysis demonstrated SNMP communication. However, this took place on the Ethernet layer, not on the IP layer of the network. In RFC 4789 “SNMP over IEEE 802” it is described how to send SNMP messages via the payload of the IEEE 802.3 LAN MAC frames. The security considerations are the same as with SNMP over IP. Although the Ethernet packets identify themselves as SNMPv3, observation has shown that no form of authentication is required.

What is troubling here is the possibility to set certain values, especially the network settings of the PLC.
As no form of authentication is required, a malicious attacker with access to the same (non-routed) LAN network as the PLC could manipulate the network settings of the PLC during runtime just by knowledge of the MAC-address. This would render all IP-based communication to the PLC useless.
The following example shows how easy such an attack can be performed.

Before the attack the PLC is available at 10.43.49.90:

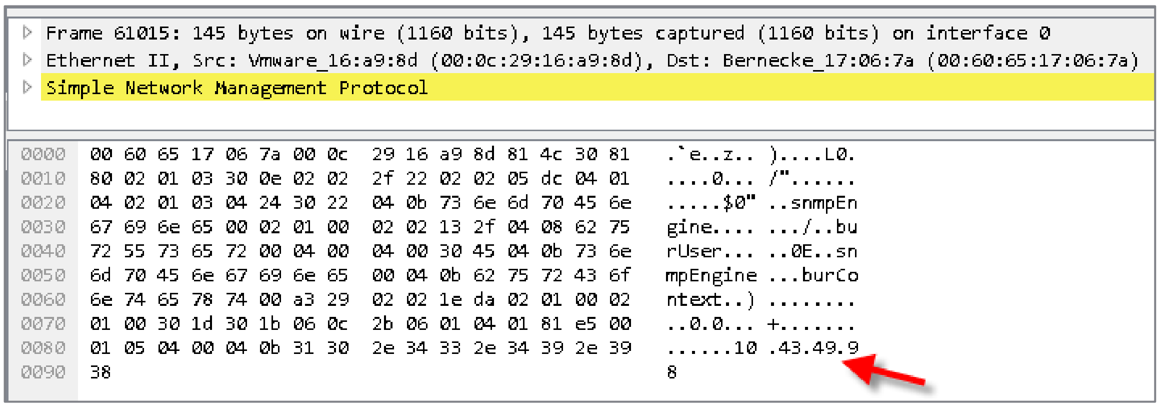


```
root@kali:~/Desktop# nmap 10.43.49.90

Starting Nmap 7.31 ( https://nmap.org ) at 2016-11-25 15:20 CET
Nmap scan report for ategge3332.br-automation.co.at (10.43.49.90)
Host is up (0.0062s latency).
Not shown: 998 closed ports
PORT    STATE SERVICE
80/tcp  open  http
111/tcp open  rpcbind
MAC Address: 00:60:65:17:06:7A (Bernecker & Rainer Industrie-elektronic Gmbh)
```

Figure 40 - NMAP Fingerprint of the PLC before the Crafted SNMP Packet Attack Screenshot

With “packETH”, a packet generation tool for Ethernet frames, we create a malicious SNMP Ethernet packet to set the IP address to 10.43.49.98. All other network settings stay the same.

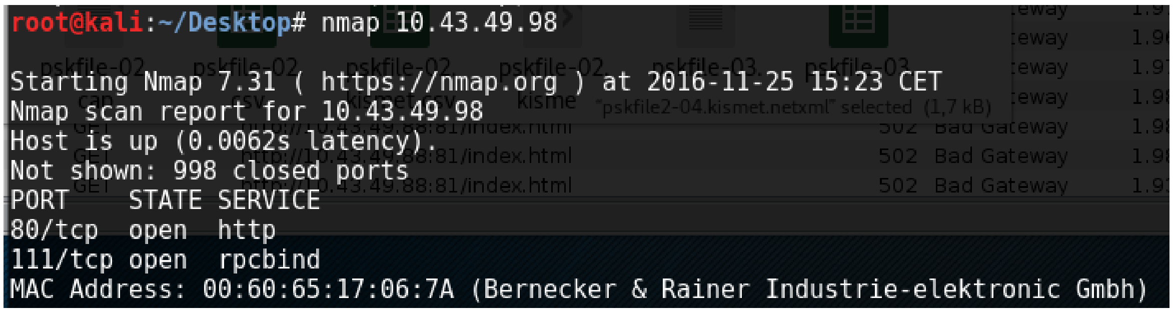


```
▷ Frame 61015: 145 bytes on wire (1160 bits), 145 bytes captured (1160 bits) on interface 0
▷ Ethernet II, Src: Vmware_16:a9:8d (00:0c:29:16:a9:8d), Dst: Bernecke_17:06:7a (00:60:65:17:06:7a)
▷ Simple Network Management Protocol

0000  00 60 65 17 06 7a 00 0c  29 16 a9 8d 81 4c 30 81   .`e..z.. )....L0.
0010  80 02 01 03 30 0e 02 02  2f 22 02 02 05 dc 04 01   ....0... /"......
0020  04 02 01 03 04 24 30 22  04 0b 73 6e 6d 70 45 6e   .....$0" ..snmpEn
0030  67 69 6e 65 00 02 01 00  02 02 13 2f 04 08 62 75   gine.... .../..bu
0040  72 55 73 65 72 00 04 00  04 00 30 45 04 0b 73 6e   rUser... ..0E..sn
0050  6d 70 45 6e 67 69 6e 65  00 04 0b 62 75 72 43 6f   mpEngine ...burCo
0060  6e 74 65 78 74 00 a3 29  02 02 1e da 02 01 00 02   ntext..) ........
0070  01 00 30 1d 30 1b 06 0c  2b 06 01 04 01 81 e5 00   ..0.0... +.......
0080  01 05 04 00 04 0b 31 30  2e 34 33 2e 34 39 2e 39   ......10 .43.49.9
0090  38                                                 8
```

Figure 41 - Wireshark Capture of Crafted SNMP Packet Screenshot

After this, the PLC is available with the new IP address, all communication targeted at the old IP is terminated.



```
root@kali:~/Desktop# nmap 10.43.49.98

Starting Nmap 7.31 ( https://nmap.org ) at 2016-11-25 15:23 CET
Nmap scan report for 10.43.49.98
Host is up (0.0062s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
80/tcp   open  http
111/tcp  open  rpcbind
MAC Address: 00:60:65:17:06:7A (Bernecker & Rainer Industrie-elektronic Gmbh)
```

Figure 42 - NMAP Fingerprint of the PLC after the Crafted SNMP Packet Attack Screenshot

3.4.6.3. Recommendation

The simple recommendation is to use SNMPv3 with any sort of authentication as described in RFC 4789:
“Security Considerations

This module does not define any management objects. Instead, it defines an OBJECT-IDENTIFIER which may be used by other MIB modules to identify an SNMP transport mapping. Meaningful security considerations can only be written in the MIB modules that define management objects. The MIB module in this document has, therefore, no impact on the security of the Internet.

SNMPv1 and SNMPv2c messages are not considered secure. It is recommended that the implementors consider the use of SNMPv3 messages and the security features as provided by the SNMPv3 framework. Specifically, the use of the User-based Security Model STD 62, RFC
3414 [RFC3414] and the View-based Access Control Model STD 62, RFC 3415 [RFC3415] is recommended.

It is then a customer/user responsibility to ensure that the SNMP entity giving access to a MIB is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change) them.” [99]

This would rule out the possibility of malicious attackers using simple replay attacks (with small changes to the packet) to tamper with the configuration of the PLC.

As an additional possible security measure, the listener daemons of the SNMP server operating at the IP layer could be deactivated, as there was no sign of traffic on the IP layer. Deactivation of the SNMP daemons could deter attackers, if the network analysis is only done via sniffing on the IP layer and the port scan does not reveal open SNMP communication ports.

3.5. Conclusion

Through this exploratory security analysis, critical vulnerabilities were found and reported to B&R. The approach showed that even with a minimum of resources spent, security loopholes were found in a product that is widely used in the field.
It was shown that even attackers with simple means and no sophisticated toolset could use wrong input data to create an error state, which would result in a shutdown of the control system. According to ISA/IEC 62443 this would mean that the device would not be suitable as a Security Layer 2 device.

Immediate actions would suggest that some of the methodologies applied should be reinstated as a permanent testing solution and the testing scope should be widened to real-world applications as there will be additional activated modules to be tested. Testing methods like fuzzing should be introduced into the development process and the test vector deepened.
Other mitigations, like network segregation (for example with VLANs or VPNs) can be applied to reduce the attack vector for such devices. But actions like these should be seen as a complementary measure and not as a permanent workaround.

The more challenging and sustainable action should be to enforce a more secure software development process so that security issues are mitigated earlier in the development lifecycle.

4. Lessons Learned & Future Work

4.1. Topics Covered

This thesis has shown known procedures and practices to ensure information security improvements for industrial products. In the age of the industrial Internet of Things this topic trends to be of utmost importance and will be a key selling point for future products in this field.
The problem of improving the security of already introduced products and their software was also addressed. The necessity for applying the business case of risk management to focus on the important issues was cleared and a sample security analysis to find potential threats was provided.
Furthermore, the task of securing future products was approached by introducing a secure software development lifecycle and different measures in the different phases of the process. This is already a best-practice scenario in the IT world and is being used by big software development companies such as Microsoft or Google.

4.2. Further Research Possibilities

4.2.1. Process Optimization

The first part of the thesis focused on the organizational approach of how to introduce security measures into existing software development lifecycles. The presented topics showed an overview of measures in every phase of the development process. Of course, there can be additional measures introduced in a specific phase as this is also still subject to intensive research. Also since the shown measures themselves are still in development, newly found best practices and case studies should be observed in ongoing research fields.

4.2.2. Extension of the Security Analysis

In chapter 3 an explorative approach for the security analysis was used. This analysis could be extended in scale and depth. For example, the possibility to do fuzz tests was only done on the web server protocols. Fuzz testing could be applied to every analyzed protocol and also directly to every interface which allows user input.
The basic strategy of the analysis was to do a black box test. This could be expanded to grey box or even white box testing, (e.g. with reviewing the underlying source code base). A more theoretical research possibility would be to try formal verification of the used code in sub components where applicable.

Also the attack vector could be expanded. In this thesis, we only covered the possibility of an attack on the Ethernet layer. Other possibilities could be the usage of the fieldbus network to directly manipulate control or process data. This should be particularly interesting, as new types of fieldbus protocols will probably use IP-based Ethernet according to current research. New technologies like Ethernet Time-Sensitive Networking (TSN) are emerging in this field and will allow real-time communication within Ethernet networks.

In addition to logical attacks, physical and side-channel attacks on the PLC could be an interesting research topic.
This would be especially interesting, because many newly found side-channel attacks deliberately target encryption-based security features.

5. Bibliography

All hyperlinks last checked: 21.07.2017

[1] Williams, C.: Today the web was broken by countless hacked devices – your 60-second summary, The Register, 21.10.2016, available at http://www.theregister.co.uk/2016/10/21/dyn_dns_ddos_explained

[2] Schneier, B.: Security Risks of Embedded Systems, 09.01.2014, available at https://www.schneier.com/blog/archives/2014/01/security_risks_9.html

[3] Schmidt, J.: Das Internet (der Dinge) darf kein rechtsfreier Raum bleiben, available at https://www.heise.de/security/meldung/Kommentar-Das-Internet-der-Dinge-darf-kein-rechtsfreier-Raum-bleiben-3529743.html

[4] European Commission: Cybersecurity, available at https://ec.europa.eu/digital-single-market/en/cybersecurity

[5] Department of Homeland Security (DHS): Executive Order (EO) 13636 Improving Critical Infrastructure Cybersecurity and Presidential Policy Directive (PPD)-21 Critical Infrastructure Security and Resilience, 29.12.2016, available at: https://www.dhs.gov/sites/default/files/publications/EO-13636-PPD-21-Fact-Sheet-508.pdf

[6] U.S. Title 44 Code § 3542 - Definitions, available at https://www.law.cornell.edu/uscode/text/44/3542

[7] Shirey R – RFC4949: Internet Security Glossary, Version 2, available at https://tools.ietf.org/html/rfc4949

[8] Anderson R.: Security Engineering: A Guide to Building Dependable Distributed Systems, Second Edition, April 2008, Wiley Publishing Inc.

[9] Barr M.: Real men program in C, August 2009, available at http://www.embedded.com/electronics-blogs/barr-code/4027479/2/Real-men-program-in-C

[10] ARM Ltd.: Q2 2016 Roadshow Slides, 2016, available at: https://www.arm.com/-/media/arm-com/company/Investors/Quarterly%20Results%20-%20PDFs/ARM_2016_Q2_Roadshow_Slides_Final.pdf?la=en

[11] Ganssle J.: The shape of the MCU market, 08.03.2016, available at: http://www.embedded.com/electronics-blogs/break-points/4441588/The-shape-of-the-MCU-market

[12] Barr Group – Embedded Systems Glossary, available at https://barrgroup.com/Embedded-Systems/Glossary-E

[13] Ravi S., et al.: Security in Embedded Systems: Design Challenges, August 2004, available at http://users.ece.gatech.edu/~dblough/8823/embedded_security.pdf

[14] Andress J. and Winterfeld S.: Cyber Warfare: Techniques, Tactics and Tools for Security Practitioners, Second Edition, Elsevier Inc. - Syngress

[15] Pathan A.: Securing Cyber-Physical Systems, 06.10.2014, CRC Press

[16] Sergei P. Skorobogatov: Semi-invasive attacks - A new approach to hardware security analysis, April 2005, available at https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-630.pdf

[17] Tehranipoor M.: Physical Attacks and Tamper Resistance, 25.12.2012, available at: http://www.engr.uconn.edu/~tehrani/teaching/hst/11%20Physical%20Attacks%20and%20Tamper%20Resistance.pdf

[18] Zhou Y. and Feng D.: Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing, 2005, available at: http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-3/physec/papers/physecpaper19.pdf

[19] Ramsay C. and Lohuis J.: TEMPEST attacks against AES, Covertly stealing keys for €200, 2017, available at: https://www.fox-it.com/nl/wp-content/uploads/sites/12/Tempest_attacks_against_AES.pdf

[20] Shamir A et al: RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis, 18.12.2013, available at: https://www.cs.tau.ac.il/~tromer/papers/acoustic-20131218.pdf

[21] Hoglund G. and McGraw G.: Exploiting Software, How to Break Code, 17.02.2014, Addison-Wesley Professional

[22] Langweg H. and Snekkens E.: A Classification of Malicious Software Attacks, April 2004, available at: https://pdfs.semanticscholar.org/2f6b/0e1b2e6ee65d5cb896ae067a074d6f4a7744.pdf

[23] Stouffer K.: NIST SP800-82r2 Guide to Industrial Control System (ICS) Security, May 2015, available at http://dx.doi.org/10.6028/NIST.SP.800-82r2

[24] Electrical Technology – What is Distributed Controls System (DCS)?, August 2016, available at http://www.electricaltechnology.org/2016/08/distributed-control-system-dcs.html#comments

[25] Galloway B. and Hacke G.: Introduction to Industrial Control Networks, available at http://www.rfidblog.org.uk/Preprint-GallowayHancke-IndustrialControlSurvey.pdf

[26] Papp D. et al.: Embedded Systems Security: Threats, Vulnerabilities, and Attack Taxonomy, 2015, available at http://www.cse.psu.edu/~pdm12/cse597g-f15/readings/cse597g-embedded_systems.pdf

[27] ISA: The 62443 series of standards: Industrial Automation and Control System Security, 2016, available at http://isa99.isa.org/Public/Information/The-62443-Series-Overview.pdf

[28] Boehm B.: Case study: Finding defects earlier yields enormous savings, Software Engineering Economics. Prentice Hall, Englewood Cliffs, NJ, 1981

[29] Boehm B. and Papaccio P.: Understanding and Controlling Software Costs, v.14, October 1988, IEEE Transactions on Software Engineering

[30] Stecklein J. et al: Error cost escalation through the project life cycle, June 2004, available at: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100036670.pdf

[31] Menzies T. et al: Are Delayed Issues Harder to Resolve? Revisiting Cost-to-Fix, September 2016, available at: https://arxiv.org/pdf/1609.04886.pdf

[32] Howard M. and Lipner S.: The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software, 2006, Microsoft Press

[33] McGraw G. et al: BSIMM7 Framework, 2017, available at: https://www.bsimm.com/framework/

[34] OWASP Foundation: Open Software Assurance Maturity Model (OpenSAMM) 1.5, April 2017, available at: https://www.owasp.org/images/8/8d/OWASP_SAMM_v1.5.zip

[35] Cisco Systems: Building Trustworthy Systems with Cisco Secure Development Lifecycle, January 2016, available at: https://www.cisco.com/c/dam/en_us/about/doing_business/trust-center/docs/building-trustworthy-systems-with-CSDL.pdf

[36] Microsoft: Microsoft Security Development Lifecycle (SDL) Process Guidance - Version 5.2, 2016, available at: https://www.microsoft.com/en-us/download/details.aspx?id=29884

[37] Wilson M and Hash J.: NIST SP800-50 Building an Information Technology Security Awareness and Training Program, 10.2003, available at http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-50.pdf

[38] Howard M and LeBlanc D: Writing Secure Code: Practical Strategies and Proven Techniques for Building Secure Applications in a Networked World (Developer Best Practices), Second Edition, 2009, Microsoft Press

[39] Seacord R.: Secure Coding in C and C++, Second Edition, 2013, Addison-Wesley Professional

[40] Firesmith D. - Engineering Safety- and Security-Related Requirements for Software-Intensive Systems, May 2010, available at https://resources.sei.cmu.edu/asset_files/Presentation/2010_017_001_23269.pdf

[41] Schneier B.: Attack Trees, December 1999, available at https://www.schneier.com/academic/archives/1999/12/attack_trees.html

[42] Vesely W. E. et al – Fault Tree Handbook, 1981, available at https://www.nrc.gov/docs/ML1007/ML100780465.pdf

[43] Braber F. eta al - Model-based security analysis in seven steps — a guided tour to the CORAS method, January 2007, available at https://heim.ifi.uio.no/massl/publications/BTTJ.pdf

[44] Mouratidis H. and Giorgini P. – Secure Tropos: A security-oriented extension of the tropos methodology, 2007, available at http://disi.unitn.it/~pgiorgio/papers/IJSEKE06-1.pdf

[45] Firesmith D. – Engineering Security Requirements, 2003, available at http://www.jot.fm/issues/issue_2003_01/column6/

[46] Carnegie Mellon University: SQUARE, 2010, available at https://www.cert.org/cybersecurity-engineering/products-services/square.cfm

[47] Shostack A. – Threat Modelling: Designing for Security, 2014, John Wiley & Sons.

[48] OWASP Foundation: Top 10 Web Vulnerabilities, 2013, available at https://www.owasp.org/index.php/Top_10_2013-Top_10

[49] Elahi G.: Security Requirements Engineering: State of the Art and Practice and Challenges, available at http://www.cs.toronto.edu/~gelahi/DepthPaper.pdf

[50] Hadavi M. et al.: Security Requirements Engineering; State of the Art and Research Challenges, 2008, available at http://www.iaeng.org/publication/IMECS2008/IMECS2008_pp985-990.pdf

[51] Saltzer J. and Schroeder M.: The Protection of Information in Computer Systems, September 1975, Proc IEEE 63(9)

[52] Huang Y.: Security Principles, 2015, available at: http://homes.soic.indiana.edu/yh33/Teaching/I433-2016/lec2-principles.pdf

[53] OWASP Foundation: Security by Design Principles, August 2016, available at: https://www.owasp.org/index.php/Security_by_Design_Principles

[54] Smith R.: Security Design Principles, October 2013, available at: https://cryptosmith.com/2013/10/19/security-design-principles/

[55] Kahn, D.: The Codebreakers: the story of secret writing, 1996, Scribners Publishing

[56] Ross R. et al: NIST SP800-160: Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems, November 2016, available at: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160.pdf

[57] OWASP Foundation: Attack Surface Analysis Cheat Sheet, July 2015, available at: https://www.owasp.org/index.php/Attack_Surface_Analysis_Cheat_Sheet

[58] Majeed M. and Quadri S.: Analysis and Evaluation of “Reducing the Attack Surfaces” to improve the security of the software at Design Level, 2016, International Journal of Computer Science and Information Technologies, available at: http://ijcsit.com/docs/Volume%207/vol7issue3/ijcsit2016070392.pdf

[59] Howard M. et al: Measuring Relative Attack Surfaces, 2003, available at: https://www.cs.cmu.edu/afs/cs/project/svc/projects/security/wadis1.pdf

[60] Schneier B.: Applied Cryptography: Protocols, Algorithms and Source Code in C, 20th anniversary edition, May 2015, John Wiley & Sons

[61] Kessler G.: An Overview of Cryptography, April 2017, available at: http://www.garykessler.net/library/crypto.html

[62] Graff M. and Wyk K.: Secure Coding: Principles and Practices, 2003, O’Reilly Media

[63] Bolton D.: Why Managed Code Is Safer, January 2014, available at: http://insights.dice.com/2014/01/29/managed-vs-unmanaged-code/

[64] Wichers D.: OWASP: Input Validation Cheat Sheet, June 2016, available at: https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet

[65] OWASP Foundation: XML Security Cheat Sheet, Mai 2017, available at: https://www.owasp.org/index.php/XML_Security_Cheat_Sheet

[66] Microsoft: Dynamic-Link Library Security, available at: https://msdn.microsoft.com/en-us/library/windows/desktop/ff919712(v=vs.85).aspx

[67] OWASP Foundation: Secure Coding Practices Quick Reference Guide, November 2010, available at: https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf

[68] J.B. Almeida et al., Rigorous Software Development, 2011, Springer-Verlag, available at: https://www.springer.com/cda/content/document/cda_downloaddocument/9780857290175c2.pdf?SGWID=0-0-45-1053837-p174029011

[69] McConnell S.: Code Complete: A Practical Handbook of Software Construction, Second Edition, 2004, Microsoft Press

[70] OWASP Foundation: OWASP Code Review Project, June 2017, available at: https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project

[71] Vranken G.: The OpenVPN post-audit bug bonanza, 21.06.2017, available at: https://guidovranken.wordpress.com/2017/06/21/the-openvpn-post-audit-bug-bonanza/

[72] FIRST.Org, Inc.: About CVSS, June 2015, available at https://www.first.org/cvss

[73] The Mitre Corporation: Common Vulnerabilities and Exposures (CVE) Numbering Authority (CNA) Rules, September 2016, available at https://cve.mitre.org/cve/cna/CNA_Rules_v1.1.pdf

[74] Stoneburner, G. et al: NIST SP800-30r1 Guide for Conducting Risk Assessments, September 2012, available at http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf

[75] OWASP Foundation: Threat Risk Modelling, 2016, available at https://www.owasp.org/index.php/Threat_Risk_Modeling#DREAD

[76] ISO/IEC 27005:2011: Information technology - Information security risk management, Second edition, June 2011

[77] Ross R. et al: NIST SP800-39 Managing Information Security Risk, March 2011, available at: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-39.pdf

[78] Kissel R. et al: NIST SP800-64r2 Security Considerations in the System Development Life Cycle, October 2008, available at: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-64r2.pdf

[79] Andress J. and Linn R.: Coding for Penetration Testers: Building Better Tools, Second Edition, September 2016, Elsevier Inc. – Syngress

[80] Cardwell K.: Building Virtual Pentesting Labs for Advanced Penetration Testing, June 2014, Packt Publishing

[81] OWASP Foundation: ZED Attack Proxy Project, 2016, available at https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

[82] Law, E.: Combating ClickJacking With X-Frame-Options, 30.03.2010, available at https://blogs.msdn.microsoft.com/ieinternals/2010/03/30/combating-clickjacking-with-x-frame-options/

[83] Dawson, I.: Guidelines for Setting Security Headers, 12.03.2014, available at https://blog.veracode.com/2014/03/guidelines-for-setting-security-headers/

[84] Microsoft Corp.: Reducing MIME type security risks, 2014, available at https://msdn.microsoft.com/en-us/library/gg622941(v=vs.85).aspx

[85] Zalewski M.: The Tangled Web: A Guide to Securing Modern Web Applications, 2012, No Starch Press, Inc.

[86] ZAP Extension project, 2016, available at https://github.com/zaproxy/zap-extensions/tree/master/src/org/zaproxy/zap/extension/fuzzdb/files/fuzzers/fuzzdb/attack/json

[87] The hashcat project forum, 03.09.2016, available at https://hashcat.net/forum/thread-5832.html

[88] Vanhoef M. and Piessens F.: All Your Biases Belong To Us: Breaking RC4 in WPA-TKIP and TLS, 2015, available at https://www.rc4nomore.com/vanhoef-usenix2015.pdf

[89] Bhargavan K. and Leurent G.: On the Practical (In-)Security of 64-bit Block Ciphers, 2016, available at https://sweet32.info/SWEET32_CCS16.pdf

[90] Pellegrini A. et al.: Fault-Based Attack of RSA Authentication, 2010, available at http://www.eecs.umich.edu/~valeria/research/publications/DATE10RSA.pdf

[91] Schneier B.: When Will We See Collisions for SHA-1?, October 2012, available at https://www.schneier.com/blog/archives/2012/10/when_will_we_se.html

[92] Stevens M. et al.: The first collision for full SHA-1, 2017, available at https://shattered.it/static/shattered.pdf

[93] Moeller B. and Langley A.: IETF RFC7507: TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks, April 2015, available at https://tools.ietf.org/html/rfc7507#section-3

[94] The RFB Protocol, 2016, available at https://github.com/rfbproto/rfbproto/blob/master/rfbproto.rst

[95] Tipton F. and Krause M.: Information Security Management Handbook, Fifth Edition, 2004, CRC Press

[96] The OPC Foundation: Unified Architecture, 2016, available at https://opcfoundation.org/about/opc-technologies-/opc-ua/

[97] Tao B. and Wu H.: Improving the Biclique Cryptanalysis of AES, 2015, available at https://link.springer.com/chapter/10.1007/978-3-319-19962-7_3

[98] Leskiw A.: SNMP Basics: What is SNMP & How do I use it?, available at http://www.networkmanagementsoftware.com/snmp-tutorial/

[99] Schoenwaelder J. et al.: RFC4789: Simple Network Management Protocol (SNMP) over IEEE 802 Network, November 2016, available at https://tools.ietf.org/html/rfc4789

JOHANNES KEPLER
UNIVERSITY LINZ

6. Table of Figures

All hyperlinks last checked: 21.07.2017

7. Curriculum Vitae

Personal Data

Name: Fabian Fisecker
Address: Rödt 5, 4922 Geiersberg
Date of Birth: 09.05.1988

Education

2015-present **Master of Science in Computer Science**
Johannes Kepler University, Linz, Austria

2008-2010 **Bachelor of Science in Software Engineering**
University of Applied Science, Hagenberg, Austria

2002-2007 **Matura, HTL Grieskirchen in Information Technology and Organization**
Secondary technical school, Grieskirchen, Austria

Work Experience

2008-present **Bernecker + Rainer Industrie Elektronik Ges.m.b.H.**
Software & Security Engineer

2007-2008 **Civilian Service, Red Cross Austria**
Paramedic

2006-2007 **Doma Elektroengineering GmbH**
System Administrator

JOHANNES KEPLER
UNIVERSITY LINZ

8. Sworn Declaration

I hereby declare that the thesis submitted is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references.
This printed thesis is identical with the electronic version submitted.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.
Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.