D1.1

Security requirements for connected medium security-critical applications

Project number:	731453
Project acronym:	VESSEDIA
Project title:	Verification engineering of safety and security critical dynamic industrial applications
Start date of the project:	1st January, 2017
Duration:	36 months
Programme:	H2020-DS-2016-2017

Deliverable type:	Report
Deliverable reference number:	DS-01-731453 / D1.1/ V1.0
Work package contributing to the deliverable:	WP1
Due date:	September 2017 – M09
Actual submission date:	15th October, 2018

Responsible organisation:	SLAB
Editor:	Vendel Laszlo
Dissemination level:	PU
Revision:	2.0

Abstract:	In this study, we provide a set of minimum requirements for interconnected products, specifically focusing on the IoT devices. To aid these requirements, we also collected the most common assets and threats related to the field of IoT. To facilitate the future work in the VESSEDIA project, we also present some risk assessment techniques and a study on, how ASCL can be used to verify security properties
Keywords:	Internet of Things, IoT security requirements, risk assessment, threat modelling, minimal contracts

The project VESSEDIA has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731453.

Editor

Vendel László (SLAB)

Contributors

Gergely Eberhardt (SLAB)

Jochen Burghardt, Jens Gerlach (FOKUS)

Felix Stornig (TEC)

Emmanuel Querrec (TUAS)

Igor Grueiro Santos (FD)

Allan Blanchard (INRIA)

Executive Summary

IoT (Internet of Things), which denote connected devices and services are on a rapid increase, and as they are gaining wider and wider adoption in the security critical fields, it becomes more urgent to ensure the security of these devices. The VESSEDIA project aims to enhance the security of IoT devices by improving already existing software analysis tools to help the manufacturers to develop more secure devices.

The goal of this document is to outline the most important security requirements of the IoT. Our goal was to determine these requirements at a higher level, so the final requirements can be applied to different IoT systems. To set these requirements, we divided the IoT into different layers, and we analysed these layers separately from the security perspective. Finally, we collected a set of requirements and recommendations, which are required for an IoT device to ensure secure functionality.

In requirements engineering, risk assessment is used to identify flaws in a given system, and associate risks according to the severity and potential exploitability of these flaws. To facilitate of this process, we outlined as well some common risk assessment technique, which can be applied in the field of IoT. Finally, we analysed, how ACSL, the specification language of Frama-C can be used to verify security related C libraries. This work will be utilized as a general estimation of the capabilities of formal verification techniques in security requirement verification, and our findings can be used as a direction of possible future development.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 VESSEDIA motivation and background

The VESSEDIA project aims to bring safety and security to the next generation of software applications and internet connected devices. In our rapidly changing world, the Internet has been the source of many benefits for individuals and companies alike, transforming entire industries. With this new technology, capable of connecting billions of devices and people together, new threats have also appeared – threats VESSEDIA will help software developers address in order to create connected applications that are safe and secure. VESSEDIA proposes to enhance and scale up modern software analysis tools, in particular the mostly open-source Frama-C analysis platform, to make them useful and accessible to a wider audience of developers working on connected applications. At the forefront of connected applications are the Internet of Things (or IoT for short), where we have seen explosive growth and where security risks have become all too real. VESSEDIA will focus on this domain to demonstrate the benefits our tools bring to the table when developing connected applications. VESSEDIA will tackle this challenge by 1) developing a methodology that makes it possible to adopt and use source code analysis tools as efficiently and with similar benefits as it is already possible in the case of highly-critical applications, 2) enhancing the Frama-C toolbox to enable efficient and fast implementation, 3) demonstrating the capabilities of the new toolbox on typical IoT applications, including an IoT Operating System (Contiki), 4) developing a standardisation plan for generalising the use of the toolbox, 5) contributing to the Common Criteria certification process, and 6) defining a “Verified in Europe” label for validating software products with European technologies such as Frama-C.

1.2 Structure of the document

After these introductory sections, 0 defines the typical security objectives for IoT, starting with a general overview of the IoT architecture. It then characterizes the possible attacker actors with their strengths and motivations. After a quick overview of the security properties, it collects the general assets along with the corresponding objectives. Chapter 3 gives an introduction to the threat modelling methodologies such as attack trees, misuse cases and SDL (Security Development Lifecycle) threat modelling. After an overview of the IoT attack surface in Chapter 4, the typical threats and general IoT security requirements are detailed in the layered approach described in 2.1. Security requirements may not applicable or worthwhile in every case. Chapter 5 describes some risk analysis techniques that can help to find out the most serious threats and the corresponding requirements. In Chapter 6 formal specifications of simple security requirements are presented based on the minimal contract concept. Finally, Chapter 7 concludes the document and paves the way for the subsequent work in the project.

1.3 Related deliverables

As it was mentioned previously this document serves as a baseline for the subsequent work by listing the most common, high level security requirements for the IoT. More detailed security requirements will be outlined in the D1.2, where the use cases from WP5 will have a more detailed analysis. These requirements will be used in WP4, when we will carry out the security evaluation for the use cases in D4.6.

Chapter 2 Security objectives for IoT

As it is predicted by IHS technology[1], the IoT market will grow from an installed base of 15.4 billion devices in 2015 to 30.7 billion devices by 2020 and 75.4 billion by 2025. This many devices connected to the internet raise many security risks, which have to be addressed in the future. Securing data on the IoT ecosystem raises a lot of challenges, like accessibility of remote devices, lack of processing power to use traditional security mechanisms, and the continuously growing attack surface, as the number of devices and systems increase. Currently (and in the future), there is no silver bullet for IoT to effectively mitigate these security issues[2].

Determining security requirements for IoT can be approached similarly to any other target. In information systems, providing security covers several aspects, such as defending information from unauthorized access, usage, disclosure, disruption, modification, perusal, inspection, recording or destruction. Security objectives are the high-level statement of intent and goals that are most important to the stakeholders and to fulfill the requirements that must be met to comply with relevant legislation, policies and standards.

The proposed methodology for determining the security requirements is the following:

1. Identify functional requirements and business goals: it is hard to determine the general functional requirements for IoT. In the definition, IoT is an inter-networking device, which is capable to collect and exchange data. In this study, we can separate the IoT architecture into 3 different layers, which can be analysed separately.
2. Collect assets, which should be protected in the system (e.g. sensitive user data). These assets can vary from device to device, but we define some general ones.
3. Identify security objectives of each asset (e.g. confidentiality of sensitive user data should be protected)
4. Perform threat modelling and identify threats to the security objectives of the assets.
5. Identify security requirements to cover, mitigate or reduce the risk associated with the identified threats.
6. Evaluate/test/verify that the system fulfils the security requirements.

[1] IHS TECHNOLOGY: IoT platforms: enabling the Internet of Things https://cdn.ihs.com/www/pdf/enabling-IOT.pdf March 2015

[2] Wind River Systems SECURITY IN THE INTERNET OF THINGS https://www.windriver.com/whitepapers/security-in-the-internet-of-things/wr_security-in-the-internet-of-things.pdf 2015

Figure 1 illustrates the terms used in this document and their relationships between each other:

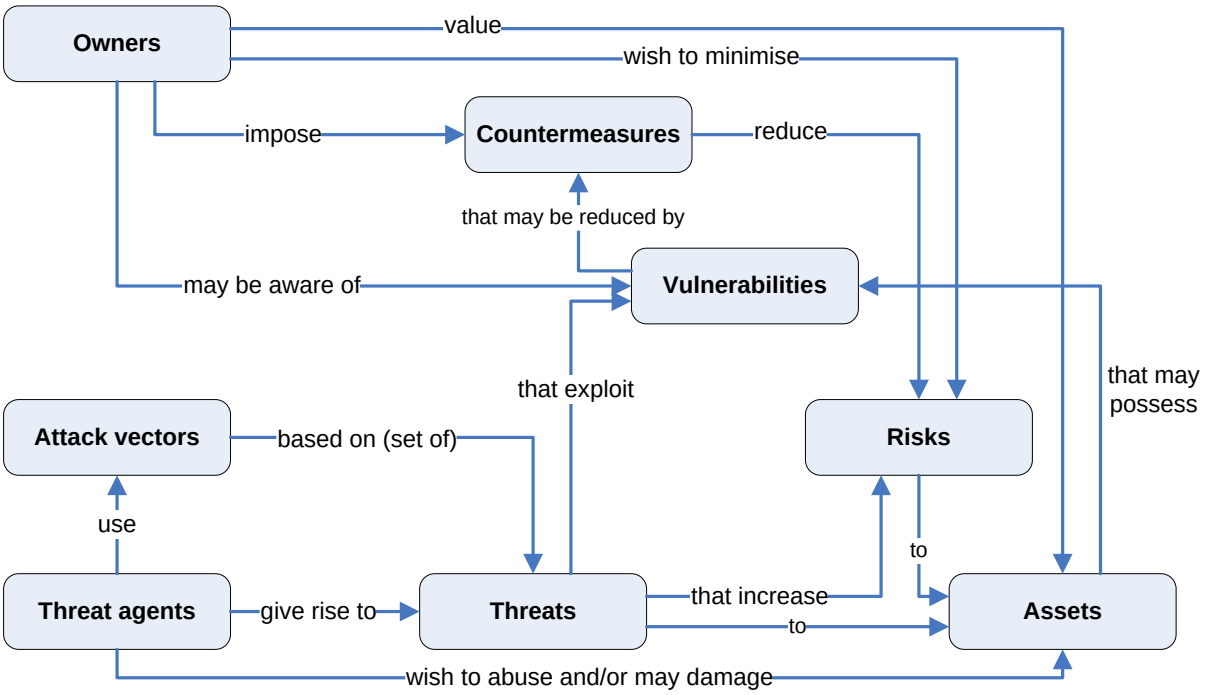

Figure 1: The elements of risk and their relationships according to ISO 15408:2005

2.1 IoT architecture

Identifying security requirements for an ecosystem as complex as the Internet of Things is non-trivial task. One way to simplify the problem is to break it down into smaller ones, by partitioning the whole into smaller parts. In case of IoT devices and systems, a multitude of approaches exist in scientific literature to identify and separate the different layers, thus helping us conduct security analysis on each one separately, focusing on problems related to a smaller group of components.

One of the simpler models for identifying these layers is the three layer IoT architecture, as seen in Figure 2. Those familiar with network stacks might recognize its structure being similar to the OSI model. An advantage of this model for the purpose of security analysis is that the layers each have very different types of devices and services, thus the number of overlapping requirements can be minimized.

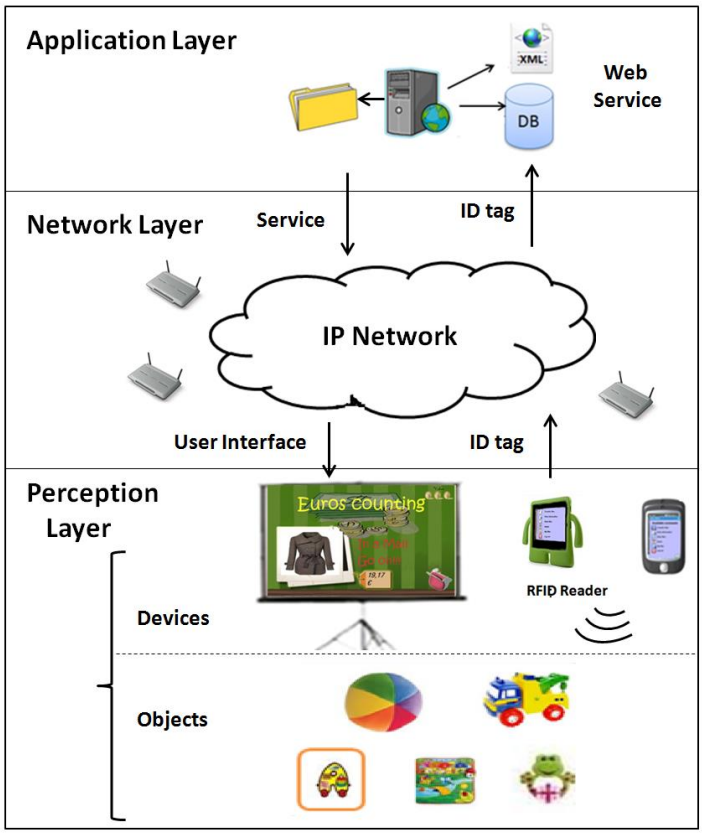

Figure 2 Three layer IoT architecture
Source: Radio Frequency Identification from System to Applications [3]

- **Perception layer**

  The perception layer percieves the physical reality around us – a fact somewhat foreshadowed by its aptly chosen name. Devices in this layer are tasked with being a bridge between the analog and the digital world. Both data acquisition – using conventional sensors or more complicated methods – and preliminary data processing are happening here, as the data is being prepared for transmission to the layers above. The perception layer also includes devices which are focused on making objects percievable – like RFID tags or even barcodes.

- **Network layer**

  The network layer includes both the components responsible for the transmission of the data between the layer above and the layer below, and the storage of such data for later retreival. A wide variety of technologies are in use today to facilitate the transmission of data gathered by the perception layer: cellular networks, traditional wired networks, meshed wireless networks and satellite communications can all serve as infrastructure for the transmission of said data. A significant portion of the data gathered ends up being transmitted to and stored in cloud computing systems. Many cloud service providers today have offerings targeting IoT applications, offering cost effective transmission, storage and processing for busineses all over the globe.

[3] Elena de la Guía, María D. Lozano and Víctor M.R. Penichet (2013). Interacting with Objects in Games Through RFID Technology, Radio Frequency Identification from System to Applications, Dr. M. I. B. Reaz (Ed.), InTech, DOI: 10.5772/53448. Available from: https://www.intechopen.com/books/radio-frequency-identification-from-system-to-applications/interacting-with-objects-in-games-through-rfid-technology

- **Application layer**
  The application layer analyzes the data coming from and stored in the network layer, and presents the data in a way that is convenient and exploitable by end users or other services building on the IoT architecture – and thus serves as the front end for the whole system. The application layer also includes the devices that are capable of acting on the information acquired – using such information to make decisions and control processes with or without human interaction.

Another version of this three layer architecture can be seen in Figure 3. While the nomenclature is different, the layers themselves can be mapped to the three layer architecture described previously.

- Communications Network → Network Layer
- Endpoint Ecosystem → Perception Layer

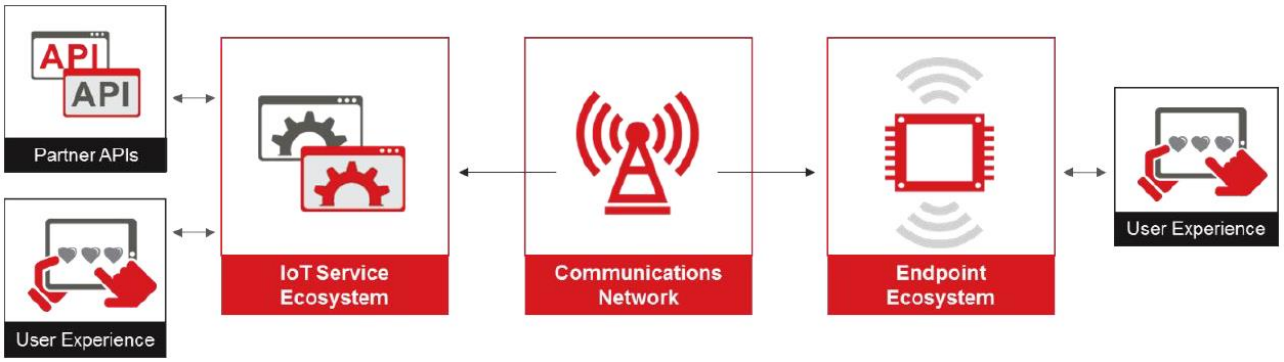

Figure 3: A different version of the three layer IoT architecture

Source: *IoT Security Guidelines Overview Document*[4]

While this three-layered architecture will serve us well for the purposes of establishing security requirements, it is important to mention that several other models exist for describing the architecture of IoT systems, as seen of Figure 4.

[4]https://www.gsma.com/iot/wp-content/uploads/2016/11/CLP.11-v1.1.pdf

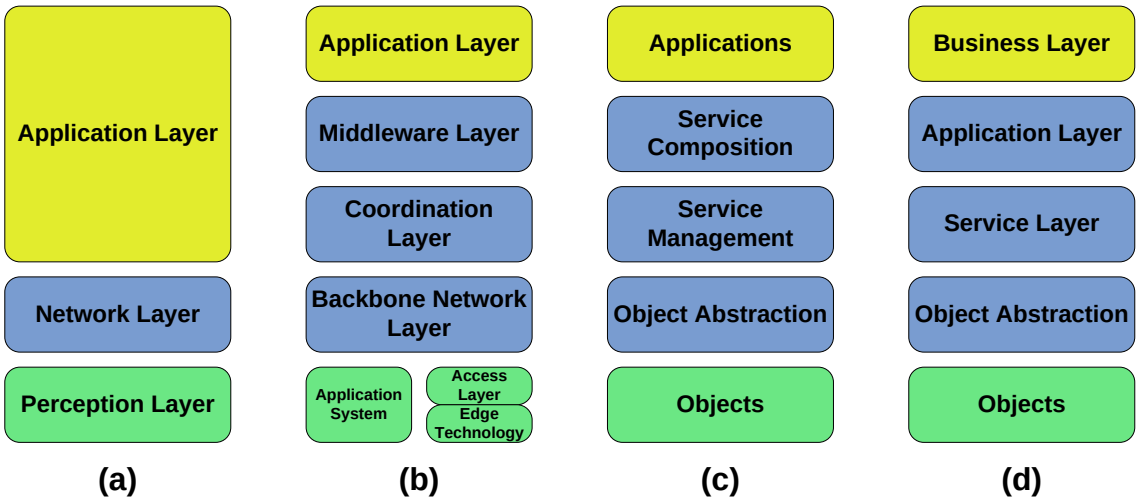

Figure 4: Comparison of proposed IoT architectures (a) Three-layer, (b) Middleware based, (c) SOA based, (d) Five-layer
Source: *Al-Fuqaha et al. 2015, p.2349*[5]

These five or even six layered architectures try to address different methodologies for designing IoT systems – by placing service composition or management into different layers, or by partitioning the application layer even further. Nonetheless, we will be using the three-layered approach presented in 0 to examine the security issues related to the different layers, analyzing them one by one to identify threats and propose ways to neutralize them.

2.2 Attacker model for IoT

An important part of the threat modelling process is the development of an attacker profile. Such a profile describes possible internal and external agents that might want to realize threats.

We have identified several different attacker profiles in a typical IoT ecosystem:

- The **Rogue Employee** works as tech support or customer service agent for a Manufacturer or a Service Provider. In order to carry out his tasks, he has limited access to the backend servers, including any functionality that allows him to access individual end-user devices remotely. He may not necessarily be capable of creating exploits, but he has full knowledge of the actual IoT platform, including possible access to its source code. He can abusethis position mainly to *steal user data for profit*; with the objectiveto remove any traces of his activities.
- The **Hacker** has access to debug and development tools, and has the resources necessary to create and deploy exploits targeting all components of the IoT platform as well as individual end-user devices. He also has reverse-engineering skills, and is assumed to have insider-level knowledge of the inner workings of the IoT platform. His eventual goal is *gaining money by stealing valuable user data*, *taking over end-user devices* (gaining access to the user's other networked devices and their data through the local network if possible) *and the IoT platform servers*, or *building a botnet*. In addition, a Hacker may be motivated to *spy on a particular high-value target* by taking over their devices and stealing their user data.
- The **Vandal** has similar resources and expertise as the Hacker, but his motivation is fundamentally different: instead of making money, he is *interested in fame and prestige*, possibly due to hacktivist reasons. Thus, he is interested in disrupting the operation of the

[5]https://www.researchgate.net/publication/305222860_Impact_Analysis_of_the_Internet_of_Things_on_the_Value_Chain_in_Manufacturing_Industries

IoT platform as well as that of individual devices in visible ways and leaving his mark through defacement of publically-accessible IoT platform web components. In addition, in case he gains access to the user's local network through a compromised device, he can abuse the user in various disruptive ways, e.g. yelling through a baby monitor's microphone.

- The **Burglar** is a professional criminal with access to high-tech equipment, and in-depth knowledge about the workings of end-user surveillance devices connected to the IoT platform. He has limited physical access to the building exterior – including the capability of getting within Wi-Fi or Bluetooth transmission range of all connected surveillance devices without being detected. His main goals are to *subvert the operation of surveillance equipment* – or manipulate stored footage – preferably remotely in order to gain entrance to the building without leaving evidence.
- **Malware** specializes in making malicious extensions and plug-ins for the IoT devices, and tricking users into installing them. Once installed on a device, such a malicious extension may have the ability to *alter the basic operation of the device*, *spy on the device's user*, and compromise other devices as well (e.g. by giving the Malware owner access to the user's account). In addition, the deployed malware can encrypt any data – such as video footage or personal media data – stored on the device, and then *extort the device's owner* for the encryption key.
- The **Advanced Persistent Threat (APT)** can be an individual or a group that specialises in operative procedures involving physical presence, *aiming at high value targets*, and specifically focusing on the IoT platform. Covert operations carried out by an APT can include dumpster diving, phone theft, wiretapping, social engineering, and non-invasive scans of biometric data.

The highest level of technical skill is possessed by the Hacker, Vandal and the Malware – and potentially the APT – followed by the Burglar who may also need to resort to hacking methods in order to gain unauthorized access to user's accounts as well as the Rogue Employee who may be able to install backdoors into the IoT service itself.

2.3 Security properties

In this section we review some of the most commonly used security properties to define the security objectives of each asset.

2.3.1 The CIA triad

Generally speaking, the Confidentiality, Integrity and Availability triad is used to express security objectives of an asset.

- **Confidentiality**: This property ensures that the information remains secret and will be disclosed only to authorized entities.
- **Integrity**: Preserving the integrity of a system means that the information it holds remains intact and complete. To preserve the integrity of information, modifications and data manipulation by unauthorized parties have to be prevented.
- **Availability**: The availability of an information or an information system means that it is accessible and available when it is needed. High availability systems aim to prevent service disruptions. Ensuring availability involves preventing denial-of-service attacks.

2.3.2 Parkerian Hexad

The Parkerian Hexad extends the **Confidentiality**, **Integrity** and **Availability** triad with three more security properties, which are extensions to the original CIA properties:

- **Possession** or **Control**: Possession means the ownership or control of the protected information. In contrary to the confidentiality, possession does not require knowledge of the information itself, just the ability to control it.
- **Authenticity**: This security property ensures that the information is original in the sense that it was not tampered or altered. It extends the integrity property by providing evidence of the origin or authorship of the information.
- **Utility**: Extends the avaliability property by describing the usefullness of it. For example encrypted data without the correct key is not useful to anybody.

2.3.3 Complimentary attributes to define objectives

This list of main objectives can be extended[6] to include a number of other security objectives, which refine the CIA triad and the Parkerian Hexad.

Objective	Description
access control	Restricting access to resources to privileged entities.
adoption of content and intend	A means to bind information to an entity, such as digital signatures.
anonymity	Concealing the identity of an entity involved in a process.
authorization	In our context it typically means the process of specifying access rights to certain resources.
certification	Endorsement of information by a trusted entity.
entity authentication or identification	Corroboration of the identity of an entity (e.g., a person, a computer terminal, a credit card, etc.). In our context it usually means the process of verifying the identity of the user or a device in order to determine its accuracy and trustworthiness.
message authentication	Corroborating the source of information; also known as data origin authentication.
non-repudiation	Preventing the denial of previous commitments or actions.
ownership	A means to provide an entity with the legal right to use or transfer a resource to others.
receipt confirmation	Acknowledgement that information has been received.
revocation	Retraction of certification or authorization.
service confirmation	Acknowledgement that services have been provided.
validation	Ensuring that data is safe prior to use.
witnessing	Verifying the creation or existence of information by third party.

Table 1: List of security objectives

The selection of security objectives strongly depends on the assumptions about the attacker and on the general scenario and policies that the software is exposed to. The exact security objectives relevant for VESSEDIA use-cases will be discussed in D1.2.

The table hereafter illustrates how security objectives can be related to threats in the STRIDE model.

Threats (STRIDE)	CIA	Parkerian	Complementary
Spoofing identity of user	Integrity	Authenticity	Authentication
Tampering with data	Integrity	Integrity	Integrity
Repudiation of the action	Integrity	Integrity	Non-repudiation

[6] A. Menezes, P. van Oorschot, and S. Vanstone.Handbook of Applied Cryptography, CRC Press, 1996.

Threats (STRIDE)	CIA	Parkerian	Complementary
Disclosure of information	Confidentiality	Confidentiality	Confidentiality
Denial of service	Availability	Availability	Availability
Elevation of privilege for user	Integrity	Integrity	Authentication and Authorization

Table 2: Connection between the threats in STRIDE model and security objectives

2.4 General assets of IoT

Data assets

- **Personally identifiable information[7] (PII)**: Some IoT devices may store sensitive or confidential user data, such as private photos, contacts, video recordings, log files, sensor data and so on. The users expect that their files are not available to and cannot be tampered by third parties, and that they would be available on demand.
- **Credentials**: IoT devices or the IoT ecosystem may require user credentials for device administration or cloud service access. After providing the correct username and password, all features and services of the device become available for administration. Thus, user credentials are critical assets which are expected to remain secret.
- **Device settings**: Device configuration settings are mandatory parts of operation. Configuration settings might include additional credentials as well and further access control related rules. In case of a security breach, reconfigured devices might serve as a hop for attackers to mount additional attacks.
- **Application data**: Applications running on IoT devices may use and store application related data. These data pieces may not contain any sensitive information, but can be used to understand application logic or to modify the state of the application in a malicious way.
- **Device identification data**: Device identification may rely on unique data stored in the device, such as MAC address, serial number or other unique device identification number.
- **Sensor data**: Correct sensor data is essential for the proper operation of the IoT device.

Software assets

- **Firmware**: The software running in an IoT device generaly consist of a bootloader, kernal and application software. Depending on the actual architecture the bootloader may contain multiple stages and in case of real time operating systems (RTOS) the application part is compiled along with the core OS.
- **Server software**: The server software provides backend services for the IoT devices and may provide remote access to the devices via web or mobile user interfaces.
- **Mobile application**: Other than web based administrator interfaces, it is trending to access the IoT device via a custom mobile application, which makes it the part of the IoT ecosystem. Typically mobile applications access the IoT device via the IoT backend server or via local access, such as using local network, bluetooth or other local wireless connections.

Hardware assets

[7] https://en.wikipedia.org/wiki/Personally_identifiable_information

- **JTAG key**: The JTAG port is the standard debugging interface of embedded devices. In case a production device provides debugging support, the JTAG interface has to be protected with a JTAG key, which is fused to the hardware.
- **Fused secrets**: IoT devices typically contain a system on a chip (SoC), which integrates all components of an electronic system. A modern SoC integrates secure processor and a secure storage also. The secure storage is a one time programmable memory, which is accessible only by the secure processor and only the secure processor can perform operations with the secrets. In most cases the fused secrets form the root of trust and made the chain of trust possible.

Cryptographic assets

- **Private keys**: SSL/TLS secured connections require public-key cryptography to securely establish a shared session key between the two parties. Obtaining the private key corresponding to the public party advertised in one of the server component's x.509 certificate would allow an attacker to impersonate the server. In an IoT environment typically the backend service, the web user interface in the cloud and the web administration interface in the device supports SSL/TLS connections, which requrire the storeage of private keys. With the private key obtained, a malicious third party could initiate man-in-the-middle attacks and intercept communication between the parties.
- **Certificates**: Signature verification (e.g. firmware update) and the establishment of an SSL/TLS connection require trusted certificates to make sure that the data sent by the other party was not changed.

Using the CIA triad, the following security objectives can be defined for the collected assets	**Confidentiality**	**Integrity**	**Availability**
Personally identifiable information	X		
Credentials	X	X	X
Device settings	X	X	X
Application data	X	X	X
Device identification data	X	X	X
Sensor data	X	X	
Firmware		X	X
Server software		X	X
Mobile application		X	X
JTAG key	X		
Fused secrets	X		
Private keys	X	X	
Certificates		X	
Encryption keys	X		

Table 3: Security objectives for common IoT assets

2.5 Regulations

Security requirements should also be derived from regulations set by national authorities. These regulations concern Digital Service Providers (DSP), Operators of Essential Services (OES), Governing agencies, and all organizations which offer goods and services or monitoring of individuals. There is no single international framework for cybersecurity law, but some multi-lateral

efforts have taken place. Two main regulations can be highlighted, one is the GDPR[8] regulation in the EU, which takes over the 95/46/EC directive, and a newly proposed legislation from the US[9] "Internet of Things Cybersecurity Improvement Act 2017".

2.5.1 General Data Protection Regulation

The GDPR aims to bring a single standard for data protection among all member states in the EU[10]. Although it was not originally intended for IoT, many of its application affect this domain. To complement the GDPR, a new proposal of the e-privacy directive (also known as the "cookie law") has been made in 2017 January, which replaces the current directive with a new legislation. The new legislation standardises privacy rules between EU members, protects the people from unsolicited electronic communication (aka. spam), and it sets a high level of privacy rules for electronic communication.[11]

The GDPR applies to any company that controls or processes personal data of Europeans through the offering of goods and services, even if the company itself has no physical presence in Europe. In case of a security breach, the company would be required to pay fines of up to 4% of its annual global revenue or €20 million for violations (whichever is greater).

The GDPR is strengthening the privacy rights of individuals, whose personal data is being processed, including through[12]

- the need for the individual's clear consent to the processing of personal data;
- the right to access by the subject to his or her personal data;
- the right to rectification, to erasure and 'to be forgotten';
- the right to object, including to the use of personal data for the purposes of 'profiling';
- the right to correct the data if it is out of date, incomplete, or incorrect;
- the right to be notified within 72 hours upon the company realizing a data breach that compromised personal data;
- and the right to data portability from one service provider to another.

The new regulation obligates the companies to integrate security and privacy by design features in their products. To support the companies, the European Union Agency for Network and Information Security (ENISA) has released a report to provide a basis for better understanding of the current state of the art concerning privacy by design with a focus on the technological side.[13] However the report specifies several methods for securing communication, private data and user anonymity, it does not point out any recommendation for the actual implementation.

2.5.2 IoT Cybersecurity Improvement Act

This legislation was proposed by the US Senate to establish minimum requirements for federal procurements of connected devices. The regulation would force the internet-connected device vendors to provide certification ensuring that the device:

- Is capable of accepting properly authenticated and trusted updates from the vendor
- Does not contain any hardware, software, or firmware components with any known security vulnerabilities or defects listed in the National Vulnerability Database (NVD) or similar databases.
- Uses only non-deprecated industrial standards.

[8] http://eur-lex.europa.eu/eli/reg/2016/679/oj
[9] https://www.scribd.com/document/355269230/Internet-of-Things-Cybersecurity-Improvement-Act-of-2017
[10] https://en.wikipedia.org/wiki/Cyber-security_regulation
[11] https://ec.europa.eu/digital-single-market/en/proposal-eprivacy-regulation
[12] https://blog.nxp.com/security/protecting-the-i-in-the-iot-gdpr-and-future-challenges
[13] https://www.enisa.europa.eu/publications/privacy-and-data-protection-by-design

- Does not use any hard-coded credentials (passwords, user names, keys etc.).

The legislation also requires the vendors to notify government customers of newly discovered vulnerabilities and defects and provide updates to address these vulnerabilities in a timely manner.

Chapter 3 Threat modelling methodology

The aim of threat modelling is identifying threats, which are then rated and classified, then countermeasures are proposed to reduce risks. Several techniques exist for a systematic approach to cover as many aspects of the system as possible for revealing threats. Threat modelling is usually preceded by understanding the system and gathering information about it, typically by identifying stakeholders, assets to be protected and relevant security objectives. We may also need to specify a selection of attacker profiles, describing both internal and external actors in order to understand their motivation. Many methodologies exist to systematically enlist and describe threats to a system. In the following sections we describe the three most common approaches: attack trees, misuse cases and the STRIDE per element approach, defined in the Security Development Lifecycle (SDLC) methodology by Microsoft.

3.1 Attack trees

Attack trees were first proposed by Schneier [12] as a systematic way to look at threats. Attack trees are conceptual diagrams which reflect the anticipated options of attackers to achieve their goals.

An attack tree consists of a root node, and some internal and leaf nodes. The root node on the top symbolizes the ultimate goal of the attackers. From the bottom up, nodes represent the conditions which must be satisfied in order to make the direct parent node true.

Equivalently, these trees (which are actually acyclic directed graphs) can be seen as monotonous logical expressions on the leaf actions (see Figure 5), where necessary actions are joined by AND clauses and sufficient actions are joined by OR clauses as their common parent nodes. In the first case, every child node must be satisfied; in the other, even one is enough.

An actual attack consists of actions for which the logical expression corresponding to the attack tree is true. In short, when the root node is satisfied the attack is complete.

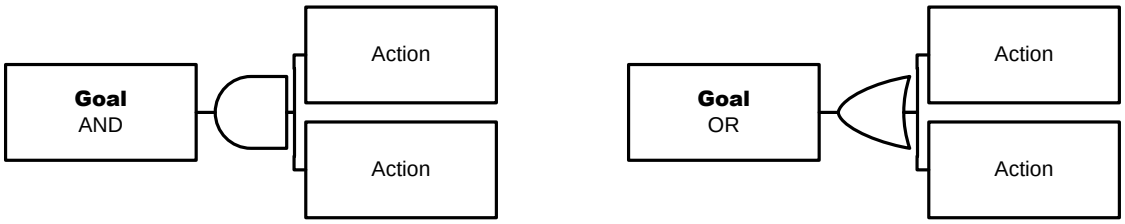

Figure 5: AND and OR parent nodes in attack trees

Such a representation of threat modelling with attack trees, being an iterative process, allows a lot more than just saving work on enumerating threats – it enables continuous incorporation of new information to the nodes based on lessons learned from past incidents that happened to different systems, like the possibility of an attack or the associated resources for a successful attack (for instance time needed to accomplish a certain step, associated costs or the preparedness of the adversary).

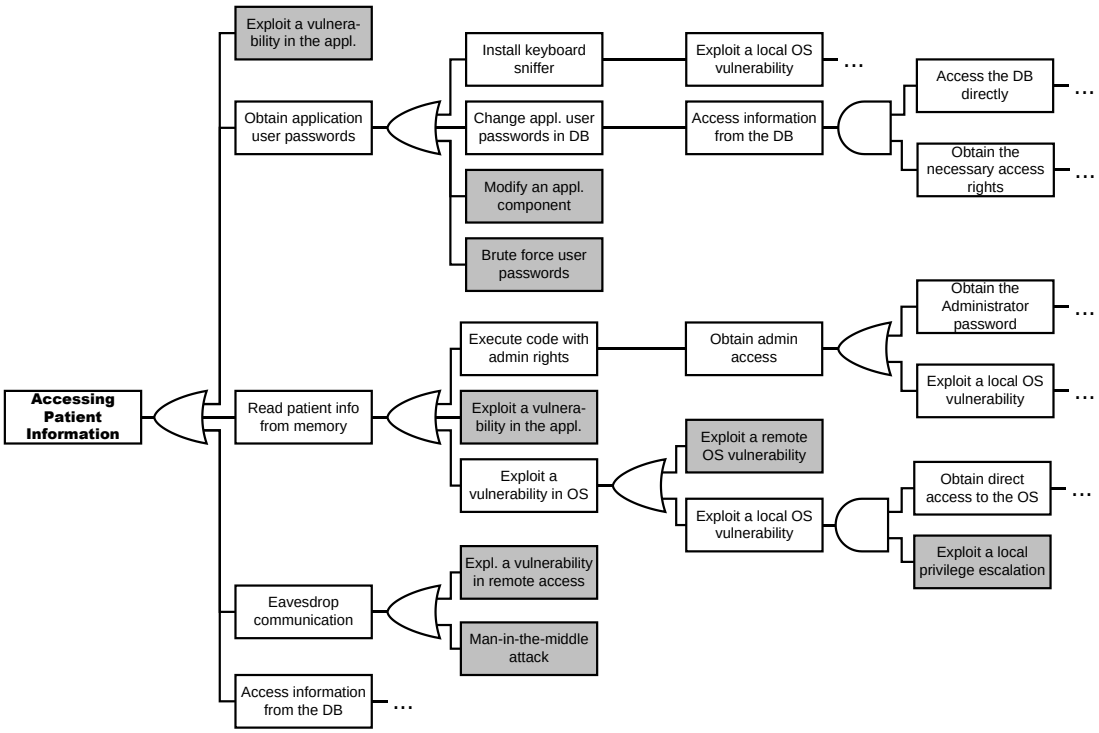

Figure 6: An example attack tree of an imaginary healthcare system

Attack trees express an inherent property of a given system and are thus implementation independent. This property allows reusing attack trees, such as including results from past work or use present ideas in the future.

Examples from literature regarding IoT include Nurse et al.'s [13] work of categorizing possible IoT insider attack vectors of 16 (8+8) different types, ranging from classic memory exploitation to data leaks. Later, Kammüller et al. [14] performed a threat analysis in IoT scenarios using the attack tree methodology. They mostly focused on insider threats that may affect the IoT system (calling them "smart insiders"). They focused on two of Nurse et al.'s 16 possible attack vectors: (I) using the storage system of the device for copying sensitive data, and (ii) compromising the communication channel for misconfiguration. These two vectors were later formally analysed and which resulted in a conclusion that they may be exploitable by many possible attacks.

3.2 Misuse/abuse cases

During requirements engineering, use cases have become increasingly widespread to describe and verify normal and correct ways of using a system for a certain purpose. Similarly, we can define misuse and abuse cases which are most commonly used for security perspectives. Misuse cases intend to describe unexpected usage or abnormal behaviour of the system, i.e. a selection of conditions, when the system does not work. Abuse cases are similar, but they describe intentional abnormal behaviour, i.e. what a hacker would intend to do with the system and define his or her requirements for a successful attack.

First suggested by Sindre and Opdahl [15], it goes beyond describing regular actors and use cases by shifting from the perspective of the owner's to the adversary's, and the ways use cases can be threatened, exploited or hindered. In addition, these diagrams also include countermeasures to mitigate the threats, thus one can be prepared for abnormal behavior and see if all applied protection techniques are sufficient. Misuse cases can be used together with attack trees.

Misuse/abuse case diagrams usually include textual description, which provide more details about these cases and the relationships between them, along with the appropriate requirements and sequences of actions.

0 below shows an example of a misuse case having a simple web shop, where the attacker's goal is to obtain specific data about a user: either the user's credentials or other confidential data specified during registration (such as credit card number).

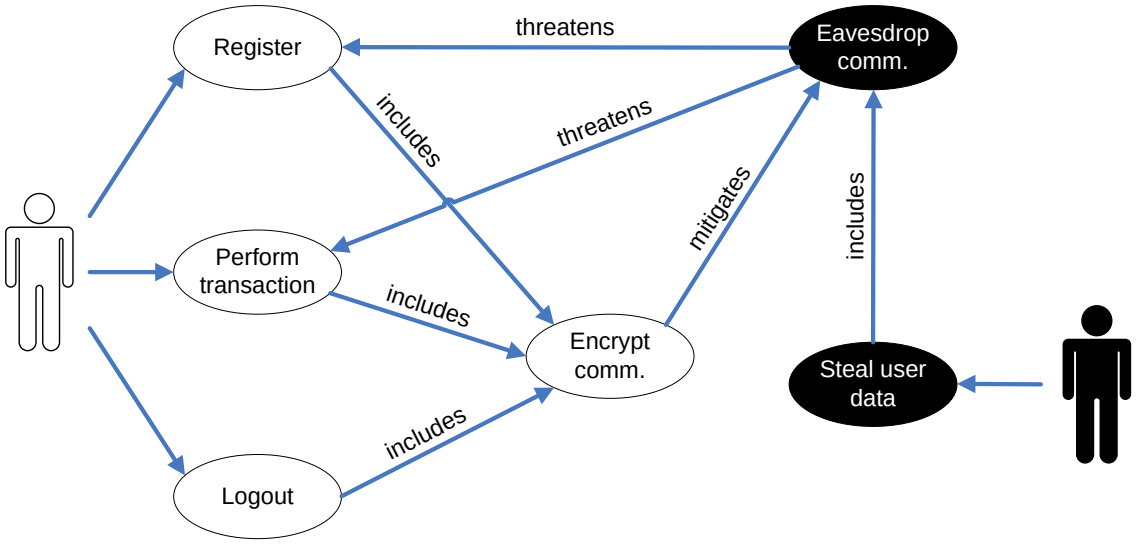

Figure 7: An example misuse case diagram

In the above described scenario, an attacker can eavesdrop (sniff) communication over the network, or can obtain some data of his or her interest from the user registration or user transactions. Defending against an attacker with these objectives is also relatively simple: by encrypting communication properly the eavesdropping attack is mitigated.

There are several extensions to the misuse/abuse case approach, for instance Røstad [16] complemented the diagrams with model elements representing insider threats and vulnerable functionalities of the system.

3.3 SDL threat modelling

The STRIDE[14] model assumes that attackers will follow one or more of the following attack vectors.

- **Spoofing identity**. An attacker will try to hide his identity or take another user's or platforms ID. Examples are illegally accessing and then using another user's authentication data or tokens or faking a network address.
- **Tampering with data**. The attacker might perform a malicious modification of data. This can be done on persistent data or on data flows in the network.
- **Repudiation**. The attacker will try to deny having performed an action, and make sure that other parties cannot prove otherwise.
- **Information disclosure**. Attackers will attempt to breach the confidentiality of information, i.e, disclose data to individuals who are not supposed to have access to it.
- **Denial of service**. Denial of service (DoS) attacks prevent valid user the access to a service, either by intercepting the communication paths or disabling the services itself (for instance by overloading it with bogus tasks).

[14] Microsoft, The STRIDE Thread Model, http://msdn.microsoft.com/library/ms954176.aspx, 2005

- **Elevation of privilege**. Unprivileged users or processes will attempt to gain privileged access, especially root or administrator rights. Through this access, the entire system could potentially be compromised.

Chapter 4 Security requirements for IoT

Security requirements can be defined as “non-ambiguous and verifiable statements implementing security objectives”; yet other definition state them being “constraints on the functions [that] … operationalize … security goals”[15]; whichever way, they should express the correct, intended system behaviour, rather than just enumerating undesired actions. There are several methods to define security requirements for a given information system, however, there is no universally accepted approach to define their extent[16]. In the following, we sketch the two main approaches:

Building requirements

The first approach consists in building requirements from the objectives, using elicitation techniques.

The CERT Coordination Center defined a methodology for requirements building and elicitation, called Security Quality Requirements Engineering (SQUARE)[17], which is referenced by US Department of Homeland Security[18]. Besides including rules for the prior threat analysis, this method recommends:

- the use of elicitation techniques, three of them being recommended[19];
- categorization of requirements (however SQUARE does not provide a definite method on this topic);
- prioritization of requirements, using Analytic Hierarchy Process (AHP);
- peer reviewing of requirements.

Selecting requirements

The second approach consists in selecting requirements from a standard catalogue in order to cover the objectives, and define ad-hoc requirements only whenever an objective is not completely covered by existing requirements. The most standard requirements catalogue is Common Criteria[20].

There is also a method called Security Requirements Engineering Process (SREP), which is a kind of cross-process including SQUARE and Common Criteria, including notions of reuse[21].

Setting security requirements is a challenging task, where the high level requirements should be considered during the design phase to achieve appropriate level of security. However, often the vulnerabilities are not found in the security architecture but instead inside the actual implementation of a given functionality. On the other hand, low level security requirements can be defined, but such requirements lose their portability, because a specific requirement for a system depends on the architecture and the applied technologies.

To achieve a middle ground, we inspected the IoT systems from a layered perspective. This way, we can set more detailed requirements for the IoT without being too specific. We applied this layered

[15] Ch. Haley, R. Laney, J. Moffett, and B. Nuseibeh. Security Requirements Engineering: A Framework for Representation and Analysis. IEEE Trans. Softw. Eng. 34, 1. January 2008

[16] I. Tondel, M. Jaatun, P. Meland. Security Requirements for the Rest of Us: A Survey. IEEE Software, vol. 25, no. 1, pp. 20-27, January/February, 2008

[17] Nancy R. Mead, Eric D. Hough, Theodore R. Stehney II. Security Quality Requirements Engineering (SQUARE) Methodology, Carnegie Mellon SEI - 2005

[18] See DHS - BSI website at https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/232-BSI.html

[19] Issue-Based Information System (IBIS), Joint Application Development (JAD), and Accelerated Requirements Method (ARM)

[20] Common Criteria for Information Technology Security Evaluation - part 2 : Security functional components - September 2012 - Version 3.1 Revision 4 – www.commoncriteriaporta.org

[21] Daniel Mellado, Eduardo Fernández-Medina, Mario Piattini. A common criteria based security requirements engineering process for the development of secure information systems – Elsevier - 2006

approach during the threat analysis and during writing the security requirements. We also investigated an approach of choosing some minimum requirements for the IoT, which have to be applied to ensure sufficient security. The latter approach led some additional recommendations for the actual implementations, how these requirements can be fulfilled.

4.1 Attack surface in the Internet of Things

Aligned with the approach of OWASP, the attack surface of a system can be defined as the boundary through which an attacker can interact with it, like feeding malicious input or extract information. It includes all inputs and outputs through which a user can influence and reach the system - application arguments and parameters, user interfaces, APIs, network connections, files from the file system, system settings, databases, inter-process communications, and so on. More formally, the OWASP approach defines the attack surface of an application[22] as:

1. the sum of all paths for data/commands into and out of the application, and
2. the code that protects these paths (including resource connection and authentication, authorization, activity logging, data validation and encoding); and
3. all confidential and sensitive data used in the application, including secrets and keys, critical business data and Personally Identifiable Information, and
4. the code that protects this data (including encryption and checksums, access auditing, and data integrity and operational security controls).

One can substantially increase security by reducing the attack surface. Given that all possible interfaces can be used by various actors, each with different roles and privilege levels, the complexity of the attack surface is usually enormous. This is why the usual approach is to group such vulnerable areas into categories based on their functionality, design, and the technologies utilized – such as administrative, transactional, or monitoring interfaces, authentication, forms, and so on.

Understanding the attack surface of the system is essential before source code analysis can be used to search for potential vulnerabilities. After the inputs and outputs have been identified, data can be followed from its initial point throughout the code to find ways it can cause undesired behavior by doing data flow analysis – using the same principle, such analysis can also show how sensitive information can be leaked from the system.

Possibly vulnerable areas should be categorized by risk level, especially when dealing with a performance and power-critical IoT environment. Even though the entire attack surface of the system may have been correctly identified, due to performance requirements, the countermeasures that have been deployed may not be complete and focus only on the most critical parts of the system (access to sensitive data, privilege escalation, or memory errors). Even these measures may be optimized for the resource constrained environment, and as such have significant limitations. In summary, whatever the scenario, our goal is to find a balance between covering all possible security threats and guaranteeing the functionality and availability of the system.

The attack surface in the domain of IoT systems is diverse and comprises numerous potential points of vulnerability, including software and data that reside with the product, communication channels, as well as remote data storage and processing, amongst others. Securing these pose a major challenge to organizations.

Atamli & Martin[23] were one of the first to identify possible attackers and attack vectors in IoT generic environments. In particular, they identified different sources of threats, viz: (I) malicious actors who own the IoT device willing to gain restricted role of the manufacturer; (ii) bad manufacturers, who

22 https://www.owasp.org/index.php/Attack_Surface_Analysis_Cheat_Sheet

23 Atamli, A. W., & Martin, A. (2014, September). Threat-based security analysis for the internet of things. In Secure Internet of Things (SIoT), 2014 International Workshop on (pp. 35-43). IEEE.

have the ability to exploit the devices gaining information about users; or (iii) external adversaries that are not part of the system but are capable of exploiting it.

In addition, Atamli and Martin presented different attack vectors for these systems including: device tampering, information disclosure, privacy breaches, (D)DoS (Denial of Service or Distributed Denial of Service) attacks, spoofing, or privilege escalation. All these possible attacks coming from the aforementioned sources can target several kinds of systems, depending on the particular environment, such as actuators, sensors, RFID tags, and Network (conventional, NFC(Near Field Communication), or the Web).

Furthermore, OWASP IoT project[24] offers a comprehensive list of vulnerabilities for different attack surface areas:

24 https://www.owasp.org/index.php/IoT_Attack_Surface_Areas

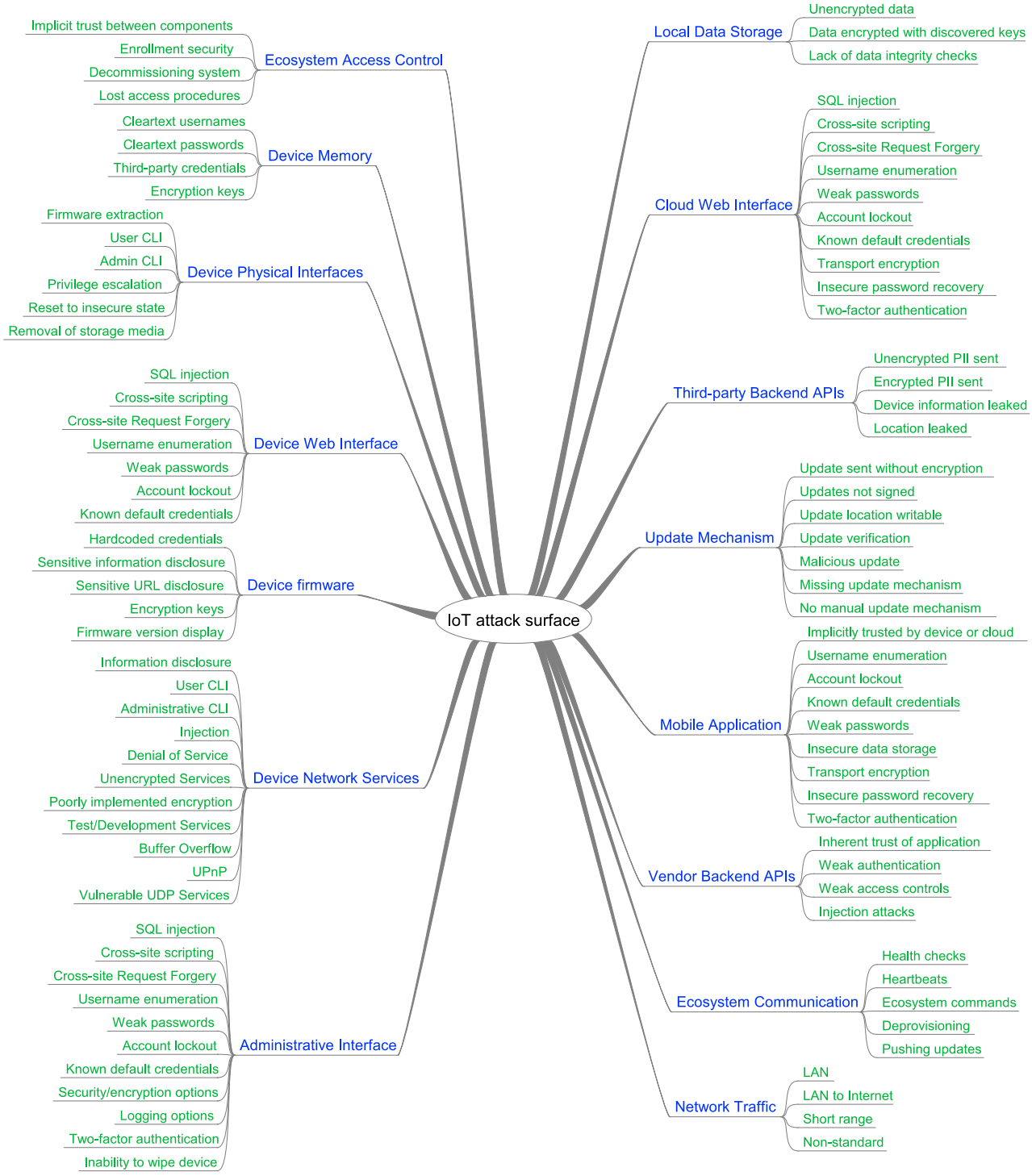

Figure 8: A list of attack surface areas for different IoT ecosystems and scenarios

These vulnerabilities offer a generic approach for IoT systems, and can be used to derive security properties, which will be studied afterwards.

4.2 Threats

Using the three layer architecture described in section 2.1, we can discuss the security aspects of each layer separately. As every layer has different security concerns (described in the table below), we can focus on the most important problems at each layer separately. In the following table, we grouped the most common security concerns listed by OWASP. Although this is not a comprehensive list of possible vulnerabilities, covering these issues greatly improves the overall security of an IoT product.[25]

Security concerns	IoT Service Ecosystem	Communications Network	Endpoint Ecosystem
Insecure web interface	X		X
Insufficient authentication/authorization	X		X
Insecure network services	X		
Lack of transport encryption		X	
Privacy concerns	X	X	X
Insecure cloud/service interface	X		
Insecure device interface			X
Insecure security configuration	X	X	X
Insecure software/firmware	X		X
Poor physical security		X	X

Table 4: Security concerns for the different IoT layers

4.2.1 Security of Endpoint Ecosystem

The basis underlying an IoT network generally comprises of a variety of interconnected endpoint devices that acquire, process and exchange data. On this layer a number of technology-related challenges affect the security of the network and make the implementation of sophisticated security features difficult. These include, but are not limited to:

- Heterogeneity: The IoT connects devices that may tremendously vary in terms of complexity and capabilities, may come from different vendors, and may have been designed for different overall functions. Also, a number of heterogeneous wireless technologies, such as WiFi, Bluetooth, Zigbee, GSM, etc., may be used to connect the devices.
- Resource constraints: IoT devices are normally subject to technical constraints in terms of processing power, memory, power consumption, and cost, bearing negative implications on eventual security features.
- Homogeneity: When deployed in batches, IoT devices may consist of similar or identical devices, with any security related vulnerabilities being common to all devices.
- Deployment and updates: IoT devices may be deployed with an intended service life-time that is significantly longer than the life cycle normally anticipated for electronic devices. This can pose a substantial challenge to providing long-term support, for example when a device outlives its own manufacturer. Furthermore, the circumstances of deployment might allow reconfiguring or updating a device only with difficulties, or not at all.
- Scalability: IoT devices can be deployed on a massive scale, and networks can consist of a large number of individual nodes. Any security measures that are applied in such networks must be appropriately scalable and take into account the potential quantity of interconnected links.

[25] https://www.owasp.org/index.php/IoT_Security_Guidance

Said technological factors pose a challenge when it comes to addressing multiple security-related threats and potential attacks that concern IoT endpoint devices. A selection of those will be briefly identified in the following and some hints on potential mitigation strategies will be given.

Physical Capture or Tampering: IoT endpoint devices may be deployed in public or generally untrusted areas where the implementation of physical security is difficult or impossible and direct access to the device cannot be effectively restricted. The attacker may obtain full control over the captured device and the information stored on it. Among others, the following threat scenarios may arise:

- Attackers may *read and manipulate the internal memory and firmware of a device*. This can be achieved by accessing debugging and programming interfaces that have been left enabled on the board (e.g. JTAG). A possible countermeasure would be to disable such interfaces before the deployment of the device. However, it may still be possible to unsolder certain parts of the device, such as flash memory chips, and access them independently of the rest of the system. Accessing a device's memory and firmware also commonly facilitates reverse engineering which may enable the attacker to discover and exploit additional vulnerabilities. In addition, secrets and cryptographic material may be extracted from the device, which potentially enables the attacker to access communication infrastructure (e.g. WiFi) or web services provided for the IoT device. Here, the implementation of measures that physically protect the information on the device will be necessary. Examples of these include hardware-accelerated flash encryption, Trusted Platform Modules (TPMs) and storing secrets not directly in the device's flash memory but securely using, among others, One-Time-Programmable (OTP) fuses or Physically Unclonable Functions (PUFs).
- *Side channel attacks* may enable the attacker to gain information about secrets that are stored or processed on the device without directly accessing its memory. This can be done by monitoring physical effects of the device, such as power consumption, electromagnetic radiation, and even sound, amongst others, which may let the attacker gain information about the data processed. Special algorithms that avoid the leakage of such information can be used as countermeasures to these attacks, and further physical shielding is required to ensure sufficient security.

Attacks on Availability: Attacks on availability include any actions that are physically or logically applied to the IoT end node by a malicious party in order to make it stop working, the most prominent of which include *Denial-of-Service (DoS)*, *Distributed-Denial-of-Service (DDoS)*. Those will be discussed in more detail in the next subsection, which deals with network layer threats. *Physical capture* of a node may also compromise its availability at the attacker's convenience. Another attack on availability that directly applies to IoT endpoint devices is the *Sleep Deprivation Attack*. As IoT endpoint devices usually have constraints on their desired power consumption or are battery-powered, they potentially implement a power conserving sleep mode which they routinely enter. During a Sleep Deprivation attack the attacked device is interacted with in an apparently legitimate manner that, however aims to break scheduled sleep cycles or keep the device from entering them. This causes an excessive power consumption which can result in failure or shut down of the device.

Eavesdropping/Interception/Hijacking: Most IoT devices communicate using wireless networks, with adversaries being able to eavesdrop and intercept the data transmitted. The peculiar aspects of IoT endpoint devices, such as constraints on processing power or memory, large-scale deployments and heterogeneous devices, however hinder the implementation of security features such as encryption and make key management difficult.

- **Data sniffing:** Sniffing is usually used in network traffic context, however this method is not limited to this domain. Sniffing is also feasible on a hardware level, monitoring internal system buses and chip interconnections. These channels are assumed to be protected from the software developer perspective, since these are implemented in the hardware. Countermeasures against this kind of attack can include encrypted transmission on the sensitive channels, or using hardware components (BGA package, 4 layer PCB), where sniffing is unfeasible.

- **Surveillance:** Surveillance is a specific type of access to information that combines the basic information access with a focus on personal/private data and the use of hardware to gather information from the physical world, for example by (ab-)using microphones, cameras, or location data. Typical personal mobile computing environments comprise various sensors that can be abused to provide strong surveillance capabilities.
- **Data tampering/Spoofing:** Comparable to surveillance threats, the tampering or spoofing of data on mobile computing devices can have wider impact than typical data tampering: Spoofed location, audio or visual data can lead to a variety of abuse scenarios.

Access control: IoT devices often have to provide some kind of administration access. Depending on the IoT platform, this can be a physical connection, a specific network protocol, remote administration via a cloud platform, or a web interface via HTTP. In every case, proper authentication and authorization is necessary to prevent threats such as private data leakage and unauthorized modification of settings.

Web interface security: Web administration interfaces are tending to be common in IoT devices, especially for complex ones such as routers, IP-cameras, but even in thermostats[26]. Any web interface may contain common web vulnerabilities (see the OWASP top ten[27] for most critical web application security risks). Since IoT devices have limited resources, most of the code handling HTTP requests, including the web server and the application logic, are written in low-level languages. Therefore classic security issues, such as buffer overflow[28], integer overflow, command injection [5], and even format string[29] vulnerabilities may be present in the IoT device.

Attacks against sensors: The behaviour of an IoT device depends on the data collected from the physical world by the various sensors. If the sensor data is faked or spoofed somehow, the IoT system will make wrong decision based on the modified data.

- In some special cases, e.g. activity trackers, a malicious user (owner) of the device might want to fake sensor data to gain financial advantages from it, for example by saving from insurance or gaining rewards[30] based on the collected data.
- To influence sensor data the attacker needs physical access to the sensor. Since sensors have to collect data from the physical world, attackers generally have a chance to access them. Moreover, some sensors can be influenced from distance also. For example, Trippel et al. [2] demonstrated an attack against accelerometers with acoustic injection. They could inject faked sensor data with acoustic waves to accelerometers from 5 different manufacturers. A similar attack was shown by Zhang et al. [3] by sending inaudible voice commands to a speech recognition system. Besides acoustic waves, electromagnetic interferences can cause false sensor data, as shown by Park et al. [4] for medical devices.

4.2.2 Security of the Network Layer

The purpose of this layer is to transmit data between the end nodes inside an IoT network. These connections can be between two devices, between the device and a server, or between the device and the user (user's PC or mobile phone). There are some specific challenges that have to be addressed in the scope of the IoT, particularly in case of low-resource devices, and that have an impact on the overall security of the whole network. These challenges include:

- Network energy efficiency (UDP protocol is favoured over TCP)
- Low bandwidth network

[26] https://blog.newskysecurity.com/iot-thermostat-bug-allows-hackers-to-turn-up-the-heat-948e554e5e8b
[27] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
[28] https://courk.fr/index.php/2017/09/10/reverse-engineering-exploitation-connected-clock/#Bluetooth_Communications_Reverse_Engineering
[29] http://defensecode.com/whitepapers/From_Zero_To_ZeroDay_Network_Devices_Exploitation.txt
[30] https://www.tomsguide.com/us/fitness-trackers-insurance,news-23053.html

- Network outages

To achieve a secure IoT system, the following threats should be addressed in the scope of network layer:

Eavesdropping/Interception/Hijacking: Most IoT devices communicate using wireless networks, which are vulnerable against adversaries, who are capable of intercepting the data transmission. For this purpose, it is important to provide secure communication channels between the parties to ensure data confidentiality. However, the keys which are used for encryption must be secured as well, since if these keys are compromised, the security of these channels cannot be granted. Even in case of low-resource devices, the encryption should rely on public key cryptography or derived keys from configurable secrets, otherwise the attacker will be able to obtain the encryption key and eavesdrop the communication, such as in the case of Linx light bulbs[31].

Spoofing of Identity and Data: During network spoofing, an attacker masquerades network information (IP, MAC address etc.) in order to gain illegitimate advantage on a given network. This type of threat includes the following attacks:

- **ARP poisoning:** The attack is achieved when an attacker poisons the ARP cache of two devices with the (48-bit) MAC address of their Ethernet NIC (Network Interface Card). Once the ARP cache has been successfully poisoned, each of the victim devices sends all their packets to the attacker when trying to communicate to the other device[32]. This kind of attack opens up for other malicious activity, like MITM or session hijacking attack initiated from the same network.
- **DNS cache poisoning:** The DNS server translates domain names into IP addresses. With DNS cache poisoning the attacker wants to redirect users from a specific domain to an attacker's controlled one. To perform the cache poisoning attack, the address record of the attacker's domain contains information related to the target domain, which may be cached by the DNS server.
- **IP address spoofing:** The attacker may want to use forged IP address to impersonate other system or perform Denial-of-Service type attacks. The attack can be performed easily by overwriting the source address in an IP packet; however it has different effects with different protocols. For example, in case of the UDP protocol, the attacker can send data with spoofed address, but won't receive the answer. But in case of the TCP protocol, which requires a simple handshake, the modified source address can be used only to cause DoS attacks.

Countermeasures against identity and data spoofing can involve different authentication methods, where trust is given based on the unique identifier of the device or the usage of public key cryptography, e.g. in the case of TLS/SSL or DTLS.

Attacks on sensor network routing: A lot of attack potential arises with the custom communication protocols introduced in the IoT network. General ad-hoc routing protocols are often susceptible to different kinds of attack. When an IoT network is designed, it is recommended to choose an appropriate network protocol to prevent this kind of attack. The following attacks are the most common against sensor networks according to Raymond et al. [6] and Karlof et al. [7]:

- **Spoofed, altered, or replayed routing information:** The most direct attack against a routing protocol is to target the routing information exchanged between nodes. By spoofing, altering, or replaying routing information, adversaries may be able to create routing loops, attract or repel network traffic, extend or shorten source routes, generate false error messages, partition the network, increase end-to-end latency, etc.
- **Selective forwarding:** Multi-hop networks are often based on the assumption that participating nodes will faithfully forward messages received. In a selective forwarding attack,

[31] https://www.contextis.com/blog/hacking-into-internet-connected-light-bulbs

[32]https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11_603839.html

malicious nodes may refuse to forward certain messages and simply drop them, ensuring that they are not propagated any further. A simple form of this attack is when a malicious node behaves like a black hole and refuses to forward every packet she sees. However, such an attacker runs the risk of neighboring nodes concluding that she has failed and deciding to seek another route. A more subtle form of attack is to selectively forwards packets. An adversary interested in suppressing or modifying packets originating from a few select nodes can reliably forward the remaining traffic and thus limit suspicion of her wrongdoing.

- **Sinkhole attacks:** In a sinkhole attack, the adversary's goal is to lure nearly all the traffic from a particular area through a compromised node, creating a metaphorical sinkhole with the adversary at the center. Because nodes on, or near, the path that packets follow have many opportunities to tamper with application data, sinkhole attacks can enable many other attacks (selective forwarding, for example).
- **Sybil attacks:** In a Sybil attack, a single node presents multiple identities to other nodes in the network. Any system whose correct behavior is based on the assumption that most nodes will behave properly may be at risk for Sybil attacks. In a scenario, where the system behavior is based on some form of voting system between the network nodes, an attacker with sufficient identities can manipulate the end result of such a scheme. The Sybil attack is especially threatening to fault-tolerant schemes such as distributed storage, dispersity and multipath routing, and topology maintenance
- **Wormholes:** In the wormhole attack, an adversary tunnels messages received in one part of the network over a low latency link and replays them in a different part. The simplest instance of this attack is a single node situated between two other nodes forwarding messages between the two of them. However, wormhole attacks will more commonly involve two distant malicious nodes colluding to understate their distance from each other by relaying packets along a channel available only to the attacker
- **HELLO flood attacks:** Many protocols require nodes to broadcast HELLO packets to announce themselves to their neighbors, and a node receiving such a packet may assume that it is within the (normal) radio range of the sender. This assumption may be false: a laptop-class attacker broadcasting routing or other information with large enough transmission power could convince every node in the network that the adversary was its neighbor, allowing the attacker to execute more sophisticated attacks. With proper protocol rules, this attack can be avoided.
- **Acknowledgement spoofing:** Several sensor network routing algorithms rely on implicit or explicit link layer acknowledgements. Due to the inherent broadcast medium, an adversary can spoof link layer acknowledgments for "overheard" packets addressed to neighboring nodes. Goals include convincing the sender that a weak link is strong or that a dead or disabled node is alive.
- **Jamming:** Jamming is defined as the act of intentionally directing electromagnetic energy towards a communication system to disrupt or prevent signal transmission [8]. Jamming attacks can be viewed as a kind of DoS attacks, which pose a threat to WSN (Wireless Sensor Network) even with strong security mechanism, simply by targeting the physical channels of the communication.

4.2.3 Security of Service layer

Most of the threats discussed here are not so different from any other server-side application. In most of the cases, these servers have web access, which without proper countermeasure pose a huge security risk for the server. The possible attack methods include code injection, session hijacking, arbitrary code execution etc.

Denial of Service: This threat is similar as in the other layers, but in this case making the server unavailable often makes the functioning of the IoT system impossible. As another source of concern, if a portion of the IoT devices are compromised, these can be used as part of a DDoS attack.

Privacy threats: In the case of IoT, a lot of data is collected about the user. This data can be processed on the end nodes, but in most of the cases it is collected on a central server. This data can be stored anonymously to avoid privacy leakage. However in some cases, the identity can be restored from this partial information as well [9].

API abuse: An Application Programming Interface (API) is a set of clearly defined methods of communication between software components[33]. APIs are regularly used by cloud service providers to allow access to their system. An example for the API abuse threat is the path traversal attack, where the attacker can access unauthorized files outside the restricted directory through using relative paths.

Unauthorized access: An attacker can exploit web vulnerabilities or server misconfigurations to access to the related services or sensitive data without proper authorization. In case of a typical IoT ecosystem, unauthorized access may provide access not only to the server functionality, but to the IoT device also.

Code execution: Code execution caused by an exploitable vulnerability in the server or the application layer code enables the attacker to access or modify all data and even the code executed by the server.

4.3 Security requirements for IoT

For specifying the requirements for IoT, we followed the same layered approach that we used in the threat examination. With this approach, we can separate the layers from each other, which is often the case in the real world scenarios, where the device vendor is separate from the network provider.

We give an overview of the most important security requirements in the scope of IoT. Instead of setting a complete requirement list, we tried to give a list of the most important requirements concerning the IoT. To achieve this, we set high level functional goals, and set other sub requirements to achieve the desired functionality.

4.3.1 Requirements for Endpoint Ecosystem

Implement secure TCB

To ensure endpoint security it is inevitable to implement some kind of Trusted Computing Base (TCB). From the Orange Book [10], TCB is the totality of protection mechanisms within it, including hardware, firmware, and software, the combination of which is responsible for enforcing a computer security policy. Any bugs and vulnerabilities occurring inside the TCB means a potential threat to the entire system. Systems that don't have a trusted computing base as part of their design do not provide security of their own: they are only secure insofar as security is provided to them by external means (e.g. a computer sitting in a locked room without a network connection may be considered secure depending on the policy, regardless of the software it runs)[34]. The TCB can be an internal part of the CPU, or a separate hardware element, such as SIM card, UICC or other kind of HSM (Hardware Security Module). Since the overall security depends on the security of the TCB, it is important to implement additional protection against additional threats such as power analysis and glitching attacks, or reverse engineering and microprobing attacks. In addition to the aforementioned functionalities, the following functionalities can be implemented using the TCB to achieve additional protection:

- Endpoint application image validation
- Network authentication and/or peer authentication
- A separation of duties

[33] https://en.wikipedia.org/wiki/Application_programming_interface
[34] https://en.wikipedia.org/wiki/Trusted_computing_base

- Provisioning and personalization
- Isolated environment (connectionless site) provisioning and communication
- Cryptographically secure randomization

Implement secure Endpoint Identity and Authentication

In order to prevent cloning of an IoT device, the device must be capable to prove that it is manufactured by the IoT service provider. In order to do this, the endpoint is required to incorporate a trust anchor into the TCB. To enhance security, the endpoint can utilize personalised cryptographic keys, in order to minimize the impact of a compromised trust anchor. Another form of a trust anchor is using HSM as an application layer trust anchor, which over secure key storage can provide other crypto-processing functionalities.

Implement secure Firmware and software against tampering

In order to prevent tampering of executable code on the endpoint, it is recommended to implement a Minimal Viable execution Platform (MVeP. This platform is capable of configuring peripherals, authenticate code snippets, which will be executed by the CPU, and manages software updates. This way, a minimal bootloader can be defined, which can check cryptographically signed application images, ensuring that the image is from a trusted source. In order to further increase security, security critical code such as the first stage bootloader or the TCB should be stored in read-only memory.

Implement secure communication between services and Endpoints

In an IoT network, not only the Endpoints must be authenticated by the services, but the services must also be authenticated by the Endpoints, so critical services, such as application updates, cannot be subverted. Although clear text messages between the Endpoint devices are not strictly prohibited, it has to be ensured that communication channels with privacy data, commands or system critical messages are secured. For this purpose, the Endpoint has to be capable of authenticate another endpoint, encrypt/decrypt critical data and check integrity of a message.

Consumer privacy requirements

Since the IoT devices have the ability to interact with their environments, this raises a lot of security concern in the field of what data should be handled, and how. It is also important to inform the user properly about what data is collected, and give him the ability to decide whether he wants to expose this data to third parties. This is not limited only to personal data, since many endpoint specific data can be used as a fingerprint (BLE, Wi-fi, cellular address etc.) to track down a user, if the malicious actor can link these data with the actual user.

Use exploit mitigation and hardening techniques

Similar to any other computer system, IoT devices may contain exploitable vulnerabilities. The possibility of a vulnerability can be decreased, but it cannot be avoided completely. So, the IoT device should implement any possible mitigation and hardening techniques, which is possible. In some case, especially the low-power devices executing a Real-Time OS cannot support all of the following exploit mitigation and hardening techniques.

- Enforce memory protection with implementation of Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR) techniques.
- Use internal memory for secrets and delete them if the secret is not used anymore.
- Run applications with least privilege levels necessary.
- Use buffer overflow protection measures, such as stack canaries.
- Enforce operating system level security enhancements.
- Minimize hardware and logical access and remove any unnecessary debug port or logical access.
- Use secure default configurations with particular regard to enforced authentication, supported strong authentication mechanisms, use of encryption features (see also Crypto) and reliable authorization components.

4.3.2 Requirements for Network Layer

When it comes to designing the communications architecture of an IoT system, there is one pitfall that is the root of most issues – using custom cryptography, or no cryptography at all. As in any reasonably complex architecture the network layer encompasses several layers of third party infrastructure, it opens up an enormous attack surface if left unsecured. Designing and implementing protocols that can stand the test of time against attackers is no small feat – and as industry standards have already been established, implemented and tested, it should not be attempted at all.

By far the most widely deployed solution for securing the communications between two parties is Transport Layer Security (TLS). This protocol and its assurances have been extensively documented, reviewed and updated, as necessary to create the current version (1.2) of the protocol. While TLS is based on TCP, Datagram Transport Layer Security (DTLS) has also been defined for applications that require UDP based communication. Both protocols have publicly available, open-source implementations, even for embedded systems, and most common operating systems support them out of the box. These implementations are subject of significant public scrutiny, and as such should be preferred over any custom solutions. **TLS and DTLS should be used to secure all communications.**

TLS has a plethora of configuration options, and some configurations are more secure than others. As of version 1.2 of the standard, these best practices should be followed:

- Only allow connections using version 1.2 of the standard – neither the server nor the client should permit earlier versions.
- Only allow cipher suites that:
    - Provide forward secrecy – even if the private keys used by the client or the server leak, previously sent data will remain undecipherable to third parties.
    - Use AES for encryption – earlier algorithms, such as DES or RC4, have known weaknesses that may undermine the security of the whole protocol.
    - Use at least SHA256 for data integrity – earlier algorithms, such as MD5 or SHA1, have known weaknesses that may undermine the security of the whole protocol.
- Some cipher suites may use Elliptic Curve Cryptography (ECC) for key exchange – whether this is desirable depends on performance requirements and available hardware acceleration.
- The performance overhead of 256 bit AES over 128 bit AES is usually not worth the rather small increase in security.

While it is out of the scope of this document to examine all available cipher suites, the following two match these requirements:

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256

Since TLS depends on a Public Key Infrastructure (PKI), another issue that should be addressed is generating certificates and establishing trust between devices. Without a well-designed PKI, the protections of TLS can be easily defeated. While TLS can be used without authenticating the other party or only performing authentication of the server by the client, this is discouraged. Mutual authentication is recommended to be used for all communications.

A guide for establishing the PKI is out of the scope of this document, but the following best practices should nonetheless be followed:

- Private keys should be generated on device, preferably stored in a hardware security enclave – or if unavailable, in a software enclave provided by the operating system.
- Private keys should be at least 3072 bits (RSA/DSA) or 256 bits (ECC) long. Shorter keys – 2048 bits (RSA/DSA) or 224 bits (ECC) – can be used on performance-constrained devices.

- Have clear policies for signing Certificate Signing Requests (CSRs) when deploying hardware or software. Ensure that only authorized personnel can perform such an operation and that all signatures can be audited.
- Certificates should use at least SHA256 as their hash algorithm, and should be clearly identified by their metadata.
- The chain should have at least one layer of intermediate certificates. The private key belonging to the root certificate should be only used on an offline machine to sign intermediary certificates.
- Certificate Revocation Lists (CRLs) should be maintained and checked against on each connection. Certificates suspected of compromise should be revoked and reissued.
- On device redeployment existing certificates should be revoked and new certificates should be issued.
- Have clear policies in place for updating certificates – avoid using certificates that cannot be updated on embedded devices.
- Only include trusted root certificates in the trust store of individual devices. Ensure that the trust store can be updated when needed.
- Have separate chains of trust for certificates that have no way to expire or cannot be validated against a CRL – for bootloaders or code signatures, for example. Treat the private keys associated with these certificates with extreme care, use them only in offline environments and set up strict physical access controls.

While this list is by no means complete, a PKI set up using these guidelines combined with the latest version of TLS can ensure the confidentiality and integrity of all communications. To help ensure the availability of the system, and to further increase its resilience against different attacks, a number of additional tools and measures can be used on the network level:

- **Firewalls** should be used on the edge network to monitor and filter incoming and outgoing network traffic. Network layer firewalls allow packets to pass through based on a predefined set of rules, such as the IP address or port of the source or destination. More complex firewalls might have additional services, such as:
    - *Deep Packet Inspection (DPI)* – examines the data part of a packet, searching for protocol non-compliance, viruses, spam or other malicious content. It may allow a packet to pass, may redirect it to a different destination, or may reject it altogether.
    - *Intrusion Detection System (IDS)* – monitors the network for malicious activity or policy violations. It matches traffic to a library of known attacks. Once an attack is identified, or abnormal behavior is detected, an alert can be sent to the administrator or automatic action can be taken.
- **ARP whitelisting** should be used to ensure that attackers cannot exploit the lack of authentication in the ARP protocol. It is done by software that relies on some form of certification or cross-checking of ARP responses. This way, uncertified ARP responses are blocked, preventing ARP spoofing attacks.
- **DNSSec (Domain Name System Security Extensions)** should be used to protect the integrity of DNS data used by applications - such as DNS data forged or manipulated by DNS cache poisoning – by ensuring that all answers from the DNSSEC protected zones are digitally signed.
- **TCP/IP cookies** (also referred to as SYN cookies) should be turned on to prevent SYN flood attacks. This technique allows a server to drop connections in case the SYN queue fills up - after sending back an appropriate SYN+ACK response to the client - and enables it to reconstruct the connection after it gets a valid ACK response from the client.
- **Router** level filtering should be used to ensure that the private address space does not leak out into the global internet and to filter incoming network traffic as a protection against DoS attacks.
- **VPN (Virtual Private Networks)** should be used to provide confidentiality in such a way that even if the network traffic is sniffed at the packet level, an attacker would only see encrypted data. As opposed to consumer VPNs, the objective is not to make online connections

anonymous, but to increase privacy and security by adding an additional layer of encryption and authentication to all protocols running on the network. Even if all communications are otherwise encrypted using TLS, a VPN tunnel can add additional security by encrypting the headers and service packets of lower level protocols.

- **Centralized logging** should be deployed to log all security incidents and possible malicious behavior that has been identified for later audit. While logging all events might be a serious performance burden, at the least failures and errors should be logged.

4.3.3 Requirements for Service Ecosystem

Implement Service TCB

Implementing a TCB is important on every platform to guarantee security, however depending on the on the system, it can have different characteristics. In the service ecosystem, this can be achieved through application images, which are deployed on the servers. These images contain all of the information, which are required to a server to function properly, including executable files, configuration files, and other metadata. This method is also useful for more flexible functioning of the service, since servers can be deployed dynamically depending on demand.

To achieve this, the service provider has to standardize hardware and software, and to configure it according to the needs. To ensure that the image is not tampered with malicious entities, the image must be signed, and an Organisational Root of Trust is required, which ensures that the images are signed properly. The Organisational root of trust can be a form of HSM and it can have additional functionalities, like authenticating other parties within the ecosystem.

To have a trustworthy system, the following additional countermeasures should be considered:

- each secret must be protected from abuse
- internal use of each secret must be verifiably tracked and monitored,
- each individual approved to utilize a secret must use multi-factor authentication when accessing the secret(s),
- define a set of policies and procedures that enforce consistent and secure usage,
- build a process to sunset or revoke a certificate,
- identify whether a key has been abused and
- choose the correct set of cryptographic algorithms.

User authentication

Authenticating a user through an endpoint relies on both the trustworthiness of the endpoint and the communication channel between the service and the endpoint. Since the user is independent from the device endpoint, user credentials should be managed separately from the endpoint. During implementing user authentication, the following should be considered:

- *Enforce strong password policy*: The authentication system should enforce strong password policy, which instead of being complex (numbers + special characters) should be long enough to prevent brute force attacks. To prevent brute forcing, the service provider should limit the threshold for the total number of attempts, increase the minimum required time between the guesses or use captcha to prevent automatic trials.
- *Force authentication through the Service Ecosystem*: If it is possible, avoid user authentication on the field, since it is easier to bypass authentication on the endpoint. Instead, use the secured service channels, and API for this purpose.
- *Separate storage systems for duties*: If the application layer of the service is compromised through SQL injection or other attack methods, the service provider can prevent privilege escalation through physically separating the systems.
- *Consider using Network Authentication Services*: Network Operators authenticate the endpoint on the network layer. Since a lot of network operators enforce network-based authentication, if these tokens provide meaningful security, they can be reused for application

level authentication, so the service provider don't have to maintain its own secure store technology.

Requirements to improve availability

Denial of service attacks are common on the internet, and pose a major threat to the service ecosystem. To avoid this kind of threats several countermeasure techniques can be applied.

- **Harden systems exposed the Public Internet:** Use DDoS resistant, load balancing infrastructure, which contains redundancy systems and firewalls.
- **Systems Logging and Monitoring Approach:** Each system must be monitored in order to detect anomalies.
- **Define a Recovery Model:** In case of security compromise, the system must be able to revert back to a safe version.

Countermeasures against endpoint compromising

- **Define secure boot process:** the service during boot should be able to authenticate itself to external endpoints, acquire identity etc.
- **Define a Persistent Storage Model:** a lot of cloud based service do not implement persistent storage model, since the resources are allocated on demand. However if the service provider needs persistent storage for its services it should be designed in a way, that in an event of a compromising, the attacker can't access other user's data.
- **Administration:** To troubleshoot and diagnose application faults, some kind of administration model should be implemented. To achieve this, system changes should be tracked, and two factor authentications should be considered to enhance security.

Countermeasures against anomalous endpoint behaviour

For the overall health of the service ecosystem, it is important to determine anomalously behaving endpoints, to protect the services from further compromising. Since it is not usually possible to completely exclude every source of malicious activity (environmental factors, social engineering), it is recommended to equip the services with extra protections against these threats. The following techniques can be implemented to enhance security:

- **Input validation:** one of the easiest routes of attack for an adversary is to abuse the data originating from the endpoint, resulting in unwanted behaviour in the service system. For this purpose, it is important to check the incoming information for malicious parts.
- **Implement output filtering:** complementing the Input validation, data leaving the services should be also filtered against malicious code or to prevent the exposure of confidential data.
- **Intruder detection system:** in order to detect malicious activity, the service should be capable to monitor its own network against malicious activity. Honeypot mechanism can be implemented in order to detect and deflect unauthorized users within the system.
- **Incident response model:** During a security breach, it is important to the service provider to react quickly to external attacks, which involve cleansing or completely shutting down services, detect the source of compromise, restart and patch systems on the whole infrastructure.

Chapter 5 Risk assessment techniques

As has been stated in previous chapters, the rapid growth and interconnectedness of the IoT gives rise to a multiplicity of security-related risks and the proper identification of exposures makes a solid risk assessment a necessity. This chapter therefore aims to provide an outline of the principal process involved as well as a discussion on different risk assessment tools and their mutual relevance in context of VESSEDIA. Two major standards for risk assessment, ISO/IEC 31010:2009 and NIST Special Publication 800-30 are briefly outlined and compared. Then we present different risk assessment techniques based on the ISO/IEC 31010:2009 standard. We give key considerations on those techniques as well as relevant characteristics of the techniques in the context of software safety and security verification tools.

5.1 Risk assessment for IoT applications

Organizations and enterprises regardless of their nature pursue a certain set of objectives and expend effort towards reaching those objectives. The achievement of the latter can be affected by what will be subsequently referred to as *"risk"*.

ISO 31000, which is used as a reference within ISO/IEC 31010:2009, defines risk as the *"effect of uncertainty on objectives"*, with the effect being a *"deviation from the expected – positive or negative",* and can be described as a deficiency in information regarding potential events and their consequences.[35] [36]

It is important to note, that the consequences of risk as referred to above, can be positive as well as negative, being able to enhance as well as to diminish the achievement of an objective.

NIST Special Publication 800-30 [11], on the other hand, refers to risk in a more specific way by stating that risk is a *"measure of the extent to which an entity is threatened by a potential circumstance or event"*, with the consequences of potential events being *"adverse impacts"*. Here, *"risk"* bears an inherently negative meaning.

Similarly, NIST SP 800-30 also employs a different and more complex terminology when it comes to potential events, labelling them *"threats"*, which more specifically decompose into several different factors. For sake of brevity only the most important terms will be presented here; their meaning and relation to each other are briefly summarized in the following:

- Predisposing condition – a condition that in- or decreases the likelihood that threat events result in adverse impact (e.g. a facility being located in a flood-prone region or not)
- Vulnerability – a weakness in an information system that could be exploited by a threat source in the context of a predisposing condition
- Threat source - an intent, situation or method that deliberately or accidentally exploits a vulnerability
- Threat event – an event or situation that potentially causes adverse impact and is in turn caused or initiated by a threat source

The magnitude of risk, which is called the *"level of risk"* in either standard, is a function of the consequences or impact of an event as well as the likelihood of an event to occur, and the determination of both factors is an integral part of the respective risk assessment processes. Within NIST the term likelihood can be further decomposed into the likelihood that a threat event will be initiated and the likelihood of impact.

[35] https://www.iso.org/iso-31000-risk-management.html
[36] https://www.iso.org/standard/51073.html

The risk assessment itself is a structured process that serves the following main purposes:

- identifying, analysing and evaluating risks to an organizations objectives and
- providing evidence-based information to make informed decisions on how to treat particular risks.

The process of risk assessment as given in ISO 31010:2009 as well as in NIST SP 800-30 are procedurally similar and consist of the following principal steps:

- preparation for the risk assessment and establishing a context,
- conducting the actual risk assessment and
- exploiting the results.

Generally, the purpose of the first step is to establish a context for the assessment by identifying its purpose and scope, assumptions made, as well as the sources of information that is used as input for the assessment, among others.

In ISO/IEC 31010:2009, the actual risk assessment process comprises of the following main activities:

- *Risk identification*: find, recognize and describe risks, including risk sources, events, their causes and potential consequences.
- *Risk analysis*: understand the nature of a risk and determine its level of risk.
- *Risk evaluation*: determine if a risk is acceptable or tolerable to the organization or not and decide which risks need treatment.

The process given in NIST SP 800-30 is very similar in structure and provides a more fine grained description of the process that corresponds to its more specific terminology.

The concluding step of the process focuses on exploiting the acquired knowledge and findings of the risk assessment process. Within NIST SP 800-30 instructs to communicate the risk assessment results and to share information developed in the execution of the risk assessment, in order to support other risk management activities. ISO/IEC 31010:2009 specifically implements a step called *"risk treatment"*, in which options for changing the likelihood or impact of a risk are selected and agreed to.

In addition, communication and review of information involved is maintained continuously during the course of the assessment.

Both risk assessment methodologies are compared in the figures below.

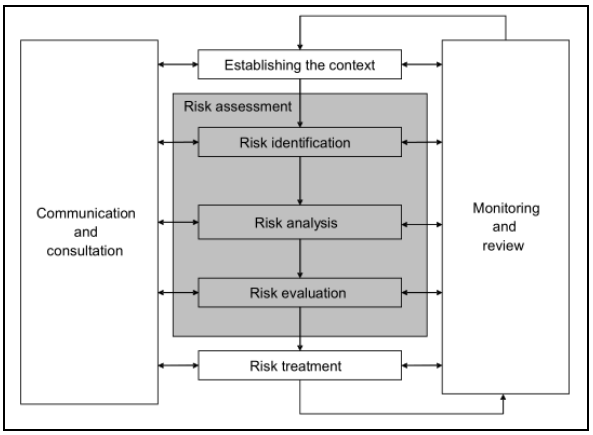

Figure 9: ISO/IEC 31010:2009 risk assessment process

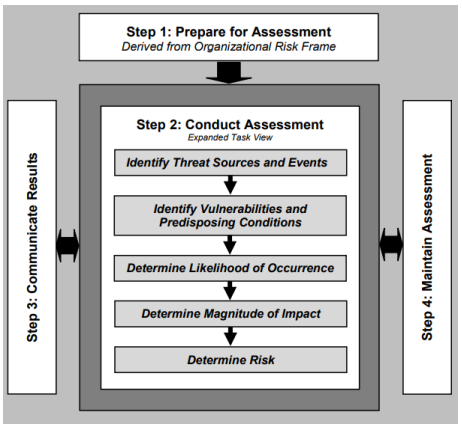

Figure 10: NIST SP 800-30 risk assessment process

ISO/IEC 31010:2009 as well as NIST SP 800-30 are both accepted as industry standards and aim at organizations of all types and sizes. While ISO/IEC 31010:2009 is an international standard, NIST SP 800-30 is the preferred risk assessment methodology of the US government and is thus heavily

US-focused and mostly aimed at the US public sector. However, NIST SP 800-30 is designed to be consistent with ISO standards and adequate with flexibility in mind, so that it can be used with other frameworks. As stated before, in contrast to ISO/IEC 31010:2009, NIST SP 800-30 also is more specifically focused on information security, while the former is kept more broad and generic. [37]

ISO/IEC 31010:2009 also provides an overview of various risk assessment techniques, which will be discussed in the subsequent section.

5.2 Risk assessment methods (based on ISO/IEC 31010:2009)

5.2.1 Introduction

This section discusses different risk assessment techniques/tools and their relevance in context of VESSEDIA. One of the VESSEDIA methodology's objectives is to offer a versatile and cost effective approach for certifying the use of security and safety verification tools during the software development lifecycle. The label "Verified in Europe" itself will provide added-value to system designers by documenting which toolset has been applied for verification.

In relation with the objectives described in the DoA, we consider a wider range of risk assessment techniques than the techniques of HazOP, Delphi, SWOT and FMEA, as they were stated in the DoA. Risk assessment is critical in order to understand the mechanisms behind the vulnerabilities, and increase safety as well as the security of connected applications and devices against risks, especially with regards to IoT systems.

Risks assessment techniques allow thorough risk appraisal, and help demonstrating the benefits of using the VESSEDIA methodology on connected applications, where improvements in verification lead to safer and more secure IoT environments. It is important to note that the above mentioned "Verified in Europe" label aims at supporting the IoT community stakeholders' efforts and building users' trust in IoT devices.

Various methods may be chosen to perform a risk assessment. While risk assessment lacks of consensus (for example in cloud computing[38]), we decided to consider the use of different techniques/tools and discuss their relevance if integrated to the VESSEDIA approach.

The tools are presented following the ISO/IEC 31010:2009 process of risk identification, risk analysis and risk evaluation. For each technique or set of techniques sharing similarities, we present key considerations and some information about their relevance in the context of safety and security verification tools:

[37] https://www.ncsc.gov.uk/guidance/summary-risk-methods-and-frameworks

[38] https://www.isaca.org/Journal/archives/2012/Volume-5/Pages/Cloud-Risk-10-Principles-and-a-Framework-for-Assessment.aspx

5.2.2 Risk identification tools

Risk identification is defined as the "process of finding, recognizing and recording risks" (ISO/IEC 31010:2009). Among the techniques considered "strongly applicable" for the purpose of risk identification, we have selected the following:

Technique	Key considerations	Relevance in the context of safety and security verification tools
Check-lists	This "look-up" method lists typical uncertainties. It is simple, and low resource consuming.	Available standards (e.g. ISO) and metrics allow to use well established listings for checking uncertainties on well-defined targets of verification.
Scenario analysis	Imagining future scenarios offers insight into possible outcomes. searching comprehensively for the sources of problems.	Those techniques bring insight on possible occurrences, for example when considering the quality in use of a given software (e.g. when embedded on an operating IoT hardware). Scenario analysis is very relevant towards environments characterized by technological changes (IoT).
Cause-and-effect analysis	Investigation of the contributory factors to an effect.	
Failure mode effect analysis (FMEA and FMECA)	FMEA is applicable for both system, service and software risk identification. It can be extended to consider criticality (FMECA).	The method can tackle issues ranging from design alternatives dependability to human errors identification.
Brainstorming	Those techniques are rather supportive to the above techniques through gathering opinions and finding consensus	Involve stakeholders (e.g. user, software developer, validator, certification body, hackers) in identifying new sources of risks given the versatility, volatility and innovative nature of IoT environments.
Delphi		
SWIFT-structured		

5.2.3 Risk analysis

The risk analysis phase is split in three distinct steps that are firstly the *consequence* analysis, secondly the *likelihood* analysis and finally, through combination of those, the determination of the *level of risk*. For simplicity and for narrowing down the range of available techniques for risk analysis, we have ruled out techniques which were not characterized as "strongly applicable", leading to the selection that follows:

Technique	**Key considerations**	**Relevance in the context of safety and security verification tools**
Root cause analysis	Is a type of scenario analysis, with the gathering a team, of the evidences on a failure or loss and through performing a structured analysis.	For verification purpose, there is a lot to learn from failure situations. This is especially true in context of complex systems such as IoT environments. The resulting documentation will prove useful for the community stakeholders in the future (e.g. validators, and certification bodies). Technological innovations may void learning from the past, and there it is relevant to continuously monitor the actual roots of new failures. Resource consumption and complexity are medium while providing deep insight.
Failure mode effect analysis (FMEA and FMECA)	The technique offers insight on failure modes and mechanisms as well as their effects.	This method is particularly interesting in that it can be applied along the software development life-cycle and adaptable to requirements in terms of verification efforts (i.e. VESSEDIA intends to create levels of applied tool capabilities). The method can provide both qualitative and quantitative inputs to other analyses techniques such as fault tree analysis.
Human reliability analysis (HRA)	As a supporting method, HRA considers humans' impact on the system performance.	As far as security matters, attack scenario give attention to attacker behaviors. On the other hand, much of **safety** issues will imply consideration on the misuse or mistakes done by users when interacting with the software/system and throughout its lifecycle. Damages to users and their surroundings are potentially critical in IoT environments. Operators and maintenance personnel can also be considered for treatment efforts.

Technique	**Key considerations**	**Relevance in the context of safety and security verification tools**
Reliability centred maintenance (RCM)	An industry spread method which is applicable throughout the whole risk assessment process. RCM identifies policies to implement for managing failures while efficiently reaching safety, availability and economic objectives.	The method is critically relevant to the maintenance stages of the verification process as described in D6.4.
Consequence/probability matrix	Coupling consequence and likelihood scales in a matrix table to produce a risk rating or risk levels.	Easy to use, the interpretation can be biased due to ambiguity and limited due to subjectivity. However, it can help for sharing understanding between stakeholders/members of the community about the impact of enhanced verification efforts.
Structure « What if? » (SWIFT)	This supporting method would be used in conjunction with a risk analysis and risk evaluation technique.	The technique gives opportunities for improvement of processes (e.g. verification efforts in the software development life-cycle).

If the need, time and resources allow, some dedicated techniques may be used for the purpose of each step of the Risk Analysis phase. The technique we present in the following table are again a selection of techniques described as *strongly applicable* (ISO/IEC 31010:2009):

Step of Risk Analysis	Technique	Key considerations	Relevance in the context of safety and security verification tools
Consequence	Hazard and operability analysis or HAZOP	The structured examination of a system will identify the risks and define the deviations from the expected or intended performances, rather than only focusing on well-known sources of accidents.	Despite its complexity, HAZOP has the advantage of giving the possibility to anticipate unforeseen events.
Probability	Fault tree analysis	The logical tree shaped diagram, starting from top with the undesired event, displays all the ways in which it could occur.	Fault tree can be built from a FMEA/FMECA during verification and be used both qualitatively (pathways to failure) and quantitatively (for probabilities), useful for systems with many interfaces and interactions.
Probability and Level of Risk	Bow tie analysis	The simple bow-tie shaped diagram displays causes and consequences of an event as well as the reviewing controls.	Not requiring high level of expertise to use, it is clear to display for example controls for prevention and mitigation. It may over-simplify the reality if compared with fault tree where simultaneous multiple causes can be illustrated.
Level of risk	Multi-criteria decision analysis (MCDA)	A range of criteria, which are assigned weights, is used to order between options in a decision-making process.	This technique is relevant in context of ranking criticality of alarms in the verification process. Where there are many alerts because of potential risks or vulnerabilities, the developer/validator needs a supporting tool to prioritize the alerts to handle.

5.2.4 Risk evaluation

In this phase, the level of risk formulated in the risk analysis is compared to the *risk criteria* set when *establishing the context*. The significance of the level and the type of risk supports the decision making process towards the risk treatment phase.

Technique	Key considerations	Relevance in the context of safety and security verification tools
Hazard Analysis and Critical Control Points (HACCP)	Specific characteristics are checked to be within defined limits through this preventive system for assuring product quality, reliability and safety of processes.	HACCP helps minimizing risks by controls throughout the process rather than through inspection of the end product. It can be applied on the stages of the verification process.
Root cause analysis	See in risk analysis techniques	See in risk analysis techniques
Failure mode effect analysis (FMEA and FMECA)	FMEA/FMECA allow to define significance levels. It also allows identifying how to avoid and/or mitigate failures and their effects on the system	As criticality of alarms is a major issue in security and safety verification, the method can serve the purpose of ranking alarms.
Reliability centred maintenance	See in risk analysis techniques	See in risk analysis techniques
Monte Carlo simulation	This technique evaluates the effect of uncertainty on the system considered.	Monte carlo simulation is a good candidate for supporting verification efforts at program lower levels, in that software algorithmic structure are prone to be fed with test inputs. However, being a resource consuming and complex technique, it may be difficult to engage stakeholders.
Bayesian statistics and Bayes Nets	This statistics approach displays variables and their probabilistic relationships.	While the requirements are simple, the definition of the interactions for complex systems is problematic. In addition expert judgment assumptions are needed on a multitude of conditional probabilities.
FN curves	FN stands for the cumulative frequency (F) at which N or more members of the population will be harmed.	It is not intuitive how to integrate this technique to VESSEDIA, but the related concept of ALARP, by showing when a risk threshold is outreached, can support decision making for resolving an alert or initiating a treatment.

Technique	Key considerations	Relevance in the context of safety and security verification tools
Risk indices	It is a risk estimate that uses ordinal scales. Scores can be applied to components of risks, allowing to rank the risks.	It is a good tool for ranking risks, but the numerical value has to be used with care, as the quantitative value does not necessarily have other purpose than allowing manipulation.

5.2.5 Conclusion

For sake of simplicity and comfort of use, we noted that some techniques are strongly applicable throughout the whole of the three phases of the Risk assessment process, namely:

- Environmental risk assessment
- Structure “What if?” (SWIFT)
- Reliability centred maintenance
- Failure mode effect analysis (FMEA/FMECA)

Among those, the FMEA/FMECA and reliability centered maintenance stand out. FMEA/FMECA is a unique technique in that it can be independently applied throughout the risk assessment process. This technique also showed strong relevance in the context of use of software safety and security verification tools. In addition, the HAZOP technique (figuring out possible deviations starting from unwanted outcomes) can be used in conjunction with FMEA (using component failures as a starting point) during the risk analysis stage. This provides comprehensive insight on security and safety matters during the verification process. Reliability centered maintenance is very relevant for the maintenance stages of the verification process.

Chapter 6 Formal specification of simple security requirements with ACSL

Static analysis approaches to software quality assurance range from a basic level, like compiler warnings, via heuristic-based tools, until mathematically rigorous formal methods. The latter are known to be powerful in their results, but they can be challenging to integrate in the development process.

Some formal static analysis tools, such as Polyspace, Astree, and the EVA plug-in of Frama-C use a formal method called *abstract interpretation* to reliably identify undefined behaviour in embedded software that can lead to serious run time errors. An advantage of these tools is that they can work on large programs. A principal disadvantage is that they only over-approximate the behaviour of the program under analysis, and hence produce often quite a substantial number of false alarms. The investigation of false alarms typically requires the intervention of experts which tarnishes the marketing claims of a high degree of automation.

On the other hand, formal static analysis tools that rely on *deductive verification*, such as the WP plug-in of Frama-C, can verify software properties whose complexity goes far beyond undefined behaviours. They require, however, that the expected behaviour is specified with explicit code annotations. These annotations are similar to Doxygen comments; however, they convey a machine-understandable semantics against which the source code can be formally verified. While this sounds very promising, our experience shows that complex properties can be tackled only for relatively small and well-designed software components. This is sometimes a problem in practice where unduly large and poorly designed software components regularly occur.

Abstract interpretation and deductive verification thus represent different ends of the spectrum of formal methods, their application areas apparently being quite complementary, yet they need not to be seen as unrelated approaches. In this chapter, we aim at combining the advantages of both approaches.

6.1 The concept of minimal contracts

In order to minimize the amount of manual specification effort, we are developing within the VESSEDIA project context a set of minimal function contract clauses, tailor-made to IoT security aspects.

Devising full contracts for a given set of function implementations usually is a substantial amount of work. In addition, each loop in the code needs manual provision of own appropriate invariants, and often additional *assert* clauses are to be interspersed as verification hints to the provers employed by Frama-C/WP.

In the context of VESSEDIA, where the main emphasis is on security properties, we can thus assign the available quality assurance efforts in a more efficient way by concentrating on the detection of undefined behaviours for a large part of the software, rather than devoting it to a full verification of a small part only.

To this end, we investigate an approach using minimal contracts only, i.e. specifying just sufficient information for the verification of absence of run-time errors (RTE). A run-time error is here meant to

be *undefined program behaviour* in the sense of the C standard.[39] Such behaviours are undesired irrespective of the program's intended functionality and include out-of-bounds memory accesses, division by zero, etc., which also can be checked by hardware at run-time. Other run-time errors include read access to uninitialized memory, (unintended) non-terminating loops, etc., which cannot.

Abstract interpretation can be used with a high degree of automation to check for RTEs. However, it usually reports false positives, i.e. warnings about undesired program behaviours that are members of the computed approximation set of possible program behaviour, but not of the (incomputable) precise set. It would be desirable, after a manual post-analysis check for true/false positives, to provide appropriate hints to the tool that help to avoid known false positives for future runs. This kind of hints is exactly what we introduced above as minimal contracts.

Minimal-contract annotations are reusable with some reasonable probability. That is, they are likely to guide Frama-C to a 100% verification rate even after minor changes in the implementation code.

Verifying minimal contracts can be a synthesis of both approaches that combines each's advantages, viz. local specifications and global analysis. A minimal contract specifies just sufficient properties of a function to verify the absence of undefined behaviours. For example, the minimal contract of the function **`int abs(int)`** that computes the absolute value might just state that the result is non-negative and that no side effects occur.

Contrary to a written report about manual post-analysis of abstract interpretation alarms, minimal contract annotations are reusable with high probability for follow-up analyses of the software. On the other hand, specifying and verifying of minimal contracts requires far less effort than more elaborate contracts for the business logic.

Using minimal contracts helps assigning the available efforts for quality assurance in a more efficient way by concentrating on run-time error detection for a large part of the software, rather than devoting it to the verification of complex properties for a small part only. In practical projects, this selectivity gains the more importance as the main emphasis is on particular aspects of software quality, such as security issues.

Within the collaborative effort of VESSEDIA task 1.1, FOKUS has annotated a critical part of a INRIA's Contiki use-case with minimal contracts (i.e. RTE requirements only) and started to evaluate the practical feasibility of this approach. This work is described in the following. We demonstrate the methodology we developed during this case-study to enable tool-support in writing minimal contracts. We discuss the issue of a basic library function needing additional requirements on its result range in order to verify a minimal contract of its caller. As a first step towards increased software robustness, we *strongly recommend* to include minimal contract information into the informal documentation (e.g. Doxygen) of each function. In the Contiki software, we actually found a run-time error during computation of a value that was, however, never used. We briefly discuss the pros and cons of ostracizing such kind of code.

6.2 Description of the software for annotation

In order to evaluate our minimal-contracts approach in a more realistic context, we performed a medium-scale annotation experiment within Task 1.1 of the VESSEDIA project. As target we chose a subset of the Contiki operating system, which is the use-case provided by INRIA and handled in VESSEDIA task 5.1.

Contiki is an Operating System for the Internet of Things. It was among the pioneers in advocating IP in the low-power wireless world. In particular, it features a 6LoWPAN stack, that is, a compressed IPv6 stack for IEEE 802.15.4 communication. This enables constrained devices to inter-operate and

[39] Since most abstract interpretation tools, as well as most deductive verification tools, are available for C, we restrict to this language. Comparable, but less versatile, tool support is available for Ada and Java. An extension of Frama-C to C++ is currently under development.

connect directly to the Internet. Sensors, actuators or consumer devices can be brought together and create applications in various areas such as home automation or the smart grid.

Contiki is targeted at constrained devices with an 8, 16 or 32-bit MCU and no MMU. The devices usually feature a low-power radio module, some sensors, a few kB RAM and tens of kB ROM. Contiki has a kernel, written in portable C, which is linked to platform-specific drivers at compile-time. It supports more than 35 different hardware platforms.

When Contiki started in 2003, the focus was on enabling communication in the most constrained devices, with no particular attention given to security. As it matured and as commercial applications arose, communication security was added at different layers, via standard protocols such as IPsec or DTLS. The security of the software itself, however, did not receive much attention. Although a continuous integration system is in place, it lacks serious quality assurance efforts and, not surprisingly, does not rely on formal verification.

Selected modules of Contiki have already been verified with Frama-C/WP.

In a VESSEDIA WP1 meeting in March 2017, the modules of Contiki were assessed with respect to their priority in the project. Two of them, **`lib`** and **`sys`**, were assessed as highly critical, and were assigned top priority. The former is concerned with memory management, lists, cryptography, etc., while the latter contains core operating system components like scheduler and timers. We concentrated our efforts on module **`lib`**, and had a glance at module **`sys`**.

6.3 Minimal-contract verification of selected files

We found that Frama-C/WP was easy to use for the *native configuration* of Contiki. However, it is important to keep in mind that Contiki has a huge amount of configuration parameters.

The main emphasis of Frama-C has been initially on analysis of embedded software. As a consequence, we had experienced problems with Frama-C support for software running on standard platforms, like e.g. Linux, where appropriately adapted versions are unavailable for less commonly-used include files. In such cases, manual adaptation of existing include files, often causing additional discussion with the software provider, had been necessary before start of analysis.

In contrast, Contiki is a fully self-contained operating system that comes with its own include files. At least in the modules we considered for analysis, no adaptation of the latter was necessary.

For the Frama-C setup, we obtained the necessary pre-processor directives from the output of the build process for a hello-world application which was recommended for tutorial purposes by the main README file of the Contiki sources. These directives are rather lengthy and hence are not shown here in detail. We also employed an external Frama-C/WP driver file registering a file `Lemmas.v` which contains some ACSL lemmas and their Coq proofs.

- For the following files, we built minimal contracts and verified them:
  - `lib/crc16.c`
  - `lib/gcr16.c`
  - `lib/ifft.c`
  - `lib/ringbuf.c`
  - `lib/ringbufindex.c`
- Besides these files we found that some files were trivial to analyse. No manual annotations were needed for them in order to make Frama-C prove the absence of run-time exceptions. These were:
  - `lib/assert.c`
  - `lib/metabs.c`
  - `lib/print-stats.c`
  - `lib/random.c`
  - `lib/settings.c`

    - `sys/arg.c`
    - `sys/energest.c`
- On the other hand, some files were intractable, i.e. couldn't be handled with Frama-C/WP due to implementation restrictions. These were, grouped by intractability reason:
  - *validity of unsized-array not implemented yet*:
    - `lib/sensors.c`
    - `sys/procinit.c`
  - ***`\valid_function`*** *not yet implemented*:
    - `lib/aes-128.c`
    - `lib/ccm-star.c`
    - `lib/trickle-timer.c`
    - `sys/process.c`
    - `sys/rtimer.c`
  - *calculus failed on strategy for XXX behaviour YYY, all properties, both assigns or not because unsupported non-natural loop(s): try [-wp-invariants] option (abort):*
    - `sys/ctimer.c`
    - `sys/etimer.c`
- Finally, the following files can be handled by Frama-C, but require additional manual annotations that we did not yet provide:
    - `lib/crc16.c`
    - `lib/gcr.c`
    - `lib/list.c`
    - `lib/me.c`
    - `lib/memb.c`
    - `lib/mmem.c`
    - `lib/petsciiconv.c`
    - `sys/autostart.c`
    - `sys/compower.c`
    - `sys/mt.c`
    - `sys/stimer.c`
    - `sys/timer.c`

For details, we refer to the technical report "Annotating IoT-Software With Minimal Contracts" (May 2017).

6.4 Discussion

In this chapter, we discuss our experiences and give some preliminary conclusions.

As a future work, it remains to be investigated if and how minimal contracts can be helpful in combining static and analyses and testing. Checks of parameter range limits can be automatically generated from a contract. On the other hand, array length information can't be observed e.g. in a running C program, but is important for fuzzing tools, and can be taken from a minimal contract.

6.4.1 A methodology to obtain minimal contracts

From our experience with the case study, we suggest the following methodology to write and verify minimal contracts.

- Provide an assigns clause for the function.
- Provide an assigns clause for each loop in its body.
    - Both clause sets just collect the memory locations that may be altered.
- Provide a variant clause for each loop, to establish its termination.
- Run "`frama-c -wp -wp-rte`" to obtain possible run-time errors.

- Repeatedly enhance the contract until that run doesn't report any unproven obligations.
- For an array access `a[i]` out of bounds:
  - Add a function requirement \valid (read/write access) or \valid_read (read-only access) about the size of a.
  - Make sure the array size is reasonable; requiring it to be greater than zero might suffice.
  - Take care of establishing the index expression i is within the array size.
    - In the frequent case that i is the running variable of a simple loop `for (i=0; i<n; ++i)`, it will usually be sufficient to add the annotation `/*@ loop invariant 0 <= i <=n */`
- In the final contracts version, check if Frama-C happens to be able to prove "for free" any additional property suggesting itself.
- Often a clause about the result value's range can be established which may be helpful later on in the minimal-contract verification of other functions.

This summarizes our typical approach in cases where no particular issues occurred. The latter are discussed separately in the following subsections.

6.4.2 Context dependency

During our experiments we experienced (unsurprisingly) that the notion of a function's minimal contract depends on the context the function is used in.

For a simple illustrating example, assume a function **`foo(int n)`** just performs some simple arithmetic on its argument **`n`**, and won't cause a run-time error if **`0 < n`** holds. That is, in a standalone context, the latter requirement is minimal to guarantee absence of run-time errors.

However, if e.g. `foo` is called inside another function, **`bar`**, and the result of the former is used as an array index, the array size induces another constraint, now on the result value of **`foo`**.

Arbitrarily complex properties may arise in this way as ensures clauses in minimal contracts.

In this sense, there is a smooth transition from minimal to full contracts. However, as our experiments show, for the overwhelming majority of functions minimal contracts are much simpler and more tractable than full ones.

6.4.3 Prover limitations

We encountered problems in verifying even simple properties about *bit operations*, in particular shifting. This is a common problem with Frama-C/WP. Such operations are rarely used in average software, and their verification support is therefore neglected to a certain extent. Our application, however, was taken from the low-level application domain of operating system kernels, and hence used these operations more often. We could circumvent most problems by devising appropriate ACSL lemmas and proving them manually with Coq, which required a considerable amount of work.

6.4.4 Visibility issues

It is well-known that–contrary to naive expectation–Frama-C cannot rely on **`const`** data fields being unaltered. However, when a field is additionally declared `static`, i.e. local to its file, Frama-C/WP could check whether the file contains code that might alter the field, and if not, rely on the initial values being kept forever. As soon as *data invariants* are supported by Frama-C/WP, they can be used express immutability of the data field.

Since the Contiki code is self-contained, we didn't have problems writing appropriate assigns clauses. However, in general a software may call routines from an external library, where the memory footprint is unknown or can only be guessed. For example, after an initialization call

`foo(&data)`, a call **`bar(&data)`** may, or may not, alter the contents of the local variable data, depending on the implementation of the external module both **`foo`** and **`bar`** belong to.

6.4.5 Tacit prerequisites

We found that some functions might cause an RTE, unless their parameters are restricted beyond what the documentation requires. For example, the array sizes of the Fast Fourier Transform function needed to be a power of two. While a look into a textbook indicated that this requirement goes without saying, we consider it good practice to state it nevertheless explicitly in the functions informal description.

6.4.6 Dispensible RTE programming

Using our minimal-contracts methodology, we found an actual underflow in the code. However, the underflown value could be shown not to be used. From a conceptional point of view, this kind of practice forces us to distinguish between

- run-time errors that actually occur and
- run-time errors that influence the program behaviour.

Note that hardware-implemented protection mechanisms don't admit this distinction: e.g. a read from a non-existing memory location will cause the MMU to throw an interrupt, even if the read value is never used.

Thus, we consider it a valid point of view to reject programs that have an RTE, even if it can't cause any damage. A warning to the programmer should be issued in such a case anyway.

This issue should be discussed among all project partners to find a consensus.

Chapter 7 Summary and Conclusion

As commonly said, the “S” in the IoT stands for security, it is either non-existent or unnoticeable. However with the increasing use of the internet connected products, this insufficient security is becoming more and more relevant. The VESSEDIA project aims to address this issue, by providing the developers with software analysis tools, which can be used to enhance security, and which also have the capability to be used during a certification process.

To achieve the goal of the VESSEDIA project, we gathered the most important security aspects of the IoT. Determining the general security requirements is a hard task, since the requirements depend on a lot of factors, like the architectural and device specific constraints or other environmental factors. During our work, we set the requirements on a system level, so the results can be applied to several use cases. Although the presented requirements do not cover every aspects of security, it gives an overview, what are the most important areas which have to be addressed during the design phase, and it can be used as a starting point during the development or security evaluation.

This document will serve multiple purposes during the VESSEDIA project. The security properties in this document will be directly examined during the evaluation of the use cases, and it also highlights potential areas, where the VESSEDIA tools can be applied during a verification process. It is also gives a general overview on the field of security, which helps for setting the general direction of further development. During our work, we will apply these conclusions and examine how these results can be incorporated into the future work of the VESSEDIA project.

Glossary

Abbreviation	Translation
Asset	Assets are entities that someone places value upon. *(from CC– ISO/IEC 15408).* Examples: • [Asset 1] Confidential data, • [Asset 2] Service or access to a service, • [Asset 3] A given functionality, ...
Attack / attacker	An actor willing to cause harm. Harm is caused deliberately, so if the possibility is there, we should assume that the attack will take place.
Attack surface	The lack of specific separations and functional controls that exist for that [attack] vector. In other words, the set of all inputs that the code uses, and which should be considered to be in the control of an attacker.
Attack tree	Multi-levelled diagrams describing either multiple steps, conditions, or kinds of a complex attacks. Attack trees are related to fault trees (used in safety analysis) since a node, in an attack tree, is considered realized when a Boolean operation is satisfied on its children nodes.
Attack Vector	Vector generally describes an interaction on an IT product. An attack vector is a single- or multi-step method by which an attacker exploits a vulnerability in an IT product.
Backdoor (also called trapdoor)	Confidential access point to functionality of an IT product. A backdoor can be created by the developers of the product or by an attacker as a part of an attack vector.
Bug	A flaw describes a vulnerability introduced at the implementation level of an IT product. Example: Vulnerability [Vul_secretprotect] would be a bug.
Computer security model / Security properties	A computer security model is a scheme which aims at enforcing a security policy. Such a model is generally based on *security properties*, which are constraints that must be enforced by the concerned system, and often expressed as logical assertions. Some examples of computer security models are Bell-LaPadula, Brewer and Nash Clark-Wilson, LBAC or RBAC (Lattice-based and Role-based access models).

Abbreviation	Translation
DSP	Digital Service Provider
ENISA	European Union Agency for Network and Information Security
Flaw	A flaw describes a vulnerability introduced at the design level of an IT product. Examples: Vulnerabilities [Vul_repeat] and [Vul_bruteforce] would be flaws.
i18n	Usual abbreviation of the word *internationalization*, denoting starting “i” and closing “n”, in between which there are 18 characters.
NVD	National Vulnerability Database
OES	Operators of Essential Services
Safety	The degree to which accidental harm is prevented, detected, and reacted to. *(from Common Concepts Underlying Safety, Security, and Survivability Engineering – SEI - CMU/SEI-2003-TN-033*) In case of safety accidental harm means that undesired events are caused by “mother nature” and not deliberately by an intelligent actor.
Security	The degree to which malicious harm is prevented, detected, and reacted to. *(from Common Concepts Underlying Safety, Security, and Survivability Engineering – SEI - CMU/SEI-2003-TN-033*) Malicious harm means that in case of dealing with security we always have to assume the presence of an attacker, who is an intelligent actor looking for possibilities of an attack.
Security objective	A security objective is the statement of an intent to counter identified threats. *(simplified, from CC– ISO/IEC 15408)* Examples: • [Threat 1] would be covered by [Objective 1]: [Asset 1] shall not be available in plain text until user authentication is successfully performed, • ... Security objectives are high-level statements, and need to be implemented through security functional requirements.
Security policy	A security policy is a consistent set of rules intended to cover security objectives. Security policies can typically be implemented as a set of security functional requirements.

Abbreviation	Translation
Security requirement	Security requirements are non-ambiguous and verifiable statements implementing security objectives. They may include: • Security assurance requirements, which lead to organizational means (rules, procedures, guidelines), or • Security functional requirements, within an IT product. The term “security requirement” will hereafter be considered a synonym to “security functional requirement”, since this study focuses on requirements that are to be implemented in a product. NB: Security requirements are commonly expected to be *architecture- and implementation-independent*: • Common Criteria has standardized several Security Functional Requirement, which are all architecture- or implementation independent • SQUARE methodology considers a mistake to “elicit implementations or architectural constraints instead of requirements” and states that “requirements are concerned with what the system should do, not how it should be done”. Examples: [Objective 1] would be covered by • [Req_auth] The TOE shall authenticate each user before allowing any action • [Req_user_secret] The TOE shall use [user secret X] to authenticate users • [Req_keygen] The TOE shall generate cryptographic keys with [algorithm X] and [keysize Y] • [Req_dataencrypt] The TOE shall encrypt/decrypt [Asset 1] with [algorithm X] and [keysize Y] ...
TCB	Trusted Computing Base
Threat	A threat is an adverse action on an asset. *(from CC– ISO/IEC 15408)* Examples: • [Threat 1] An attacker tries to disclose [Asset 1], • [Threat 2] An attacker performs a denial of service on [Asset 2], • [Threat 3] An attacker modifies the operation of [Asset 3], • ... In particular, a threat can be considered a breach to *security attributes* on assets, such *security attributes* being e.g. the “CIA triad” (Confidentiality/Integrity/Availability, see section 2.3.1
TOE	Target of Evaluation, i.e. the system that is being evaluated, in our context from security point of view.

Abbreviation	Translation
Trojan horse	A program which appears to perform a legitimate or useful functionality, while concealing malicious functions such as backdoor, data theft, keyboard logging, etc.
TSF	TOE Security Function
Virus	A virus is a program which has the capacity to spread and replicate itself when executed by a user. In order to be executed, a virus attaches itself to a legitimate executable file, a boot sector, a script or a macro (in this sense it basically uses the file system to spread). Viruses do not necessarily perform malicious actions on top of their replication functionalities.
Vulnerability	A vulnerability is a flaw or weakness in a system's design, implementation, or operation, procedure and management that could be exploited to violate the system's security policy. *(based on RFC 2828)* Consequently a threat can also be seen as a *potential threat*. Examples: • [Vul_repeat] Not having a control on the number of authentication attempts can be a vulnerability to [Req_auth], as it enables an attacker to perform a repeat attack. • [Vul_secretprotect] An implementation flaw enables an attacker to access plain text user secrets in memory space. [Vul_bruteforce] the algorithm used to generate cryptographic keys is vulnerable to brute force attack.
Worm	A worm is a program which has the capacity to spread and replicate automatically. Unlike a virus, a worm does not require an action from the user, and generally use network vulnerabilities to spread. Worms do not necessarily perform malicious actions on top of their replication functionalities.
Zero-day attack	A zero-day attack consists in exploiting a zero-day vulnerability.
Zero-day vulnerability	A yet undisclosed vulnerability, unknown to the developer and not fixed.

Chapter 8 Bibliography

[1] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. Fu, “WALNUT: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks,” in Security and Privacy (EuroS&P), 2017 IEEE European Symposium on, 2017, pp. 3–18.

[2] S. Li and L. D. Xu, Securing the Internet of Things. Syngress, 2017.

[3] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, “DolphinAttack: Inaudible Voice Commands,” arXiv:1708.09537 [cs], Aug. 2017.

[4] Y.-S. Park, Y. Son, H. Shin, D. Kim, and Y. Kim, “This Ain’t Your Dose: Sensor Spoofing Attack on Medical Infusion Pump.,” in WOOT, 2016.

[5] G. Eberhardt, Gy. Bácsi, I. Rad, A.Szász, “Evaluation Report, Security evaluation of the Compal Broadband networks CH7465LG “Mercury” Modem”, July 2016, http://www.search-lab.hu/media/Compal_CH7465LG_Evaluation_Report_1.1.pdf

[6] D. R. Raymond and S. F. Midkiff, “Denial-of-service in wireless sensor networks: Attacks and defenses,” IEEE Pervasive Computing, vol. 7, no. 1, 2008.

[7] C. Karlof and D. Wagner, “Secure routing in wireless sensor networks: attacks and countermeasures,” in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003.*, 2003, pp. 113–127.

[8] A. Mpitziopoulos and D. Gavalas, “An effective defensive node against jamming attacks in sensor networks,” Security Comm. Networks, vol. 2, no. 2, pp. 145–163, Mar. 2009.

[9] A. Narayanan and V. Shmatikov, “Robust de-anonymization of large sparse datasets,” in *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, 2008, pp. 111–125.

[10] S. Brand, “Department of defense trusted computer system evaluation criteria,” 2005.

[11] J. T. F. T. Initiative and others, “SP 800-53 Rev. 3. Recommended Security Controls for Federal Information Systems and Organizations,” 2009.

[12] B. Schneier, Attack Trees. Dr. Dobb's Journal, vol. 24, pp. 21 - 29, 1999.

[13] J. R. C. Nurse, A. Erola, I. Agrafiotis, M. Goldsmith, and S. Creese. Smart insiders: Exploring the threat from insiders using the internet-of-things. International Workshop on Secure Internet of Things 2015 (SIoT 2015), in conjunction with ESORICS’15, LNCS. Springer, 2015.

[14] Florian Kammüller, Jason R. C. Nurse Christian W. Probst, Attack Tree Analysis for Insider Threats on the IoT Using Isabelle, HAS 2016: Human Aspects of Information Security, Privacy, and Trust pp 234-246

[15] G. Sindre and A. L. Opdahl, Eliciting security requirements with misuse cases, Requirements Engineering, vol. 10, pp. 34-44, Jan 2005.

[16] Hilpinen Risto, Deontic Logic, in Goble, Lou, ed., the Blackwell Guide to Philosophical Logic. Blackwell, 2001.