Olli Huuhtanen

THE USE OF CVE-RELATED DATABASES IN IMPROVING THE CYBERSECURITY OF EMBEDDED SYSTEMS

UNIVERSITY OF JYVÄSKYLÄ
FACULTY OF INFORMATION TECHNOLOGY
2021

ABSTRACT

Huuhtanen, Olli
The use CVE-related databases in improving the cybersecurity of embedded systems
Jyväskylä: University of Jyväskylä, 2021, 69 pp.
Information Systems Science, Master's Thesis Literature Review
Supervisor(s): Costin, Andrei

Cybersecurity is an important core concept for all information systems currently being used. It is a subject that has become more relevant with each passing year as information systems become more and more prevalent in our everyday use. It is also often a very poorly understood concept by the general public, which contributes to the fact that the most severe cybersecurity threats most information system face are directly or indirectly caused by negligence by non-malicious humans, i.e. mistakes due to poorly understood practices and concepts. The subject of cybersecurity is heavily research, but there is always room for more research especially concerning more specific parts of information systems, like the embedded systems.

This study focuses on embedded systems and their cybersecurity trends through the lens of CVE -entries. Embedded systems are generally defined as small and simple, but also critical, parts of larger systems, that are responsible for certain dedicated functions, which also contributes to their specific cybersecurity vulnerabilities.

Keywords: Common Vulnerabilities and Exposures (CVE), Common Weakness Enumeration (CWE), embedded systems, hardware, software, firmware, cyber security, vulnerabilities, weaknesses

FIGURES

Figure 1 Structure of a typical embedded computing system 10
Figure 2 Common security requirements of embedded systems 14

TABLES

Table 1 All CVE -entries..... 40
Table 2 CVE -keyword search results, manually filtered 41
Table 3 Strong keywords, manually filtered 42
Table 4 CVE -keyword search, final..... 43
Table 5 Strong keywords, final 44
Table 6 Embedded systems CVE -entries per year 45
Table 7 Embedded systems CWE IDs..... 46
Table 8 All CVE -entries per year 49

TABLE OF CONTENTS

ABSTRACT 2

FIGURES 3

TABLES 3

TABLE OF CONTENTS 4

1 INTRODUCTION 6

2 EMBEDDED SYSTEMS: DEFINITIONS AND CHALLENGES .. 8
2.1 Defining embedded systems 8
2.2 Design challenges 10

3 EMBEDDED SYSTEMS SECURITY 13
3.1 Security requirements of embedded systems 13
3.2 Security weaknesses and threats 16
3.2.1 Embedded system limitations and weaknesses 16
3.2.2 Security threats for embedded systems 18
3.2.3 Security solutions for embedded systems 21

4 IDENTIFYING, CATEGORIZING AND STORING VULNERABILITY INFORMATION 23
4.1 Exposures vs vulnerabilities 23
4.2 Categorizing and identifying vulnerabilities 24
4.3 Vulnerability databases and repositories 27
4.4 Research on embedded system vulnerabilities 28

5 EMPIRICAL RESEARCH 29
5.1 Aim of the study 29
5.2 Research method and process 29
5.3 Data retrieval 31
5.3.1 Vulnerability datasets 31
5.4 Data processing 32
5.4.1 MITRE CVE -dataset 32
5.4.2 NVD CVE -dataset 33
5.4.3 CWE -dataset 34
5.4.4 Filtering software 36
5.5 Data analysis 36
5.5.1 Keywords 36
5.6 Reliability and validity 38

6 RESULTS 40
6.1 Results of the CVE -data keyword search 40
6.2 Initial notes on the analysis 44
6.3 Applying the results to CWE 46
6.4 Comparing embedded systems vs vulnerabilities overall 49

7 DISCUSSION 52
7.1 Differences and issues in the MITRE and NVD datasets 55
7.2 Limitations and future work 57

8 CONCLUSION 59

9 EXTRA REFERENCES **VIRHE. KIRJANMERKKIÄ EI OLE MÄÄRITETTY.**

REFERENCES 61

APPENDIX 1 MITRE CVE -DATASET EXAMPLE ENTRY 66

APPENDIX 2 NVD CVE -DATASET EXAMPLE ENTRY 67

APPENDIX 3 CWE -DATASET EXAMPLE ENTRY 68

1 INTRODUCTION

Technology is evolving at increasingly fast rate, and so the evolution of its security needs to keep up, or else we risk material damages, or in the worst case, loss of human life. These days the capability to process information and communicate through a network are a part of most of human made devices, and this makes the cybersecurity of these devices as important as physical security.

Cybersecurity is an increasingly important concept to consider when dealing with computers, networks, programs and data, especially since information technology is being inserted to almost every aspect of our day-to-day lives. One of the core concepts of my thesis are the embedded systems, and cybersecurity is an important factor to consider in their design since these systems often play a a critical role in larger information systems. There have been numerous studies on cybersecurity in general, and also on the security of embedded systems. Yet the field of information systems is constantly evolving, so need for new studies on its security is also there.

Embedded systems are often smaller parts of larger information systems, and are often tasked with handling mission and safety-critical roles (Papp, Ma, & Buttyan, 2015). With the rise of Internet of Things, these systems are becoming more and more common place, and this along with their important roles in larger systems created a need for efficient security solutions.

The purpose of the first part of this study is to shed light into what constitutes an embedded system/device, and what is important when considering its security. This paper will also introduce the concept of CVEs (Common Vulnerabilities and Exposures), along with the idea of vulnerability databases and the naming and scoring systems related to them to the reader, and how they are related to embedded systems and their security. This also serves the purpose of helping the reader to understand the basics of cybersecurity concerning embedded systems, and thus aids in understanding the second part of the study. The following research questions were set as a guideline for the first part of the study:

- What constitutes to the security of embedded systems?

- What are the security threats for embedded systems, and how to solve them?
- What are vulnerability naming schemes, scoring systems and databases?

This part of the study was conducted as a literature review. The information was mostly retrieved from academic sources found from online databases and search engines, for example IEEE Xplore, Science Direct and Google Scholar, concerning embedded systems, vulnerability databases, security and cybersecurity. The more technical information regarding CVEs and the vulnerability databases was retrieved from their dedicated websites, for example cve.mitre.org.

The second part of the study focused on retrieving, processing and analyzing vulnerability information concerning embedded systems in order to determine what are the most important factors to consider when discussing or designing security solutions for embedded systems, and what factors potentially make embedded systems vulnerable to security threats. The main research questions for the second part were:

- Q1: What are the current and past trends in embedded system vulnerabilities?
- Q2: Can these trends be useful in predicting and preparing for future trends in embedded systems vulnerabilities?

The research also aims to answer the following secondary research question, in order to enable future research into the same subject to be more precise and comprehensive:

- Q3: Are there any noticeable weaknesses or missing information in the CVE datasets, or inconsistencies that could be improved on?

The research part of the study was conducted by retrieving vulnerability information from three open source databases, filtering the data to only concern embedded systems, and then analyzing and categorizing the data based on what kind of vulnerabilities are present in what systems and devices, how these vulnerabilities are exploited and how do they compare to the general trends of vulnerabilities.

2 EMBEDDED SYSTEMS: DEFINITIONS AND CHALLENGES

Defining embedded systems can be difficult, since the field is constantly evolving thanks to the advances in technology and the reduction on production costs (Noergaard, 2013). But some reasonably good definitions exist, for example according to Papp et al. (2015), embedded systems generally refer to computing systems that are built into a larger system, which are designed for dedicated functions. They often consist of a combination of hardware, software and potentially mechanical parts. This combination of parts is applicable to almost any computing systems, but embedded systems are not general-purpose PCs or mainframe computers, but smaller parts of these larger systems.

Because embedded systems are often a critical part of larger information systems, their cyber security in understandably critical. Implementing efficient cyber security methods for embedded systems can also be difficult due to, for example, poor security design and implementation and the difficulty of continuous patching to plug holes in the security design (Papp et al., 2015). Poor security design especially has been a problem since security is often mistakenly taken as an addition of features to an already existing system or a device, when in reality it should be considered in the design of the system/device along with other basic concepts, like cost, performance and power (Ukil, Sen, & Koilakonda, 2011).

This first section of my literature review will be focusing on embedded systems and their cyber security as a whole, to provide a basic understanding of their functions and why their security is such an important concept. Even though the definition of these systems can be difficult to create, thanks to their great variety and constantly advancing technology, this review will list some of the most common factors between different embedded systems, in order to establish a somewhat understandable view to what they are when compared to more traditional computing systems. This section will also cover the challenges of designing embedded systems, and what are the limiting factors when considering their cybersecurity.

2.1 Defining embedded systems

The term embedded systems is not a new one; it has been used to describe engineered systems that combine physical processes with computing for some time now (Lee, 2008). But as already mentioned, creating a simple all-encompassing definition for embedded systems can be difficult. According to Noergaard (2013), there are common descriptions that apply to most of them:

- *Embedded systems have limited hardware and/or software functionality than compared to an PC.* This is less relevant in modern embedded

systems, but traditionally the limitations of hardware and software of embedded systems have been stricter than in traditional computers.

- *An embedded system is designed to perform a dedicated function.* Most embedded systems have one primary purpose, which differ from traditional computers that often have multiple functionalities.

- *An embedded system is a computer system with higher quality and readability requirements than other types of computer systems.* Since embedded systems are often critical to the functions of the systems they are part of, they typically have higher quality requirements than other parts of the system.

- *Some devices that are called embedded systems, such as PDAs or web pads, are not really embedded systems*. This relates to the fact that embedded systems are difficult to accurately define, for example the aforementioned PDAs or web pads are not embedded systems according to engineers, but marketing and sales personnel keep calling them that.

From these points it is easy to come to the same conclusion as Crnkovic & Stafford (2013), that embedded systems are the dominate type of computer systems in todays' information systems environment. Also, the term Cyber-Physical Systems (CPS) can applied to embedded systems, due to the integration of computation and physical processes (Papp et al., 2015).

When considering the architecture of embedded systems, Noergaard (2013) tells us that the embedded system architecture is an abstraction of the embedded device, i.e. a generalization of the system that is typically light details. Architecture can be used as the first blueprint when designing embedded systems, without knowing any of the internal implementation details.

The following figure by Serpanos & Voyiatzis (2013) shows us the basic structure of most embedded systems, which resembles the structure of many traditional computing systems, although with different emphasis on different parts. The figure also shows the connections which different parts have with each other, like the data bus between the systems memory and processor. The design and the challenges of each part of structure will be covered in the next chapter.

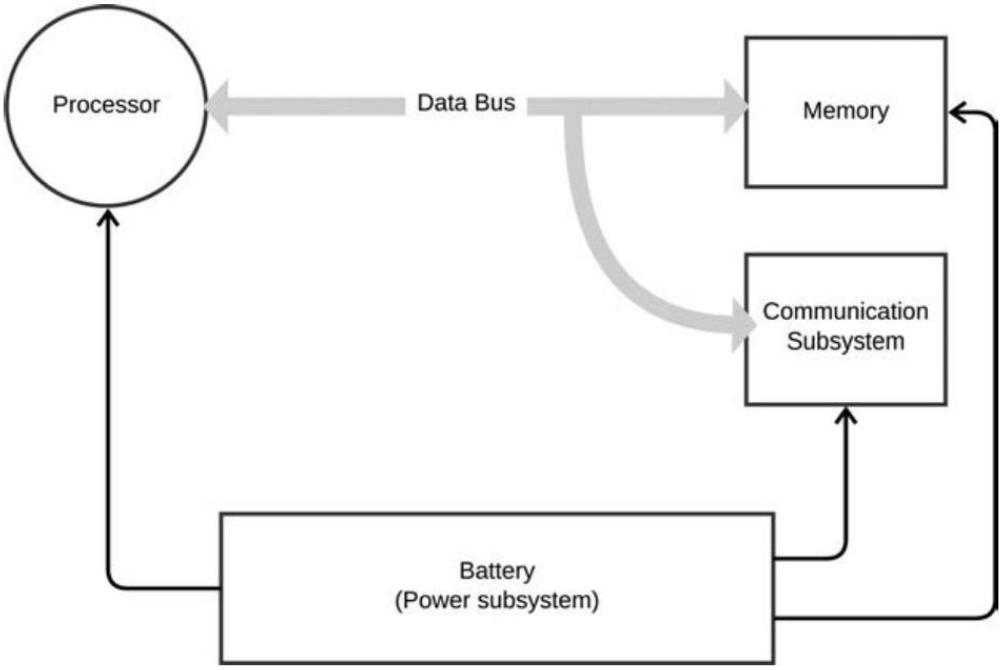

Figure 1 Structure of a typical embedded computing system

2.2 Design challenges

Embedded systems have always been held to a higher standard in terms of functioning than general-purpose computing (Lee, 2008). This naturally leads to increased requirements and design challenges for them when compared to traditional computers.

There are multiple dimensions to the problems in designing embedded systems, especially in terms of cybersecurity, but the reasons for these problems can be broadly put into to a few common categories. The first on is the **energy consumption**, which is an major concern in many embedded computing systems (C. Zhang, Vahid, & Najjar, 2003). Many embedded systems are battery-driven, such as PDAs, cell phones and networked sensors, and this creates major problems for energy consumption (Ravi, Raghunathan, Kocher, & Hattangady, 2004). This can be seen very well in the everyday use of mobile phones; prior to the invention of smart phones, cell phone batteries lasted for multiple days without recharging.

Now with smart phones, even with moderate use daily recharging is almost required. This problem with energy consumption is also referred to as the **battery gap** (Ravi et al., 2004). Ravi et al. tell us that the growth of battery capacities is relatively slow (about 5-8% a year) when compared to the growth of energy requirements of embedded systems, especially when concerning security. If this gap continues to widen, designer will need to come up with much more optimized systems in terms of energy consumption. Alternatively, new ways to provide power to embedded systems that operate wireless need to be created, such as fuel cells.

The next design problem is **processing power**, which is somewhat related to the problems with energy consumption. Since embedded systems are generally small parts of a larger whole that have dedicated functions, they do not have the processing capabilities of the more traditional computing systems. This puts limits on almost all aspects of designing embedded systems, since the limitations can close out many previously applied solutions, especially in terms of cybersecurity.

Another problem with design as pointed out by Thomas Hezinger (2008) is **predictability**. According to him, the first universal challenge in systems design is the creation of a system that's behavior can be predicted. Predictability is a problem because high-level programming models that are often used to create information systems generally don't allow for great control over the reaction and execution requirements, and instead focus on the functional requirement of the system. This makes these models unsuitable for safety-critical, real-time and resource-constrained software systems, as they put more emphasis on the reaction and execution requirements than other systems. Related to this, Hezinger also defines **robustness** as an important design challenge for embedded systems, in the sense that the reaction and execution properties change only slightly in the case that the environment changes slightly, making the software more stable. Embedded systems are often used in IoT (Internet of Things) devices across the globe, which can be located in a much more hostile environment than traditional computing system. This increases the requirements for the systems robustness, since they need to be able to withstand multitude of different environments. These kinds of environments also can be very unpredictable in terms of the changes that can happen in them, making the robustness of the embedded systems a critical factor whether they can be even deployed (Lee, 2008).

Closely related to robustness is the concept of **reliability**. When robustness relates to the resistance that the system has for change, reliability simply means how reliably the system can work even if there are no changes in the operational environment. Reliability is an obvious concern in critical systems, but is also important for noncritical embedded systems too. A failing application, like a portable video player, can erode the reputation of the manufacturing company, and lessen the user acceptance of other devices of the said company (Narayanan & Xie, 2006). Hardware reliability is also often determined by the software the system contains, as the failure of the software can result in the failure of the hardware.

These design challenges encompass all aspects of embedded systems, but are also very closely tied with their cybersecurity. In the next chapter, this paper will on into more detail on how these challenges are met currently by embedded system designers, and also what remains to be improved upon as the field in constantly evolving.

3 EMBEDDED SYSTEMS SECURITY

This chapter will focus on the current state of the embedded systems security, and how they differ from security issues affecting traditional information systems and devices.

A fundamental problem plaguing embedded systems security designers currently is that security is considered as addition of features, instead of a new dimension of design that should be considered from the start of the design process (Ravi et al., 2004). This along with the other challenges in designing embedded systems introduced in the previous chapter means that new approaches for security design are needed, as solutions that have been created for traditional computing systems often do not function directly without adapting them heavily to the embedded systems environment. This chapter will focus on the security part of embedded systems, i.e. how do the design challenges affect their security, and how to potentially solve these problems. The first part of this chapter will focus on introducing the commonly accepted security requirements for embedded systems. The second part will introduce current security problems that these requirements create, and potential solutions for them if they exist. Finally, this chapter will introduce potential future directions research on embedded systems security could take.

3.1 Security requirements of embedded systems

Embedded systems have many similar security requirements that traditional systems have, but their nature as critical parts of larger systems as well as differences in their deployment environment bring many more additional and difficult to solve requirements. The specific requirements of an embedded system also vary based on its specific operation purpose (Vai et al., 2015). Vai et al. used the CIA triad (confidentiality, integrity and availability) to define the security and resilience requirements of embedded systems:

- **Confidentiality** is used to assure that the code/data of the system remains secure from unauthorized access
- **Availability** assures that the purpose of the application is not disrupted
- **Integrity** makes sure that the application functionality is unaltered

Ravi et al. (2004) and Rosenberg (2016) list the following as the most typical security requirements that most embedded systems have:

- **User identification**, which refers to the validation of the user in order to verify is they are authorized to use the system

- **Secure network access** means that a network access is only provided if it is authorized
- **Secure communications** encompass authentication, confidentiality and integrity of communicated data, preventing repudiation of a communication transaction, and protecting the identity of communicating entities
- **Secure storage** refers to the fact that the information stored by the embedded devices needs to be secured against unauthorized access
- **Content security** refers to the usage limits of the digital content stored or accessed by the system
- **Availability** means that the system should be able to function and provide its services always when on an authorized users' request.

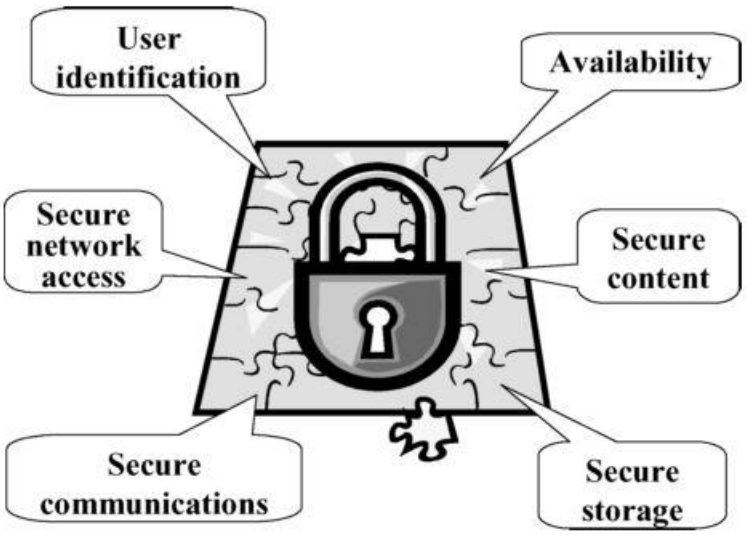

Figure 2 Common security requirements of embedded systems

Kocher et al. (2004) recognize these same security requirements as Ravi et al., but also add **tamper resistance** as one them, which refers to the capability of the device to maintain the other requirements even when it is accessed by a malicious entity, either physically or logically. Kocher et al. call all these listed requirements the **basic security functions**, which denote the set of requirements needed to ensure confidentiality, integrity and the authentication of the device, system or user.

These requirements can be applied quite extensively to traditional computing systems along with embedded systems, but generally with different focus on different aspects. Ravi et al. also point out that while these requirements are shared among most embedded systems, the security model for each embedded system dictates the actual combination of requirements that apply.

Serpanos & Voyiatzis (2013) also list several requirements they think are necessary for efficient security of embedded systems and also serve to differentiate them from traditional computing systems:

- **Deployment in significant numbers**
- **Simpler and limited resources**
- **Low cost requirements**
- **Deployment in hostile environments**
- **Strict safety requirements**

These differences mostly come from the nature of embedded systems; they are smaller parts of larger systems deployed in most parts of the world. This makes it so that embedded systems have to be able to withstand much more interference from outside sources than traditional systems, since they are often located in environments where they are easily accessible by unauthorized personnel, or the environment itself can affect them (for example, extreme changes in temperature or large amounts of rain). Some of the differences are also connected; the significant number of embedded systems naturally creates the need for the production costs to stay low so that their deployment in multitude of environments stays possible (Fournaris & Sklavos, 2014).

Cox (2016) gives us more specific examples of embedded systems requirements in the context of life critical systems. According to her, an embedded system is life or safety-critical when its failure or malfunction results in death or serious injury to people, loss or severe damage to expensive equipment, environmental harm, or large non-recoverable financial losses. The security tenets that Cox presents are divided into 7 categories: general security, communications security, boot-time security, run-time security, managing life-critical embedded systems safely, security for back-end systems and monitoring for advanced threats. These tenets cover most of the security aspects of embedded systems in a similar way as other authors presented earlier in this paper did, but Cox also adds that even if security designers were to follow her tenets to a letter, advanced threats, such as insiders, will be able to circumvent the security brought by even the best practices. She also points out that although many of the already deployed embedded devices can be difficult to update thanks to their nature, new devices with better security can be deployed in securing the old devices, thus combating some of the potential threats.

New security requirements for embedded systems are most likely required also because of the emerging utilization of system-on-chip (SoC) and network-on-chip (NoC) that are in use in some of the modern embedded systems (Elmiligi, Gebali, & Watheq El-Kharashi, 2016). This design creates a high vulnerability to hardware-based attacks for the system. Researchers have suggested multiple different potential solutions to this, like changes in architecture that would help the system recover from hardware-based attacks while still maintaining seamless operation (Kim, L W. & Villasenor, J. D. 2014). Elmiligi et al. on the other hand propose that a new classification of embedded system attacks, that take into

consideration the whole systems perspective, is required. This classification method is explained later in this literature review.

Most research categorizes the embedded system security requirements unsurprisingly in similar way, at least when considering these requirements in broad terms. It is difficult to give exact requirements for all embedded systems since they differ greatly in function and form, and exist in very different environments that require very different functions. Life-critical systems, like pointed out by Cox (2016), have generally very strict requirements in their security and function as they are critical parts of a system that is potentially responsible for human safety. On the other hand, embedded systems residing in, for example, a mobile phone generally do not require heavy security measures, quite opposite in fact, since effective security measures are often very taxing on processors and battery-life, which isn't very desirable in a phone with limited power and battery. But the general requirements that exist are similar to those security requirements that exist also in traditional computing system, though often more difficult to achieve thanks to the limited nature of embedded systems. The next chapter of this review will focus more on the problems that embedded systems have in terms of security, why these problems exist and why the requirements presented in this chapter are sometimes difficult to achieve.

3.2 Security weaknesses and threats

This chapter of the literature review will focus on the factors that affect embedded systems security, mainly how they limit it, and also what specific threats exist and how to counter them.

3.2.1 Embedded system limitations and weaknesses

As already stated in the previous chapter, embedded systems designers face significant problems in the fact that security is no longer an additional feature to be potentially considered, but instead an important core part of their design. A reason for this is that these systems are often small, but critical, parts of larger systems, and as such their failure can result in the failure of the entire system. Another significant reason for this is that their security requirements are generally higher and more difficult to accomplish thanks to their limited nature when compared to traditional computing systems. Earlier chapter of this review already discussed **limited processing power** and **battery-life** as the problems in designing embedded systems, and these problems extend as major factor in to the security limitations of these systems But, in addition to the, Ravi et al. (2004) define **the ever-increasing range of attack techniques** as and additional problems related to security. These attack techniques include for example software, physical, and side-channel attacks, and create the need for security solutions that function

even when the system is physically located in an accessible location. Kocher et al. (2004) also consider the earlier to be important limiting factors, but also make the following inclusions:

- Embedded system **architectures need to be flexible** enough to support the rabid evolution of security mechanisms and standards
- New security objectives, like denial-of-service and digital content protection, demand more **co-operation** from security personnel and system architects. Also, failure of co-operation between software engineers and system engineers can cause significant failures with safety-critical systems, which embedded systems often are (Knight, 2002).

Kocher et al. also consider the software part of an embedded system to be the major source of security vulnerabilities, and define the following three factors as the source for most of the security challenges related to it:

- **Complexity**. Software is complicated in terms of code, and is most likely only going to become even more complicated in the future. And it is only natural that when something becomes more complex, the likelihood of error only increases, thus making complex software more vulnerable to attacks. This is complicated further by the use of unsafe programming languages like C and C++ according to Kocher et al.
- **Extensibility** Modern software is built to be extended through updates and extensions to evolve the systems functionality. This creates problems in terms of security in a similar way that complexity does; more the software is extended, more likely it is that vulnerabilities will accidentally slip in.
- **Connectivity.** Both traditional computing systems and embedded systems suffer from the problems created by being in constant connection in one or more networks, but this problem is exasperated in embedded systems thanks to their weaker security measures when compared to other kinds of systems.

Processing power, battery-life and wide range of attack techniques are generally considered the major limiting factors that encompass most of the other requirements, though Serpanos & Voyaitzis (2013) list **confidentiality, integrity, authentication, access control, non-repudiation, dependability, safety** and **privacy** as the functional requirements for a secure embedded system. Three of these, safety, dependability and privacy differ from other requirements in that safety and privacy are the result of other security considerations while dependability is mainly a system issue.

One of the major challenges in terms of reliable and secure for design for embedded systems is also the environment they are deployed in. Serpanos & Voyaitzis (2013) point out that the environment for embedded systems is often

hostile, which puts significant additional requirements in their design. This is closely connected to the aforementioned, ever-increasing range of attack techniques, since the more vulnerable and publicly accessible environment opens up a significant number of different methods for attacking the system. Physical attacks are much more common in embedded systems than in traditional computing system that are often located in much more secure locations. The physical location of an embedded system can also often be remote or otherwise difficult to access, which creates additional pressure to implement security solutions that do not have to rely on human interaction to function (Parameswaran & Wolf, 2008).

Multiple sources (Costin, Zaddach, Francillon, & Balzarotti, 2014; Costin, Zarras, & Francillon, 2017; Jormakka, 2019; Zaddach & Costin, 2013) also point out that embedded system **firmware** is an especially vulnerable for embedded systems. Firmware is defined by IEEE Standard Glossary of Software Engineering terminology to be read-only memory -based software, that controls a computer between the time it is turned on and the time the primary OS takes control of the machine. Firmware is often cited to be one of the most important parts of embedded systems (Zaddach & Costin, 2013).

The factors mentioned in this chapter are generally considered to be the most limiting for embedded systems security, as most other more specific problems limited to certain sets of systems often derive from limitations in processing power, battery-life or connected nature of the systems. The wide variety of attack methods is a significant problem that will be focused on in the next part of this chapter.

3.2.2 Security threats for embedded systems

The variety of different attacks against embedded systems is one of their major weaknesses, as their limited and exposed nature enables more potential threat vectors than in traditional computing systems.

Papp et al. (2015) conducted a relatively comprehensive study on the CVE -records of embedded system security risks, and found the following to be the most common methods of attacks against embedded systems:

- **Control hijack attacks**, which diverts the normal control flow of a system to execute code injected by the attacker.
- **Reverse engineering**, where the attacker analyzes the embedded software (firmware or application) to gain access to sensitive information. Commonplace between companies trying to gain a competitive advantage (McLoughlin, 2008; Zaddach & Costin, 2013)
- **Malware**, where attackers infects the system with a malicious software that can accomplish various things, generally modifying the behavior of the infected device. Also, a very common problem in traditional computing systems.

- **Injecting crafted packets or inputs**, which are attack methods against the protocols used by embedded devices. Both methods exploit parsing vulnerabilities in protocol implementation or other programs.
- **Eavesdropping**. As opposed to packets crafting which is an active attack, eavesdropping is passive, which attacker uses to observer the messages that an embedded device sends or receives.
- **Brute-force search attack**. These kinds of attacks are often used against weak cryptographic and authentication methods, where security can be breached by simply trying enough. Useful if the search space is sufficiently small.
- **Normal use.** The attacker accesses restricted information or functions of the embedded device through using it like a normal user. This can be done if the device has no access control mechanisms.
- **Unknown**, which comprise of CVEs that described vulnerabilities that did not identify a known attack vector that could exploit these vulnerabilities.

The effects of different attacks methods can vary wildly, depending on what the embedded system is used for and in what environment. The effects that Papp et al. found were **Denial-of-service, code execution, integrity violation, information leakage, illegitimate access, financial loss, degraded level of protection**, along with some miscellaneous and unknown effects that did not have enough information about them in the CVE -records to be listed under any of the other effects.

The classification made by Papp et al. is very detailed, but only uses information collected from the CVE -databases they used. For a more general classification of different attacks, Elmiligi et al. (2016) split the different attacks under different categories, based programmability level, integration level, or life cycle phase. This is a notably different approach to what Papp et al. have done, since the attacks aren't categorized based on the methods, but on what part of the multi-dimensional representation of embedded systems, that Elmiligi et al. created, the attack targets. The first categorization they provide is based on the programmability level of the embedded system, which is divided into **Hardware (HW) attacks**, which include hijacking, data monitoring and denial-of-service attacks, physical attacks like reverse engineering of a chip or a printed circuit board, **Firmware (FW) attacks**, which include attacks against the OS kernel, and finally **Software (SW) attacks**, designed to alter the behavior of the system, consisting of Trojans and other malware or viruses. One of the more common software attacks is the buffer overflow.

The second classification made by Elmiligi et al. is based on the integration level, where the categories are **intellectual property (IP) -level attacks,** which target the various IP types in order to gain access to the system, **chip-level attacks**, which include chip cloning and changing the mode of operation based on the geographical location of the system, and **board-level attacks,** which are

categorized into three main groups: invasive, that requires physical access to reverse engineer the layout, semi-invasive, where the attacker scans the top and layers followed by converting the scanned image from pixel-format to vector-format that can be read by CAD tools, and finally non-invasive attacks, which can be either passive or active. Passive attacks have limited interactions, instead simply observe and monitor the data traffic. Active attacks on the other hand often meddle with voltage and clock signals to disable protection or force the chip to do wrong operations.

The final classification is based on the life cycle phase of the embedded system, which means that the attack targets the system based on which part of the production or consumer use the system is. The first phase is **design phase attack**, often executed by an insider. This makes it significant since most severe security threats are often created by insiders and can result in a wide variety of problems. One of the potential solutions is to implement and processor encryption so that it cannot be tampered with in the design phase of the system. The next phase is the **fabrication phase**, where attacks are often related to market competition, which means the attacker is often someone who is trying to gain a competitive advantage by copying or reverse engineering the system. Attacks in this are often reverse engineering by a rival manufacturer or designer. The final phase is the **after-production phase**, which basically means that the device is in the customers hands, and as such this phase contains the largest amount of different attack vectors against the device.

When considering the threats that embedded systems can face, it is also beneficial to understand why embedded systems are often targeted in attacks against computing systems. Mao & Wolf (2010) illustrate the following points as good examples of what makes an embedded system a tempting target:

- Retrieval of protected information, for example reading of cryptographic key material from a smart card
- Modification of stored or sensed data, like tampering with utility meter readings
- Denial-of-service attack
- Hijacking of hardware platform, e.g. reprogramming a device to a different function

The combining factors for all these examples is that they all require the attacker to gain access to the system to change its behavior or its data. These points are similar to reasons why traditional computing systems are attacked, but as the previous chapters in this review have pointed out, embedded systems suffer from more limited security measures when compared to other systems. This naturally makes them a tempting target since attacking can often be easier. Embedded systems often also exist as critical parts of larger systems, so if the purpose of the attack is to harm or debilitate a system, targeting embedded systems can achieve these results more easily than attacking other parts of the same system.

3.2.3 Security solutions for embedded systems

The problems with security in embedded systems are numerous and more difficult to solve than in traditional computing systems, and thanks to their ubiquitous nature, these problems are also important to solve.

Threats against embedded systems and their countermeasures can be considered on an individual level, for example what is an denial-of-service attack and how to defend against it, but Elmiligi et al. (2016) prefer a more general view on implementing security, as along with their categorization of threats they also present a layered approach to embedded system security by dividing it into four levels based on measures taken to enforce security, meant to be used by embedded systems designer as a guideline. This approach does not consider individual threats or specific countermeasures to them, but instead focusses on providing information about what defense mechanisms are generally good to implement relative to how critical the functions of the embedded system are. The first level, **basic security**, is the bare minimum that must be done to secure an embedded system. It requires that the designer include tamper-resistance mechanisms to the system to make the tampering of the individual parts and systems difficult. The second level is **intermediate security** which adds the detection of internal malicious behavior and protection in the fabrication phase of the system to the measures taken in the first security level. These additions can for example include access control, hardware obfuscation, watermarking or secret key activation as security measures. The third **high security** level requires designers to apply both level 1 and 2 measures, also adding more features for tamper evidence, detection and response. In case malicious behavior is detected, the responses in this level can include system shutdown or a memory wipe for example. An important note that these actions do not include the destruction of the system physically. The final security level is called **advanced security**. This level is designed to prevent the attacker from gaining any kind of access to the targeted system, allowing even the physical destruction of the system if nothing else is effective.

Earlier in the review we saw Kocher et al. (2004) and Ravi et al. (2004) list the requirements what they refer as the **basic security functions** of an embedded system. In order to ensure that these requirements are met, Kocher et al. point to three different classes of cryptographic algorithms; **symmetric ciphers** that require the sender to use a secret key to encrypt data and then transmit the encrypted data to the receiver, **secure hash algorithms**, which convert arbitrary messages into unique fixed-length values which gives the messages a unique "fingerprint". The final class of algorithms are the **asymmetric algorithms**, also called public-key algorithms, which use a pair of keys to lock and unlock data. The key used for encryption of the data is public, but only the recipient of the data has the private key that is used to decrypt it. These algorithms also require various security technologies and mechanisms in order to work as solutions. Examples of these technologies would be secure communications protocols like IPSec and SSL (commonly used in VPNs), digital certificates that help with identifying users as legitimate and digital rights management (DRM) protocols that

along with digital certificates protect devices from unauthorized use. It is also important for the devices/systems that their architecture can be tailored for security considerations.

4 IDENTIFYING, CATEGORIZING AND STORING VULNERABILITY INFORMATION

The vulnerabilities for all computing systems, traditional or embedded, are relatively numerous, and even though many of the vulnerabilities are solved by updating software and implementing new designs for the devices, new vulnerabilities are constantly appearing, creating an endless race between security designers and potential attackers. In order to help others to deal with vulnerabilities, numerous databases (open source or otherwise) have been established, often accessible from the Internet by anyone, to collect and store information about these vulnerabilities; what devices and systems do they affect, what do they cause and how to protect the device or the system from them. This chapter will focus on introducing some of the well-known databases and repositories, and methods of identifying and categorizing security vulnerabilities of information systems in general, which also include embedded systems. But before that this chapter will introduce the difference in the concepts on vulnerabilities and exposures, as the CVE -format used in the research part of this thesis separates these two terms from each other.

4.1 Exposures vs vulnerabilities

Commonly when the security of an information system or a digital device is discussed in a casual environment, the terms vulnerabilities, exposures and weaknesses are relatively interchangeable. But when concerned with a more official use of terms, the MITRE CVE separates exposures and vulnerabilities in the following way ("Terminology", 2017):

- **Vulnerability** refers to a weakness in the computational logic (code) in either software or some hardware components (firmware), that when exploited, can result in a negative effect in confidentiality, integrity or availability. An example of exploiting a vulnerability would be denial of service problems that enable an attacker to a Blue Screen of Death in the targeted system.
- **Exposure** on the other hand is a system configuration issue or a mistake, which allows a hacker to access information or capabilities that enable further access to the systems, i.e. act as stepping-stones. CVE considers issue with a system an exposure if, it doesn't directly allow the system to be compromised, but is an important component of a successful attack on a system, and is not in accordance to reasonable security policies. An example of an exposure would be the continued use of applications or services that can be successfully attacked and

breached with brute force method (e.g., use of bad encryption, small key space, etc.)

As a simplified answer to the question about the differences between vulnerabilities and exposures would be that exposures are faults in a system or a device that allows the malevolent actor to access a weakness, while a weakness is the factor that allows these actors to compromise a system or a device. But the distinction is not particularly relevant in most contexts, as according to the research done for this thesis, only MITRE CVE makes this distinction because of their naming scheme.

4.2 Categorizing and identifying vulnerabilities

In order to correctly identify security vulnerabilities and create efficient countermeasures, these vulnerabilities need to be efficiently named and categorized by a mutually agreed upon standard. This lack of interoperability can be challenge when trying to compare information from different databases (Tripathi & Singh, 2012), and as such, this review tries to focus on sources of vulnerabilities that share a common taxonomy. Next, this review will introduce some of the most commonly used standards, identification methods and toolsets used to identify and categorize vulnerabilities and their information, and which are recorded in the same format.

Common Vulnerabilities and Exposures (CVE) is a vulnerability naming scheme which functions as a dictionary for publicly known IT system vulnerabilities (P Mell & Grance, 2002). It provides the computer security community with:

- A comprehensive list of publicly known vulnerabilities
- An analysis of the authenticity of newly published vulnerabilities
- A unique identifier to be used for each vulnerability

This naming scheme is widely adopted by different organizations dealing with security vulnerabilities (Guo & Wang, 2009). The scheme was used by the MITRE Corporation to put together a list, called CVE like the naming scheme itself, which contains these vulnerabilities and exposures with corresponding identifiers for each, which enable data exchange between security products and provides a baseline index point for evaluating coverage of tools and services. The list was launched in 1999 ("About CVE", 2018), and served as one of the first attempts to standardize the categorization of security vulnerabilities and their identifiers, since up to that point different cybersecurity tools used their own databases with their own identifiers and names for security vulnerabilities.

The classification and categorization standards that MITRE's team developed for CVE's were sufficient on a preliminary level, but rough to be used to identify and categorize the functionality offered within the offerings of the code security assessment industry ("About CWE", 2018). Additional fidelity and

succinctness were required, so MITRE's solution was to create the Preliminary List of Vulnerability Examples for Researchers (PLOVER), which eventually led to the creations of the Common Weakness Enumeration (CWE), which now serves as a mechanism for describing code vulnerability assessment capabilities in terms of their coverage of the different CVEs. CWE acts as a community-developed formal list of common software weaknesses and also functions as a common language in describing them. The NVD uses CWE as a common language to discuss, find and deal with causes of software security vulnerabilities. ("NVD – Categories", 2019) Each individual CWE represents a vulnerability type, which makes it a useful tool for scoring and categorizing CVEs. As explained by Tripathi & Singh (2012), the structure of CWE can be viewed as a three-tiered approach:

- The lowest tier consists of the full list of weaknesses that CWE records, which is often used by tool vendors and in detailed research efforts
- The middle tier has descriptive affinity groupings of individual CWE -entries, most used in software security and development
- The highest tier contains groupings of the middle tier in a easily understood form, used to define strategic classes of vulnerabilities, and is widely used for high-level discourse concerning security weaknesses and vulnerabilities

While Tripathi & Singh (2012) point out that the CWE is one of the most comprehensive tools in implementing good taxonomy properties, the major problem it has is that it isn't used as the absolute standard across the entire industry, and as such makes comparing vulnerability and weakness information between sources that use it, and those that don't, a difficult task. CWE is used for this research because of its comprehensiveness, and that CWE and CVE -entries are easy to connect together as relevant CWE -entries are described on the CVE -entries.

CVE and other similar specification languages share the need to refer to IT products and platforms in a standardized way that is also usable by machines, not only by humans. This need is solved by the Common Platform Enumeration (CPE) ("About CPE", 2013). While the CVE is an naming scheme for vulnerabilities, the CPE is a structured naming scheme for IT platforms (Buttner & Ziring, 2009). In order to subject the platform for the guidance that CPE offers, three parts of the platform must be addressed; the hardware that is supporting the IT system, the operating system which controls and manages the hardware and supports application, and the application environment, such as software systems, servers and packages that are installed on the system. If these factors are specified, CPE can be used to name related vulnerabilities.

CVE and CWE are useful in identifying and describing security vulnerabilities in a common language, but what they do not provide is a standardized measure for discussing the severity and impact of these vulnerabilities. The Common Vulnerability Scoring System (CVSS) aims to solve this (S. Zhang, Caragea, & Ou, 2011). It is an scoring system used to determine the severity of security

vulnerabilities, which is an specification for documenting the major characteristics of these vulnerabilities and measuring the potential impact of exploiting them (Scarfone & Mell, 2009). The reason for its creation was to provide standardized information for organizations in order to prioritize vulnerability mitigation. It is composed of three metric groups (Peter Mell, Scarfone, & Romanosky, 2007);

- Base, which represents the intrinsic and fundamental characteristics of a vulnerability that are constant over time and user environment
- Temporal, consisting of vulnerability characteristics that change over time, but not among user environments
- Environmental, which are the characteristics of vulnerabilities that are unique to a specific users' environment

The metrics within these groups are used to calculate a score between 0 to 10 for each of the base, temporal, and environmental groups. These scores then represent how severe the vulnerability they are assigned to is. This scoring system is often used by different vulnerability bulletin providers, software vendors, organizations, vulnerability scanning and management, security management and researchers, making it a widely accepted industry standard.

Related to the standards set by the CVE, CWE and CVSS is the Security Content Automation Protocol (SCAP). SCAP is a method for using CVE set standards to enable automated vulnerability management (NIST Security Content Automation Protocol, 2018). This management includes automated vulnerability checking, technical control compliance activities and security measurements (Radack & Kuhn, 2011). It accomplishes this by, for example, automatically verifying the installation of patches, checking system configuration settings and examining systems for signs of compromise.

While CVE and its related languages and schemes are a major focus for MITRE Corporation, they also created the Open Vulnerability and Assessment Language (OVAL) as an international community effort to create and promote public security, and to standardize the way security information is communicated among different tools and services ("About OVAL", 2014). OVAL is made up of two parts that work together to promote security:

- The language, which consists of three community made schemes written XML. These schemes serve as the framework and vocabulary for the OVAL Language.
- The repositories for storing and sharing vulnerability knowledge, for example the OVAL Repository by MITRE Corporation, Altex-Soft and Cisco Systems Inc.

The control of OVAL Language and the main repository was transferred to the Center for Internet Security (CIS) in 2015 ("About OVAL", 2014).

These presented standards, schemes and classifications are created and maintained by the MITRE corporation and the National Institute of Standards

and Technology (NIST) and are in wide use among many organizations working in the IT field. These terms will also be important for the research part of my master's thesis, since most the of the data I will be using will be collected from databases related to CVEs.

4.3 Vulnerability databases and repositories

Organizations and individuals that discover and classify software and hardware vulnerabilities can keep the information to themselves, but often the information is released to the public. In that case, it needs to be stored somewhere in a form that is commonly agreed upon and can be understood by security experts and preferably computers alike. This chapter will focus on introducing some of these vulnerability databases, focusing on the ones that use the naming and classification methods introduced earlier in this chapter.

A comprehensive database about cybersecurity vulnerabilities is maintained by the MITRE Corporations ("About CVE", 2018). Already mostly explained in the previous chapter, this database contains a list of vulnerability entries, which each contain an identification number, a description and at least one public reference. These are called the CVE entries, and are used widely around the world in numerous cybersecurity products and services.

Even though both are funded by the US-CERT (United States Computer Emergency Readiness Team), the NVD (National Vulnerability Database) is a separate program from the CVE. NVD is the U.S government repository of standards based on vulnerability management data represented using the SCAP (NIST National Vulnerability Database, 2018). Originally created in 2000, it has gone through multiple different iterations, from which the current one performs analysis on CVEs that have published to the CVE Dictionary, making a connection between NVD and MITRE. This analysis results in association impact metrics (CVSS), vulnerability types (CWE) and applicability statements (CPE), and other metadata.

Another significant repository also created by the MITRE Corporation is the OVAL Repository ("About OVAL", 2014). As explained in the previous chapter, this repository is a community made database for storing security vulnerability information, which has continued to grow and develop even though MITRE Corporation surrendered the control of the language and the repository to the CIS in 2015. The OVAL Language is also used in numerous other vulnerability repositories, maintained for example by Cisco Security ("Cisco Security Advisories and Alerts", 2018).

4.4 Research on embedded system vulnerabilities

Up to this point, this study has focused on introducing the concept of embedded systems and what constitutes to their security. The next part of the thesis will focus on the research conducted on the state of security of embedded systems, and how the vulnerability data retrieved from online vulnerability databases can be used to analyze and potentially predict current and future trends in embedded system cybersecurity.

This research follows the study conducted by Papp et al. (2015) by adopting similar methods in retrieving and analyzing the vulnerability data for embedded systems, but attempts to build upon their study by increasing the number of databases the vulnerability data is retrieved from, and also improving upon the taxonomy they created by using more current and vulnerability information, and using the results of analyzing this information in order to form a clear image of current and future vulnerability trends in embedded systems.

5 EMPIRICAL RESEARCH

This chapter will focus on the empirical research conducted on the state of embedded system vulnerabilities. The aim of study, research methods and the process of conducting the study are introduced and explained in this chapter. The final part of the chapter will focus on explaining the process of analyzing the gathered and, and also how to test the reliability and validity of the study.

5.1 Aim of the study

The literature review conducted as part of this study shows that embedded systems face a wide variety of different cybersecurity threats, many of which overlap with threats that traditional information systems face, but also many that are unique to embedded systems, for example physical attacks. The second part of the conducted study aims to first and foremost answer the following research questions:

- Q1: What are the current and past trends in embedded system vulnerabilities?
- Q2: Can these trends be useful in predicting and preparing for future trends in embedded systems vulnerabilities?

The results of answering these questions with the analyzed data can hopefully help future researchers and security experts to better understand the field of embedded system vulnerabilities, and device better methods in providing solutions to these vulnerabilities. This thesis will also aim to answer the secondary question of

- Q3: Are there any noticeable weaknesses or missing information in the CVE datasets, or inconsistencies that could be improved on?

The answer to this question aims to provide guidance for future implementation of CVE -entries, so that the way the information is displayed can be standardized better, and the amount of missing information can be minimized.

5.2 Research method and process

The research method for this thesis was chosen as mostly quantitative, as the research focused on measuring and analyzing entries from open source databases concerning CVEs. This method was chosen as it was the most logical, since the used data was numerical in nature and the analysis focused on the amount of

specific data entries, and the information stored in them. The decision to use qualitative research methods was also reinforced by the fact that all other earlier research done on similar subjects is also qualitative research based on analyzing data from either from online databases, or personal data collected through experimentation. Since experimenting with embedded systems and their vulnerabilities was out of the scope of this research, as discussed earlier, gathering existing data from online databases was chosen to be the best method for the purposes of this research.

The steps taken during the research were the following:

1. Determine where and how to retrieve the research data, and how to utilize it effectively
    - CVE -datasets were determined to be relevant sources of information for embedded system security vulnerabilities, and the information is simple to analyze
2. Collect the research data
    - Information was collected from three sources; mitre.cve.org, nvd.cve.org and cwe.mitre.org. All three sources are free to access and the datasets can be downloaded in preferred format, which in this case was .XML and .CSV -files.
3. Determine how to validate and filter the retrieved data
    - Research data was filtered by searching specific keywords strongly related to embedded systems and their security. The results were then manually analyzed to ensure the validity
4. Process the datasets and filter the CVE -entries
    - A small program was developed to go through the datasets, and print out all the CVE -entries where a defined keyword was found in the description or the summary of the CVE
    - The program was run on the NVD, MITRE and CWE -datasets, using long keyword lists and then individual keywords to further filter the results.
    - Once the amount of filtered CVE -entries was small enough, the results were validated manually.
5. Log the results from the filtering process.
    - The results were logged separately from each dataset and were visualized for easier interpretation.
6. Analyze the results and form deductions on trends on threats to embedded systems
    - First determine whether the amount of embedded systems related entries have increased or decreased over the years in comparison to all CVE -entries
    - Determine which types of weaknesses these entries have, and vulnerability exploitation methods are the most common

- Based on this information, drawn conclusions on the current landscape of embedded systems cybersecurity, and what should be improved in the future
- Finally, explain the weaknesses in this research, and how future research and avoid them.

5.3 Data retrieval

This section of the thesis will focus on explaining the process retrieving the datasets used for this research, and also the reasoning behind the decision to use these specific datasets over other ones available.

5.3.1 Vulnerability datasets

The research data was retrieved from three different repositories: The MITRE CVE -list from cve.mitre.org, NVD -list from nvd.nist.gov and the CWE list from cwe.mitre.org. The MITRE and the NVD lists use the same Common Vulnerabilities and Exposures naming scheme to store the vulnerability information, and the CWE -list is used to help categorize the information gathered from the MITRE and NVD into a standardized form.

The dataset retrieved from cve.mitre.org was the full CVE -list available for download on their site, contained around 140 000 CVE -entries from 1999 to 2018. whit the caveat that the dataset contains duplicate, reserved and rejected entries, thus making the actual number of unique CVE entries smaller. For this research, only the non-duplicate and valid entries from 2010 to 2018 are considered in the evaluation of the trends in embedded system vulnerabilities, since data from before 2010 was considered too old to be particularly relevant in analyzing the current situation or predicting future trends, as trends in vulnerability threats change quickly. The number to entries in this 2010 to 2018 range is around 97 000. The NVE datasets were divided on a yearly basis already on the NVD database, so retrieving the information only concerning the years between 2010 and 2018 was easier. The combined amount of CVE entries in these NVD datasets was around 76 770 entries, most of which were from 2014 to 2018. Finally, CWE dataset contained entries from 1995 to 2018, amounting to around 900 individual weakness IDs, which was a significantly smaller amount than either in the MITRE or the NVD datasets, the reason being that unlike a CVE -entry which refers to a specific vulnerability in a specific device or a system, a CWE -entry refers to a weakness that can apply to a multitude of different CVEs, and as such is used to categorize the CVEs gathered in this research under specific categories of weaknesses. The CWE dataset will not be included in the data processing phase when searching for vulnerabilities relating to embedded systems, but it will be useful in the next analysis phase when categorizing these vulnerabilities.

These three datasets were chosen mostly for the reasons of easy accessibility and the large amount of vulnerability data, as well as the prevalence of CVE -based method of storing vulnerability data. Accessing the datasets was done by simply downloading the most recent complete list of vulnerabilities from each of the respective websites, where information is stored in multiple lists organized by year or a specific point of view. The CVE -format also functions as a commonly used industry standard for vulnerability and exposure identifiers and contains a particularly large amount of them, which also acted as a strong incentive in choosing these specific datasets. The CWE -list was chosen as it is the intended tool to be used in categorizing the CVE information from the MITRE and NVD -lists, already in use, for example, by the NVD, which also makes it a logical choice for this research as well.

5.4 Data processing

The first important part of the research was to determine whether the retrieved datasets contained relevant information for the research, and criteria to used in filtering the data to only concern embedded systems, as the datasets contained vulnerability information of a wide variety of devices and systems that were not the focus of this research. This section of the research will first explain the format in which the vulnerability information is stored in the datasets, and then what parts of the information for each of the vulnerability entries were used in the determining whether the particular entry was relevant for the research or not. I will also shortly introduce the functions of the small program used to filter the data and print the results in easily readable form.

5.4.1 MITRE CVE -dataset

The first and the most comprehensive dataset used in this research was the CVE list created by MITRE. As mentioned earlier, the complete list contained around 140 000 CVE -entries from 1999 to 2018, stored in a single XML -document. This section will explain what information each of the individual sections of a single CVE -entry present, and which of these sections were considered relevant for the purposes of determining whether the entry referred to an embedded system or not, and why this is the case. The information for this section was mostly retrieved from the official MITRE CVE FAQ page ("Frequently Asked Questions", 2019). See Appendix 1 for the visual representation of the CVE -entry in question, viewed in Notepad++.

The information in each of the sections in the example CVE -entry is divided in the following way:

- **<item>** section contains the entire CVE entry, and the headline displays the CVE ID, which shows the CVE prefix, year and sequence

of the CVE entry in question. In the above example, the year of the entry is 2017 and the sequence is 9231.

- **<status>** shows the status of the current entry. The above example has the status of CANDIDATE, which means it is an accepted CVE entry. Other possibilities are
    - RESERVED, which means an organization, or an individual has reserved the specific ID for an CVE, but they haven't published the information yet. RERSERVED entries can remain as such even if the information is never provided
    - DISPUTED means another party has disputed the specific entry in question as a legitimate vulnerability. Important thing to note here is that the CVE makes no effort the determine whether the disputed entry is a legitimate vulnerability or not, and instead encourages the reader the research it on their own.
    - REJECT, which refers to an entry that is not accepted as a CVE, for example because it is a duplicate or has been withdrawn by the original submitter
- **<desc>** contains a short description of the CVE entry in question, for example what devices and systems the vulnerability concerns, and what does the vulnerability allow the attacker to accomplish
- **<refs>** displays the sources which provided the information for the creation of the CVE entry

Rest of the fields shown in the example (phase, votes and comments) are abandoned, and as such won't be used in this research then determining whether are a specific CVE entry is related to embedded systems or not.

The important parts of an individual CVE -entry for the purposes of this research were the <item> and the <desc> fields, as these two provide the name and the ID of a particular CVE, and the actual description of the vulnerability the entry referred to. The keyword-based filtering that was used on the datasets also utilized the <desc> field for finding matches for the chosen keyword(s), for example the CVE in the picture above would give a match for the keyword "mobile", which was one of the important keywords used when filtering the datasets.

5.4.2 NVD CVE -dataset

The NVD CVE -dataset was the second large source of CVE -information used for this research. Unlike the dataset retrieved from cve.mitre.org, the NVD separates their CVE -entries in different datasets based on the year when the CVE was added. This made retrieving data only from the years between 2010 and 2018 easier. As in the earlier chapter with the MITRE -dataset, this chapter will give an example CVE -entry from the NVD -dataset and explain the information each of the sections in the entry contain, and which of these sections are relevant to this research and why. For a visual representation of the example CVE -entry viewed in Notepad++, see Appendix 2

The CVE -entries in the NVD -dataset contain significantly more information per entry than the entries in the MITRE -dataset. This does make the NVD -dataset much more comprehensive and a better tool when trying to gather information about software vulnerabilities. The following part of this chapter showcases the important sections of individual CVE -entries in the NVD -dataset for the purposes of this research, and provides a short description about what information they contain:

- **<entry id>** shows the year in which the CVE was added to the database, and also the ID number of the CVE (for example, "CVE-2017-002")
- **<vuln:product>** contains information about the product the CVE concerns (for example, Microsoft Edge)
- **<vuln:published/last-modified-list>** shows the date when the vulnerability was published and the last time the information was modified
- **<vuln:cvss>** shows the CVSS score for the specific CVE entry, along with other CVSS information. A more comprehensive explanation of the CVSS can be found in an earlier chapter of this study
- **<vuln:cwe>** contains the id the CWE which is related to the CVE entry in question.
- **<vuln:references>** shows the sources where the CVE information was gathered
- **<vuln:summary>** has the written explanation and summary of the CVE entry, i.e. what is allows the attacker to do and how

From these sections, <entry id> and <vuln:summary> are used by the filtering software when searching for keywords in the dataset. <vuln:cwe> is also very useful for the purposes of this research, since it directly offers the link between the CVE in question and the software weakness that is related to it, which will be make the process of determining the trends in security vulnerabilities much easier.

5.4.3 CWE -dataset

The CWE -dataset is different from the MITRE and NVD -datasets, since it doesn't list CVE information, but instead lists the current known CWE -entries. As such the data will not be processed with the filtering software, as searching for keywords related to embedded systems would be pointless, but the dataset will instead be useful in the next phase of the study, data analysis, in which the CVE -entries related to embedded systems will be categorized.

This chapter will showcase an example entry from the CWE -dataset and gives a short explanation for each of the sections in the example entry. For a visual representation of the example entry viewed in Notepad++, see Appendix 3:

- **<Weakness>** contains information about the ID, name, abstraction, structure and status of the CWE -entry in question (for example, ID=”1004” Name=”Sensitive Cookie Without ‘HttpOnly’ Flag”)
- **<Description>** gives a short description of the weakness the CWE -entry is related to; how the weakness manifests itself, and what are its potential consequences
- **<Extended_Description>** provides additional information about the weakness
- **<Related_Weaknesses>** names other CWE -entries that are related to the entry in question
- **<Weakness_Ordinality>** mentions the ordinality of the current entry in relation to other CWE -entries
- **<Applicable_Platforms>** mentions the languages, paradigms, operating systems, architectures and technologies the weakness is applicable to
- **<Alternate_Terms>** describes an alternate term to use for the CWE
- **<Modes_Of_Introduction>** provide information about how or when a weakness can be created
- **<Likelihood_Of_Exploit>** determines how likely it is that the weakness will be exploited, compared to other weaknesses
- **<Common_Consequences>** specifies different individual consequences that can be associated with the weakness
- **<Detection_Methods>** describes how to detect the weakness in question in a vulnerable system or device
- **<Potential_Mitigations>** lists potential mitigation methods against the weakness in different phases
- **<Demonstrative_Examples>** shows examples of how exploiting can look like
- **<Observed_Examples>** shows real-life examples of the weakness being exploited, referencing specific CVE -entries related to the weakness
- **<Related_Attack_Patterns>** shows IDs for other, similar attacks
- **<References>** outsides sources for the CWE -entries information
- **<Content_History>** the date for when the CWE was originally submitted, and the dates of potential modification to the CWE information

As can be seen from the above list, each of the CWE -entries in the dataset contain a comprehensive set of information about all the aspects of the CWE. In terms of this research, the most important sections in an individual CWE entry are <Weakness> and <Description> as they offer a short explanation of the basic nature of the CWE entry. Also, <Observed_Examples> is of significant importance, as it offers a direct reference to CVEs that are related to the CWE entry.

5.4.4 Filtering software

The total amount of CVE -entries in the datasets used in this research is around 200 000, and the amount of unique entries is around 140 000. This coupled with the fact that the CVE -entries contain a relatively large amount of information, makes it so that analyzing each of the entries manually would not be efficient in terms of time and resources.

This is why a lightweight filtering software was created to limit the amount of entries that have to be manually analyzed. How the software functions, is relatively simple:

- The user first places the datasets into their respective folders where the programs can read them
- Next the user creates a text file containing a list of the keywords used to filter the data
- The user then runs the program, defines the text file containing the keywords, and names the file the program prints the results in
- After the initial filtering, the program offers the user the option to do further filtering with a new set of keywords, or the option to terminate the program

The program outputs a text file which contains the results of the search for each of the datasets included. The information displayed in these files is limited, as it is mostly used as a reference point when determining which CVEs are related to embedded systems and which are not; the full description of each of the relevant CVE -entries is still best retrieved in the original dataset.

5.5 Data analysis

This chapter will focus on explaining the process of filtering and analyzing the vulnerability data in detail.

5.5.1 Keywords

After the datasets used in this research were chosen and retrieved, which was explained in detail earlier, the next part of the research was to choose the keywords used for filtering the data. As the datasets were too large to go through manually, it was determined that the best way to limit the data to only concern embedded systems was to create a whitelist of specific keywords which are mostly used in the context of embedded systems. The keywords were chosen based on three criteria determined to be the best for the purposes of this research:

- **Application**. Embedded systems are often a small part of a larger whole and fulfil a certain specific purpose. These keywords where chosen based on what are common uses for embedded systems, for example communication.
- **Manufacturers.** These keywords are names of known embedded system manufacturers, for example Cisco
- **Devices.** Keywords chosen based on what hardware often use embedded systems as part of its functionality, for example modems.

Though the keyword list significantly reduces the amount of CVE -entries, it is still important to note that since the functionality of the software is relatively limited, manual processing of the CVE -entries is still required to make sure the data is actually valid. An example of a keyword would be "microprocessor", which refers to a small processing unit of limited capabilities, mostly used in embedded systems thanks to its compact nature and small resource requirements. But since microprocessor can be involved in non-embedded system related issues, further manual filtering, based on the description of the CVE, of the results is still required. For a full list of keywords used in this research, see Appendix 5.

The total amount of keywords used in this research was 71, though this included a noticeable number of "weak" keywords which could easily be connected to information systems in general, instead of just embedded systems. For example, the word "micro" might often appear in CVE -entries concerning embedded systems, but is general enough to also appear in numerous entries that are not related. This is why a further filtering the lists created with these general keywords was done with more specific keywords that are harder to apply to information systems in general, for example "embedded" or the afore mentioned "microprocessor".

The list of these strong keywords was based on other research on embedded systems, and what words commonly appear in them. Some of them were relatively easy to determine, for example "embedded" is easy to define for obvious reasons. The rest of the strong keywords were chosen to be the following:

- **cyber-physical systems (CPS)** is heavily related to embedded systems, and as such considered an strong keyword. They are a integration of computation and physical processes (Humayed, Lin, Li, & Luo, 2017) (Lee, 2008) and as such a part of most embedded systems.
- **microcontroller/microprocessor/microkernel/microserver** refer to small parts of computing systems, often used in embedded systems thanks to their small size and power requirements
- **physical**, as embedded systems can often be situated in environments which enable physical threats to be relevant (Fournaris & Sklavos, 2014).
- **hardware** as embedded systems often have a tight integration of hardware and software (Gürgens, Rudolph, Maña, & Nadjm-Tehrani, 2010).

- **mobile** as embedded systems are heavily used in mobile devices thanks to their small size.
- **firmware** and its vulnerabilities are very relevant in the field security, as it is the key part connecting hardware and software in embedded systems (Hou, Li, & Chang, 2017). Firmware is also categorized as one of the most vulnerable parts of embedded systems by other similar research (Costin et al., 2014; Costin, Zarras, & Francillon, 2017)
- **real-time** (including real-time operating system, RTOS). Real-time processing often exists in cyber-physical systems, which was chosen as a strong keyword (Lee, 2008)

These strong keywords were used as is to filter the entire datasets, but also to further filter the data from search results from weaker keywords which produced too many results to go through manually. Nevertheless, even the results produced with the strong keywords were manually reviewed before accepting as valid, as that was deemed to be the most reliable way to produce accurate results.

5.6 Reliability and validity

In order to show that the results of this research are trustworthy, the reliability and the validity of the study must be confirmed, which is what this section of the thesis focuses on. Reliability measures whether the results of the study are credible and can be replicated, while validity refers to whether the testing methods actually measure what they are supposed to.

A significant factor of the reliability and validity of this research comes from the CVE -data that is analyzed for the results, as well as the tool used to filter and analyze this data. Reliability is mostly determined by the processes used to validate the CVE data as it is reported and stored on the databases. CVEs are often submitted by verified vendors concerning their own products, which increases the reliability of the submitted data significantly, as organizations have a high incentive to provide accurate information for the CVE -entry to make it as accurate and comparable to other CVEs as possible. Otherwise, it is very difficult to increase the reliability of the data through experimentation during this research, as the requirement would be to try replicate the conditions in CVE -entries and try to exploit the vulnerabilities. This is not feasible for multiple reasons:

1. There are a very large number of CVE -entries, even if they are filtered to only concern embedded systems.
2. Time and resource requirements to try to exploit many of the vulnerabilities are out of the scope of this research
3. The technical knowledge to exploit many of the vulnerabilities

As such the most logical and feasible course of action is to trust the information stored in CVE -entries as reliable, since proving otherwise would be a difficult

task. As such using multiple sources for the CVE -data is recommended to confirm the validity of the information, as was done in this research by acquiring data from both the MITRE and the NVD websites, as well as using the website cvedetails.com to help categorize the data correctly.

The second factor to the reliability is the tool used to filter the CVE -entries so that only entries concerning embedded systems are used in the research. The tool itself is simple; it only filters and prints results based on the inputted parameters, so taking the results directly from the output would not be reliable. This is why in order to increase the reliability of the results, the tool is only used to filter the most obvious entries, and once the number of individual entries has been narrowed down significantly from the original approximate 140 000, the remaining entries can be confirmed to be relevant manually. The reliability can also be improved by comparing the results to similar studies concerning CVE -entries.

Validity of the research is also closely related to the usage of the filtering tool. Since the validity of research greatly depends on whether the processed data concerns only embedded systems or information system vulnerabilities in general, the software and its correct usage is the primary tool ensure that the data is valid. As mentioned earlier, the tool isn't very "smart", and as such should not be trusted to provide completely valid data without oversight. As such the validity of the results must be confirmed manually, by looking through each of the entries logged during the automated keyword search done by the software. Also, comparing the final results to other similar analysis done on CVE -data, we can see how the results vary between them and this research, and then make assumptions about the accuracy of the results of this research. Recent analysis similar to this research on embedded systems CVE -data was not found after the 2015 analysis done by Papp et al., so their research will be used as the primary comparison point for the results of this research. Other, not embedded systems related, CVE -analysis was easier to find, for example the research on vulnerability classification by Kuhn et al. (2017) and Neuhaus & Zimmermann (2010). These are also used as reference in the research section of this thesis, when comparing CVE -entries in general to embedded systems -specific ones.

6 RESULTS

This chapter will discuss the results of the data processing and analysis of the CVE data related to embedded systems conducted during the research, which includes the most common results of the keyword -search, and how these results can be classified with the help CWE.

6.1 Results of the CVE -data keyword search

The list of keywords user for filtering the data from the CVE -datasets contained around 71 keywords based on the criteria explained earlier in this research. Before the keywords -based filtering was started, the CVE -entries between 1999 and 2009 were removed. This resulted in around 97 500 entries remaining in the datasets. After this the datasets were filtered with each keyword individually, and the end result after the initial filtering yielded around 33 000 CVE -entries from both the MITRE and NVD datasets. As mentioned, these entries were from between the years 2010 and 2018, as older entries were deemed to be not relevant to the current environment of embedded system security, as many of the vulnerabilities before are most likely either solved or appear in a new CVE -entry at a more recent year. Also not included are incomplete and disputed entries, as they do not provide useful information for the purposes of this study due to not being reliable. The following graph visualizes the initial results.

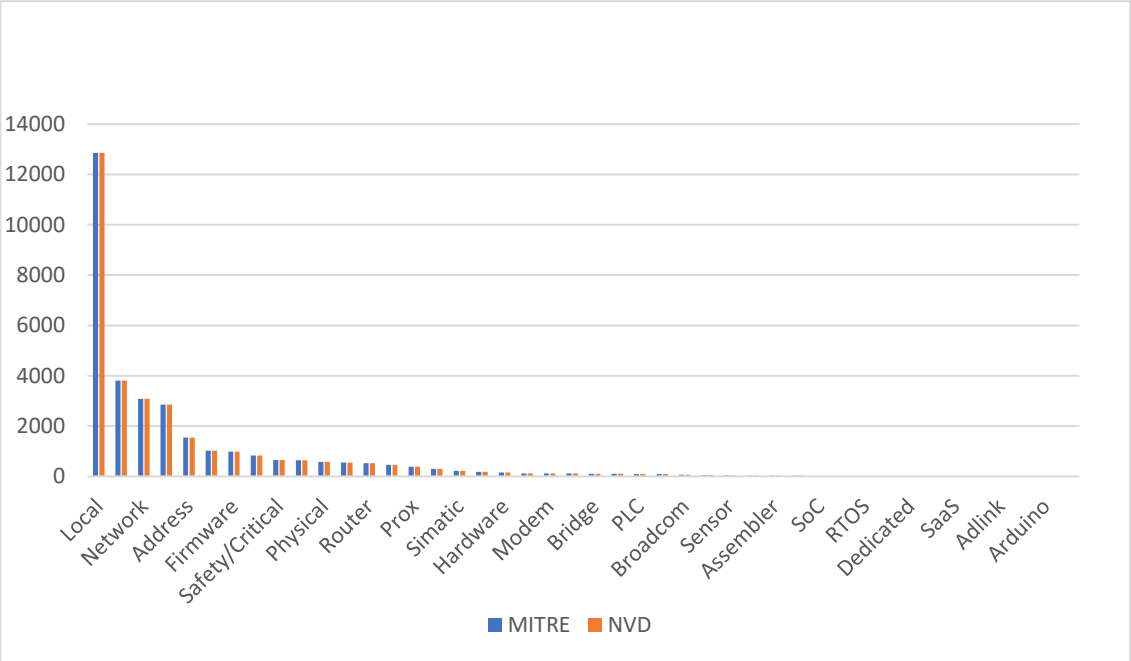

Table 1 All CVE -entries

Any analysis from the initial results would be heavily skewed, as the keyword **local** has significantly more results than any other keyword (12 847 entries), showing how unreliable it is as a keyword in the context of this research.

After the initial filtering of the results, the remaining entries were first reviewed manually to remove a large amount of RESERVED and DISPUTED -entries from the filtered data, along with large clusters of similar entries referring to the same vulnerability so that only one of entries remained from each cluster. After this the data was reduced enough for a more thorough manual validation to remove the rest of the non-relevant entries that the software missed. All of this lowered the amount of entries to 7650, which are visualized in the following graph.

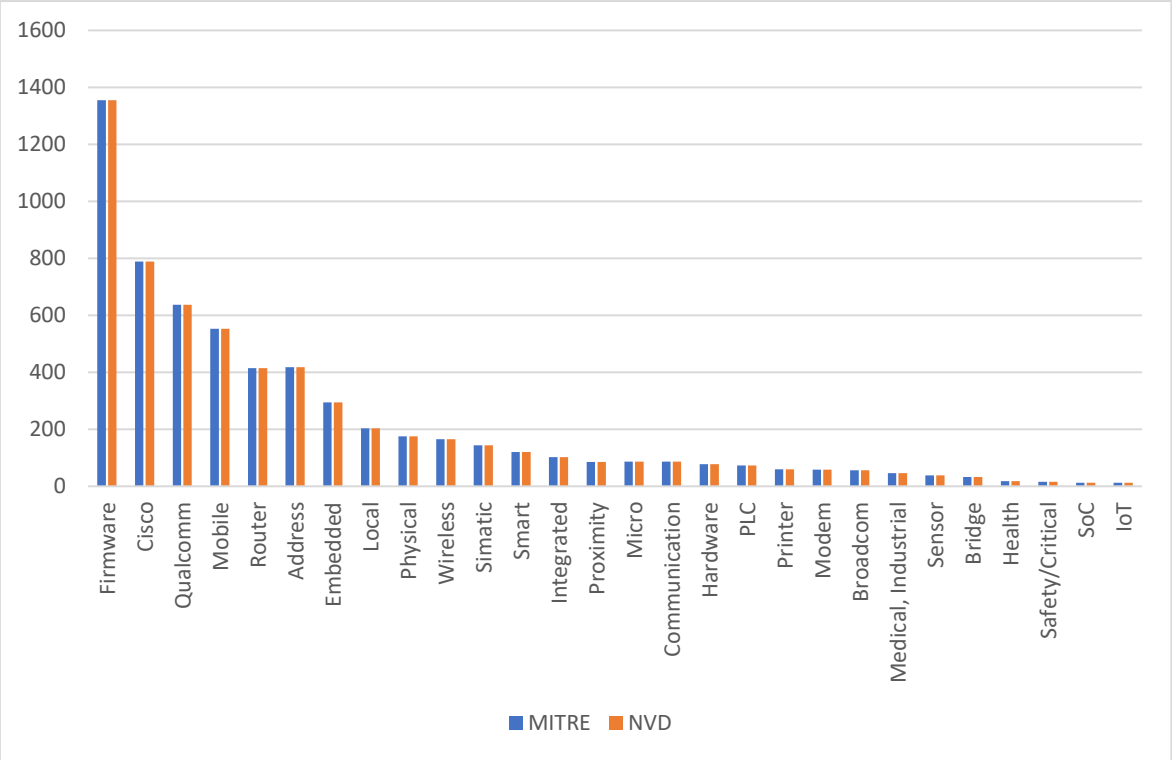

Table 2 CVE -keyword search results, manually filtered

The other keywords gave less than 10 hits during the filtering process, and as such were deemed not significant enough to include in the analysis, and hence this graph.

At this point a large amount of CVE -entries from the research material had been removed as non-relevant, as around 75% of the CVE -entries had been filtered out. To help with the analysis, searched based on the strong keywords defined earlier in this research were separated on to their own graph, with the following results.

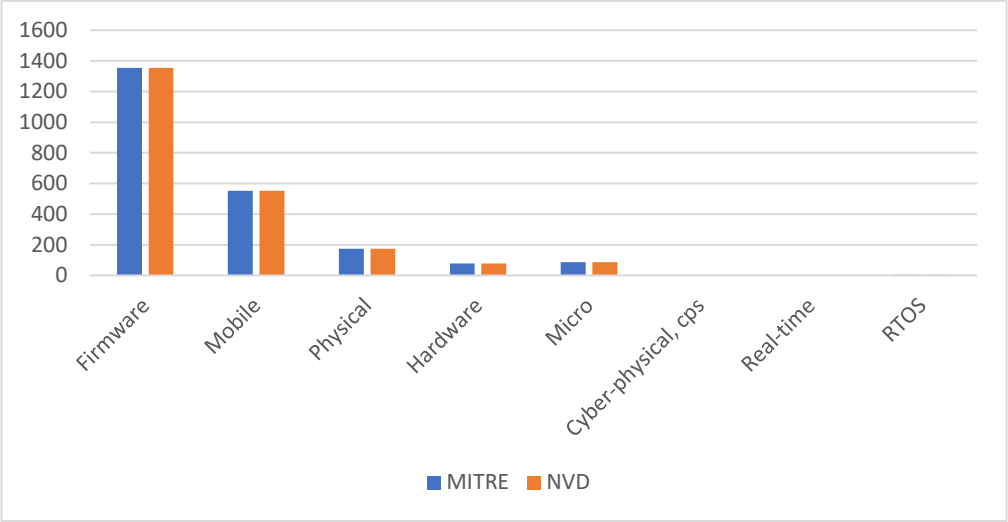

Table 3 Strong keywords, manually filtered

The top 3 results (firmware, mobile and physical) noticeably provided the most hits, and even after the manual filtering process very few entries were filtered out, which reinforces the assumptions that they are heavily connected to embedded system vulnerabilities. The rest of the strong keywords provided less results, meaning either vulnerabilities relevant to them are less common, or the strength of the keywords was overestimated in this research.

The final step in filtering the search results was to removed duplicate entries from the filtered data. Since the filtering software combed through the entire data individually for each of the keywords, many of the more prevalent keywords included the same entries under multiple search results. After removing the duplicates, the final number of relevant CVE -entries was 5139, visualized below.

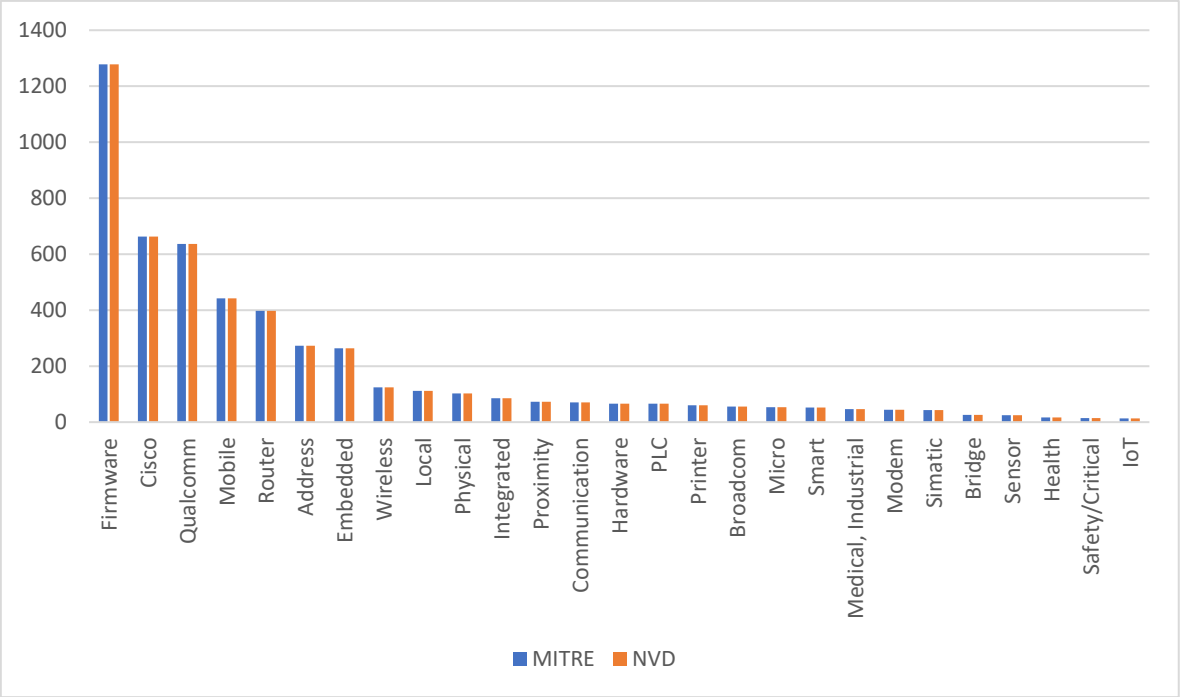

Table 4 CVE -keyword search, final

The results remained mostly the same, with a noticeable drop in **Cisco** as a number of these entries were included in other keywords too. As Papp et al. (2015) pointed out, there is a heavy representation in CVEs towards a small number of embedded system manufacturers, of which Cisco is has the most amount of CVE -entries dedicated to them. As can be seen from the above graph, this trend has continued since 2015. At this point the overall amount of CVE entries remaining in the data was 5139.

Removing the duplicate entries effect on the strong keyword search results can be seen in in more detail on the following graph.

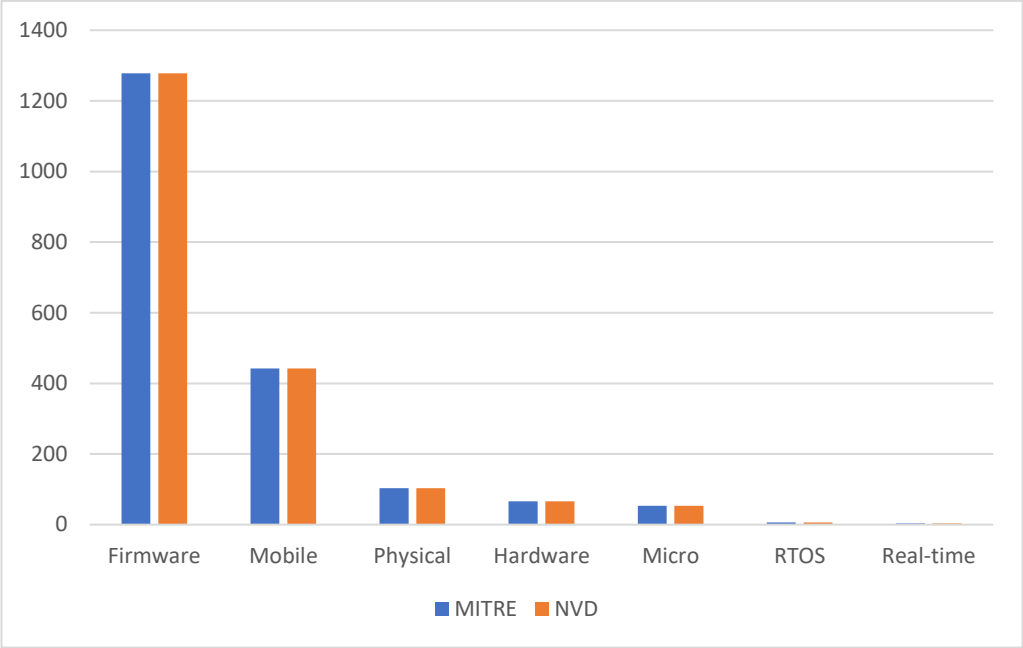

Table 5 Strong keywords, final

The overall amount of the entries naturally dropped, but the drop was relatively equal between different keywords, and the same top 3 of firmware, mobile and physical stayed in the same order, firmware still being the strongest keyword overall, providing the most results.

The next part of this chapter focuses on analyzing these results and drawing conclusions on the vulnerability trends in embedded systems.

6.2 Initial notes on the analysis

Overall, the result of the data filtering was that around 15.60% of all CVE -entries from the start of 2010 to the end of 2018, which amounted to 5139 individual entries. Comparing this to the 3826 entries that Papp et al. (2015) found to be embedded system related on their research, this amount has increased around 1200 entries between 2016 and 2018, though it is important to note, their analysis included entries from 2005 onward, which means that the exact amount of increase in entries is lower. This makes a key point that could be improved in future research, to make the analysis more comprehensive.

From the keywords used to filter these entries, the following notes can be made based on the number of hits:

- Initially **local** was the most prevalent keyword, with over 12 000 hits, but as the data was further filtered most of these were removed as duplicates that also appeared under other keywords, or were otherwise non-relevant or incomplete entries, leaving **firmware** as the most

prevalent keyword in CVE -entries concerning embedded systems, with 1278 hits.

- **Cisco** and **Qualcomm** produced a large part of the results, both being brands that produce hardware heavily tied to embedded systems.
- After **wireless**, the drop in keyword hits becomes more prominent, each keyword providing less than a hundred hits from the datasets.
- **Embedded** appears to be among the top 10 keywords, but its usefulness is significantly reduced as most of the CVE -entries do not have an CWE id defined. The issue with this is explained in the next chapter.

We can also make comparisons based on the year of the CVE -entries. The following graph visualizes the number per year.

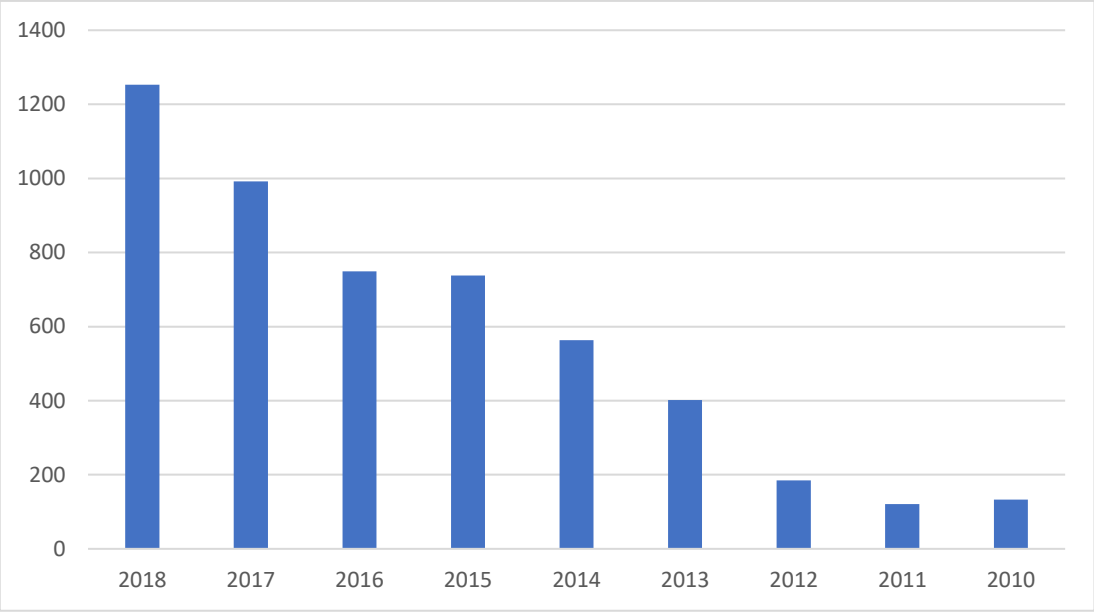

Table 6 Embedded systems CVE -entries per year

We can see that the amount of CVE -entries has been steadily increasing from 2012 to 2018, with large increases in the last few years. In terms of the keywords used in this research, the increase in from 2016 to 2017 is very noticeable in firmware (from 140 to 302) and mobile (from 66 to 144). Since the use of embedded systems increases every year as they are ubiquitous in information systems and hardware firmware is an integral part of embedded systems as pointed out by Hou et al. (2017), the increase is understandable. The applies same applies to mobile, as the amount of mobile devices steadily increases every year (Hintze, Hintze, Findling, & Mayrhofer, 2017).

While most keywords saw an increase in the amount on hits every year, Qualcomm saw a sharp decline in from 2017 to 2018 (from 104 to 10) similarly to Cisco (from 99 to 73). This is though mostly explained by the removal of duplicate hits from the final research data. Both Qualcomm and Cisco were heavily

featured in the analyzed CVE entries as they are large manufacturers of device that user embedded systems, but most of their CVE -entries also included other keywords, so they were chosen to be categorized under them since it is more descriptive than to only refer to the manufacturer.

6.3 Applying the results to CWE

In the previous chapter we discussed the results of the data analysis in terms of what how embedded system vulnerabilities have progressed from 2010 to 2018, and how this appears in the keywords used in the analysis. Now we will apply those results to CWE -entries to determine the most common vulnerability categories the embedded system vulnerabilities can classed under and see how the vulnerabilities are can be exploited.

Attaching CVE -entries to the correct CWEs was done by searching the entries on cvedetails.com, where most stored CVE -entries also contain the related CWE ID. The entries that had incomplete information on cvedetails.com were analyzed from NVD nvd.gist.gov, as the NVD CVE -entries also contain the CWE ID listed on them. The reason why all the entries were not searched from the official NVD site was that the search engine is not as good, making retrieving the information more time consuming. All of the 5139 CVE -entries were attached to the corresponding CWE ID, with the top results visualized below.

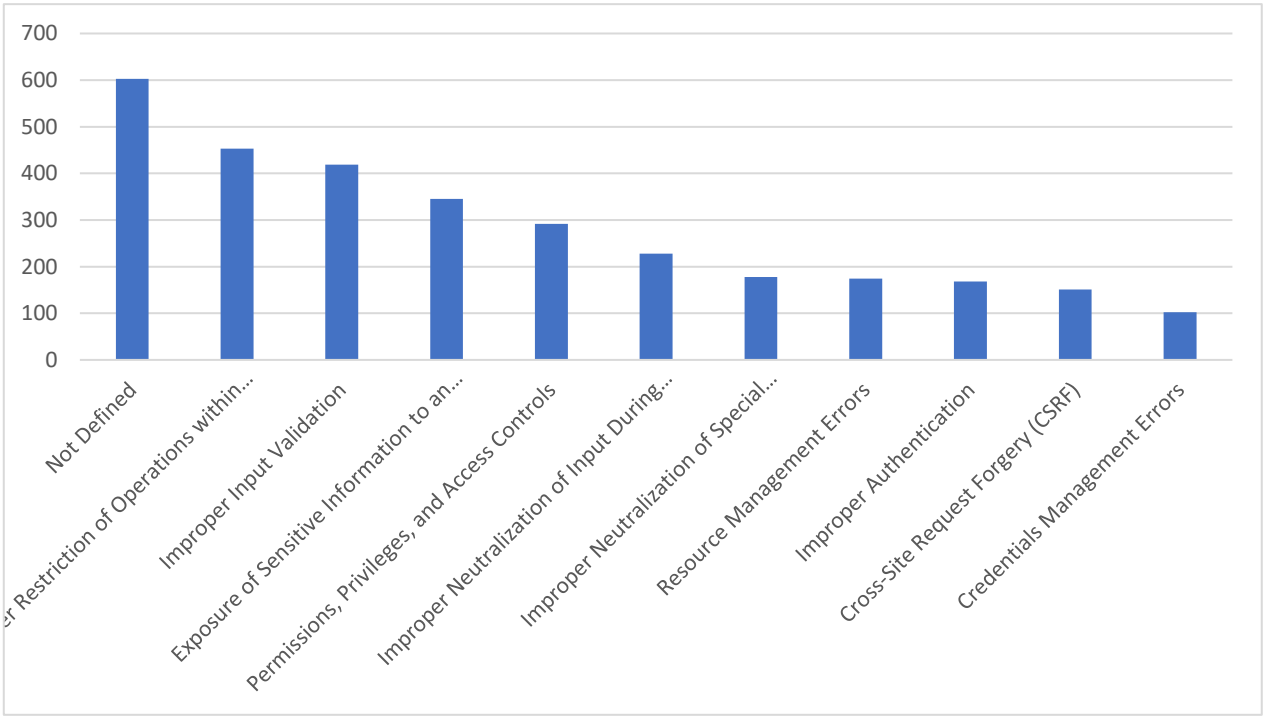

Table 7 Embedded systems CWE IDs

Overall, there were 111 different CWE IDs found to related to the CVE -entries used in this research, and the above 11 CWE IDs had more than a hundred entries connected to them. The following description of these CWE IDs was retrieved from cwe.mitre.org as it is the official CWE website and provides the most reliable information on them.

- **Not defined** means that no specific CWE ID has been defined for the CVE -entry, either because the particular weakness is not well defined enough to classify under a CWE ID, or is unique enough to not be categorized with others. Overall, the CVE -entry has insufficient information to accurately define the correct CWE ID for it.
- **Improper Restriction of Operations within the Bounds of a Memory Buffer (CWE-119)** is a software weakness that can enable potential attackers to execute arbitrary code, alter control flow, read sensitive information or crash the system by causing write or read operations to be performed on locations where the memory is associated with other variables, data structures or internal program data.
- **Improper Input Validation (CWE-20)** means that the software is not validating the inputs is received properly, which can enable attackers to provide inputs that can potentially alter control flow, control resources or execute code.
- **Exposure of Sensitive Information to an Unauthorized Actor (CWE-200)** enables non-authorized actors to access information they are not supposed to
- **Permissions, Privileges, and Access Controls (CWE-264)** contains weaknesses related to permissions, privileges and other access control related security features. Note that this CWE was obsoleted by NVD in 2016 as it overlaps with other categories, and hasn't been used since.
- **Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (CWE-79)** refers to software that do not remove uncontrollable inputs to be used in the out for a webpage. This enables attackers to input malicious scripts, for example, into the webpage.
- **Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (CWE-78)** is similar to the CWE ID above, but enables external inputs to the be given to OS instead of as website.
- **Resource Management Errors (CWE-399)** contains weaknesses that are related to improper management of system resources
- **Improper Authentication (CWE-287)** enables actors not provide sufficient proof of an authorized identity
- **Cross-Site Request Forgery (CSRF) (CWE-352)** concerns web applications that do not whether valid requests made by users are intentional or not

- **Credentials Management Errors (CWE-255)** relates to the category of weaknesses that deal with management of credentials (weak password encoding, no password aging, use of hard-coded credentials, etc.)

These eleven CWE categories contained 3113 of the CVE -entries used in this research, while the remaining 2026 entries were divided among the 100 CWE IDs not listed above, so the focus will be on the listed IDs as they contain most of the vulnerabilities in them.

Based on these CWE IDs, the most common vulnerability types of embedded systems are **denial of service, unauthorized code execution, overflow,** and **accessing sensitive information.** The entries that did not have an CWE ID connected to them had **denial of service** as the largest category, though multiple entries did not have enough information to even categorize the vulnerability type under a specific CWE, as can be seen from how large the not defined -category is in the above graph. Some of the not defined -entries did contain information about the vulnerability type, most commonly either **denial of service** or **unauthorized code execution**, but as the information is otherwise lacking, the validity of these entries should be put into question. Details for each of the vulnerabilities provided by previous research are as follows:

- Denial of service -attack is generally an attempt by an malicious entity to deny other entities to access specific services of resources, often by flooding the system with large amount of requests or disrupt the connection between multiple systems (Lau, Rubin, Smith, & Trajković, 2000)
- Overflow (or buffer overflow more commonly) are a common category of attack methods, which aim to modify the memory state of an program so that it gives control to the attacker (Ruwase & Lam, 2004)
- Code execution (or remote code execution) attacks are a specific type of XSS -attacks that allow client inputs to be stored and executed as a server side script, which can for example enable the attacker to access user data stored in a database (Zheng & Zhang, 2013)
- Access to sensitive information (or weak access control) means the attacker is able to circumvent protections mechanisms of the system or a device (for example password) to gain unauthorized access to information and functionalities (Papp et al., 2015)

There vulnerabilities are also the most common ones in all CVE -entries, so vulnerability trends do not significantly differ between embedded systems and all information system vulnerabilities on a first glance. The research will go into more details on the differences in a later chapter.

Previous research on the embedded systems specific entries by Papp et al. (2015) listed **denial of service, execute code, illegitimate access, integrity violation** and **information leakage** as the major vulnerabilities for embedded systems. Comparing these results to the ones in this research, the landscape of embedded systems vulnerabilities continues to be very similar to what it was in 2015. The

overall amount of entries continues to rise on a yearly basis, but the categorization has stayed relatively same.

6.4 Comparing embedded systems vs vulnerabilities overall

The number of CVE -entries added to the MITRE and NVD -databases has been growing each year overall, and this applies to entries concerning embedded systems as well.

First, we can compare the overall amount of CVE -entries to the amount of CVE -entries that was presented in the last chapter. The following graph was formed from information retrieved from cvedetails.com, as they have categorized all CVE -entries in a way that the data was easy to retrieve.

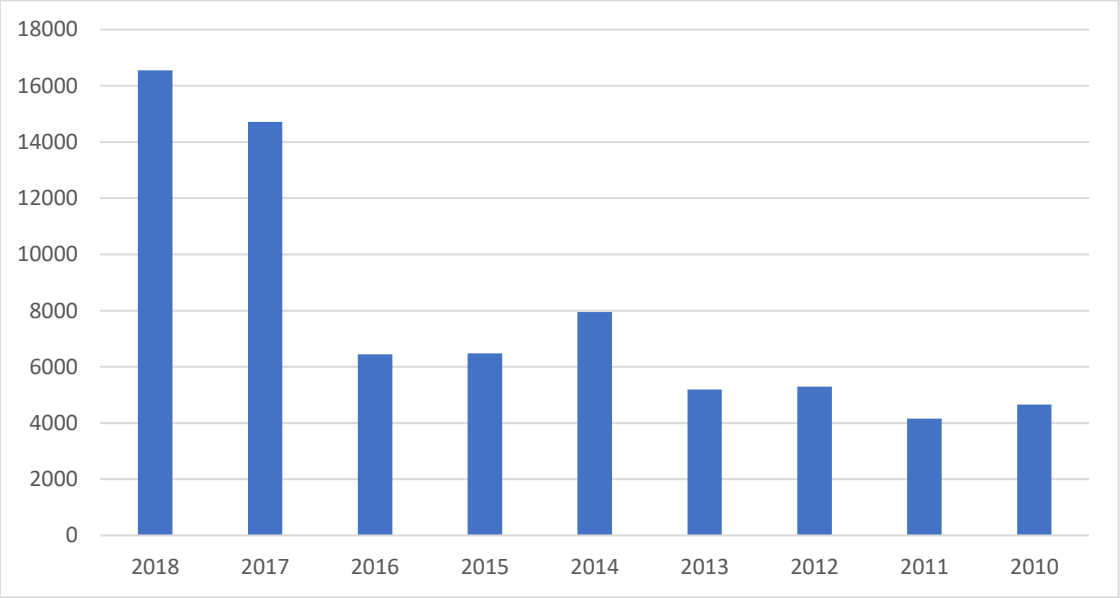

Table 8 All CVE -entries per year

Based on the graph the overall CVE numbers have been growing from 2010 to 2018, like entries concerning embedded systems, but the growth has not been as steady. The numbers fluctuated from 2010 to 2016, after there was a sizable spike in recorded entries (from 6447 to 14 714). As we can see when looking at the previous chapter, this does not directly correlate with how the amount of embedded system specific entries have evolved over the years, i.e. a similar spike in the amount entries cannot be seen, and the rise has been more stable. Above graph shows that there in the years 2015 and 2016 there was reduction in all reported CVE -entries, while entries concerning embedded systems stayed around the as the earlier year 2014, with a minor increase. Also, the large spike in the entries

between 2016 and 2017 does not happen with embedded systems entries, as the increase in much more linear from year to year.

The most common vulnerability types for CVEs in general according to older CVE trend analyses (Chang, Zavarsky, Ruhl, & Lindskog, 2011; Kuhn, Raunak, & Kacker, 2017; Neuhaus & Zimmermann, 2010) are **denial of service, unauthorized code execution, overflow** and **XSS (Cross-site scripting)**, with denial of service being the most common of these. This is also supported by the trend data stored on cvedetails.com, though they list code execution as the largest category. As seen in this research, cvedetails.com is missing information concerning some CVE -entries that can be found from the official MITRE and NVD datasets, so the information might not be entirely accurate, and would require further research to see if the amount of missing CVE -entries would skew the data enough to change the largest vulnerability type -category.

As mentioned earlier, the most common vulnerabilities for embedded systems are **denial of service, unauthorized code execution, overflow** and **accessing sensitive information.** This is mostly the same as with CVE -entries overall, with the difference that access to sensitive information appears to represent a larger part of the vulnerabilities than when including more than just entries that concern embedded systems.

Next, we compare the distribution of CWE IDs for all CVE -entries and to the embedded systems specific ones determined in this research. The following CWE IDs are listed as the ten most common ones in 2019, according to MITRE official description on the cwe.mitre.org website, in the order to most common to least. The top entries from 2018 were not offered on the portal, so the closest match was chosen, hence the 2019 list. The amount of entries was calculated by using the search functionalities on the NVD portal (as the MITRE portal does not offer similar functionality), between year 2010 and 2018 to match the data on embedded systems, so even though the IDs are based on the 2019 list, the amount of entries match the timespan for research data used for embedded systems.

- CWE-119 (8713 entries)
- CWE-79 (8209 entries)
- CWE-20 (5201 entries)
- CWE-200 (4978 entries)
- CWE-89 (2794 entries)
- CWE-22 (1827 entries)
- CWE-352 (1739 entries)
- CWE-125 (1616 entries)
- CWE-416 (1068 entries)
- CWE-190 (1054 entries)

Similar list for embedded systems CWE IDs (visualized earlier) is the following. Note that the not defined -entries are not included, due to the incomplete, and thus unreliable, information presented in them:

- CWE-119 (453 entries)
- CWE-20 (419 entries)
- CWE-200 (345 entries)
- CWE-264 (292 entries)
- CWE-79 (228 entries)
- CWE-78 (178 entries)
- CWE-399 (174 entries)
- CWE-287 (168 entries)
- CWE-352 (151 entries)
- CWE-255 (102 entries)

Comparing, CWE-119 is at the top of both lists. As this ID refers to general buffer errors caused by a variety of different attacks, and has been on the rise in the especially in the last few years (Kuhn et al., 2017), it is not surprising that it is the top CWE ID for both embedded systems and all IT systems in general. Most of the other CWE IDs also appear on both lists, with the most noticeable difference in how the ratio of embedded systems specific entries changes in relation to all entries. For example, for CWE-119 the embedded system specific entries comprise around 5,1% of the total entries related to that ID, but for CWE-352 this same ratio is 8,7%.

We can also see some unique CWE IDs on the embedded systems list. CWE-264 appears high on the list, but as the ID is considered somewhat obsolete as according to the official CWE description of the ID, it heavily overlaps with other IDs, and thus is not recommended to be used due to being inaccurate. This is why most of the mappings to this ID are older than 2016 as this was when the ID was obsoleted, though it is still being referenced in some more recent CVE -entries.

CWE-78 allows attackers to execute commands directly to the OS of a device, and based on this research, appears to be comparatively more of an issue with embedded systems. According to Kuhn et al. (2017) CWE-78 has been trending upwards in the recent years, which so a potential factor to this might be the increase in continual increase in the user of embedded systems.

CWE-339 is a larger category of weaknesses, similarly to CWE-264, meaning CVE -entries that refer to this do not provide precise information about the specific weakness, only informing that the weakness is related to improper management of system resources.

CWE-287 refers to a weakness in authenticating the actor that tries to interact with the software. Authentication is an important security feature concerning firmware (Choi, Lee, Na, & Lee, 2016), and as the analysis showed in this research, firmware is a security critical part of embedded systems, it is understandable that a CWE ID prevalent in firmware related vulnerabilities would appear high on the list.

7 DISCUSSION

The following primary research questions were presented at the start of the study portion of this thesis:

- Q1: What are the current and past trends in embedded system vulnerabilities?
- Q2: Can these trends be useful in predicting and preparing for future trends in embedded systems vulnerabilities?

Along with the following secondary question:

- Q3: Are there any noticeable weaknesses or missing information in the CVE datasets, or inconsistencies that could be improved on?

This section focuses on the results presented in the last chapter, whether the study answered the research questions fully, what implications the answer provide and how potential future studies can avoid the limitations of this study, and provide additional information on the subject.

To provide an answer to Q1, we need to look at the results of the analysis. First, we can see that **firmware** is the most vulnerable part of embedded systems, as it got the most keyword hits during the filtering process. This is also reinforced by other, earlier research, of similar nature (Costin et al., 2014; Costin, Zarras, & Francillon, 2015; Costin et al., 2017). Of the 5139 CVE -entries that remained after filtering and removing duplicates, 1278 entries mentioned firmware in their description. This amounts to almost 25% of all the listed entries, while the second most common keywords **Cisco** only got 663 hits (13%). Firmware being the most prevalent keyword is supported by previous research into embedded systems (Hou et al., 2017), as pointed out earlier in this research, firmware is considered a part of the physical hardware layer of embedded systems, which is critical in terms cybersecurity as accessing hardware can give to the software too (Elmiligi et al., 2016). Looking at the individual CVE -entries where firmware was mentioned, most of them concern firmware vulnerabilities on small electronic devices, like cameras, routers and smart phones.

The importance of the small electronic devices mentioned is also reinforced by the second and third largest keywords, **Cisco** and **Qualcomm**. Both of them are among the largest manufactures of devices that heavily use embedded systems, i.e. networking hardware, software, telecommunications equipment and other similar products. Common Cisco products are often routers, while Qualcomm products can be found from most mobile devices as they are the manufacturer of the Snapdragon -series processors. Analyzing the CVE -entries in this research does show that almost all of the entries that mentioned Qualcomm mentioned that the vulnerability concerns some version of the Snapdragon. The next two keywords, **mobile** and **router** further provide confirmation that the security

issues with embedded systems are heavily tied into these kinds of devices, specifically their firmware.

If we take a look at the most common CWE IDs that are tied to embedded systems based on this research, CWE-119 is the most common ID linked on the researched CVE -entries, which points to the fact that issues with buffer overflow is the most severe security issue for firmware for embedded systems. As was pointed out in the previous chapter, this is not surprising as it is the most common CWE ID for all CVE -entries, not just those which concern firmware. CWE-20 is a close second, with only 34 less entries, which tells us that input validation is almost as common an issue as buffer overflow is for embedded systems. CWE-20 is also common for all CVE -entries, but significantly less common than CWE-119 or CWE-79 are, which tells us that comparatively input validation is more of an issue with embedded systems than it is when considering all vulnerabilities. Most of the CVE -entries for CWE-20 concerned either Cisco products, or routers. As Cisco produces different models of routers, we can safely assume that this CWE ID is most common router devices. CWE-79 is the second most common CWE ID for all CVE -entries, but for embedded systems it is much less common, and most of the related entries concern firmware, which tells us that along with buffer overflow, neutralizing improper user-controllable inputs is also a noticeable issue, though not as large of an issue for embedded systems as it is for other IT systems. Very similarly CWE-78 has around the same amount of this as CWE-79, which is understandable as the weakness also concerns improper neutralization, though instead of input validation CWE-78 concerns OS command injections. CWE-339 interestingly appears to be common only for CVE -entries referring to for Cisco products, but as pointed our earlier this ID only refers to general issues with managing systems resources, so it cannot be easily used to pinpoint security weaknesses. CWE-200 is most common among entries concerning firmware once again, and also provided a number of hits with the keyword local. Most of the entries in this keyword refer to vulnerabilities where malicious entities exploit the system through a local attack, and based on CWE-200 often leads to exposure of sensitive information to the attacker. CWE-264, as mentioned in earlier, is an obsoleted CWE ID so CVE -entries to it are more or less either outdated or refer to incorrect CWE ID, as CWE-264 should no longer be used.

Answering Q1 based on this analysis, we can see that like CVE -entries in general, those entries related to embedded systems have been steadily on the rise since at least from 2010, and will most likely continue to rise as the number of embedded devices keeps growing. Most of the security vulnerabilities for these systems are related to firmware, with buffer overflow being the most common weakness and improper input validation being a close second. A large portion of embedded systems vulnerabilities are also focused on Cisco and Qualcomm products, which was also noted by Papp et al. (2015) concerning Cisco, though this can be explained by the fact that there are simply more products from these two brands than others, not that they are necessarily more vulnerable than similar devices from other manufacturers.

If we take a look at how vulnerabilities and weaknesses have evolved over the years, as the seen in the chapter analyzing the results, the number of entries has been steadily growing each year, with a noticeable spike both in 2017 and 2018. Looking into specific weaknesses, CWE-119 started increasing in the analyzed entries around 2012-2013, with a steady yearly increase until in 2017 the amount it spiked very noticeably, which also explains the spike in the overall amount embedded system specific entries. For example, almost 70% of the CWE-119 IDs for the CVE -entries concerning firmware were in either 2017 or 2018. Earliest CWE-119 entries can be found from 2010 concerning some Cisco products, potentially even earlier but data before 2010 was not analyzed in this research due to reasons explained earlier. CWE-20 has appeared more consistently on CVE -entries from 2010-2018, with a small increase in the last few years, but much less than CWE-119. Distribution for CWE-79 and CWE-78 in the research data are also relatively even, although years 2010 and 2011 have very few entries. CWE-200 mostly follows the same pattern; only few results in the earlier years, with a heavy increase in 2017 and 2018. For CWE-339, all the entries that appeared in the research data were from 2016 and before, which is important to note as the CWE ID was invalidated in 2016, so any use after would be inaccurate. The same cannot be unfortunately said about CWE-264, as even though this ID has also been obsoleted by MITRE and NVD, it still appears in a few entries even from 2018. Though on a positive note, there are only a few CVE -entries in the data that refer to it. CWE-200 started appearing in the analyzed data from 2011 onwards, but was relatively uncommon until 2016, before which there only a handful of references to it in the analyzed CVE -entries. From 2016 onward the ID has become more common, though not on level of CWE-119 or CWE-20.

Answering Q1 based on the presented information, embedded systems vulnerabilities have experienced mostly a steady growth since 2010, with the most noticeable increases in 2012-2013, when the number of reported entries almost doubled, and 2016-2017 when the reported entries increased by around 25%. The largest part of the increase falls under the keyword firmware. For the other keywords, the increases in entries have not been as linear. Cisco and Qualcomm had a lot of entries reported for both of them in 2015 and 2016, but in 2017 there was a sharp decline in the reported entries. As the overall amount of Cisco and Qualcomm products has not decreased in those years, this decline could be either explained by improved cybersecurity measures implemented by the companies on their products, or simply that no new vulnerabilities were reported that year due to other reasons, for example new products had vulnerabilities that already had an CVE -entry. Reported entries overall declined heavily between the years 2015 and 2016 for most of the keywords. The only reason why the entry numbers between these two years stayed close to together was that both firmware and Qualcomm related entries increased, which kept the number of reported entries roughly equal. Qualcomm and Cisco related entries have overall stayed low after the decline in 2016, while entries for the keyword mobile saw a sharp increase (from 66 to 144) between 2016 and 2017, which along with firmware related entries mostly explain the overall growing number of reported entries. Based on

this analysis, while the overall number of reported entries does keep increasing every year, the affected systems, and thus the trends, do change overtime. Cisco and Qualcomm devices seem to not contain too many new reported vulnerabilities, while embedded systems firmware and mobile devices are receiving new reported vulnerabilities at an increasing pace.

Using this analysis to answer Q2, using past trend in embedded system vulnerabilities can be difficult to use to predict future trends, at least in the scope of this research. We do know that the number of embedded systems in continually increasing, so in that sense it can be safe to assume that more CVE -entries will be reported in the future at with a steady increase. Predicting which types of systems can be more difficult, at least based on this research. Based on the past trends we can for example assume that Cisco and Qualcomm products will not be as prevalent in CVE -data as they were earlier, but on the other hand vulnerabilities with firmware and mobile devices will be reported more, but this will also depend on what possible new applications for embedded systems there will be in the future, and their security is being developed. For example, smart homes are becoming more common will provide new challenges for developing security measures, as they heavily employ embedded systems with IoT related solutions, and one vulnerable device can then make all the connected devices potentially vulnerable, instead of just the single device (Anthi, Williams, Slowinska, Theodorakopoulos, & Burnap, 2019).

7.1 Differences and issues in the MITRE and NVD datasets

This chapter aims to answer the secondary research question (Q3) of the thesis, i.e. are there any noticeable weaknesses or missing information in the CVE datasets, which could be improved upon in the future. This chapter also focuses on the differences in the MITRE and NVD datasets, and the issues noticed in both during this research. The two datasets were very similar as expected, since their purpose is store information in the same format and synchronize the entries between each other, but some small differences can still be noted. Zhang et al. (2011) point out the following weaknesses in NVD datasets:

- Missing information, as for example, many Microsoft related entries are missing version information.
- The date of vulnerabilities depends on the date when vendors release this information, not the date when the vulnerability is discovered.
- Data errors, i.e. false information provided erroneously most of the time.

The experiences when conducting this research mostly reinforce these issues, for both the MITRE and the NVD datasets. The NVD also has issues with chronological inconsistency, inclusion, separation of event and the documentation of

vulnerabilities (Ozment, 2007), which also applies to the MITRE -datasets as they store vulnerability data in the same format. The chronological issues were apparent in this research when trying to categorize CVE -entries based on the year, as CVE -entry is named based on the year of reporting for the entry (for example, CVE-2010 -prefix means the CVE -entry was created in 2010), but numerous entries are being updated after their creation, as the information concerning the vulnerability has changed. This creates issues when trying to process large amounts of CVE -data, as it can limit the usefulness of using the name of the CVE -entry for filtering the data. In this research it was decided to categorize entries based on their year of release, but more advanced research that could take into account the update years for the entries might have more issues with the categorization, depending on the method of processing the CVE -data.

In terms of differences between the datasets for the purposes of this research, the first effect was that both of the datasets gave a very similar amount of results, as pointed out by mitre.org, NVD and CVE databases are synchronized to update with the same entries, meaning any differences in the amount of keyword hits during the filtering were due to differences in the entry descriptions. A more relevant difference is that the NVD CVE -entries have the <vuln:cwe> -field, which contains the information which CWEs can applied to the CVE, and as mentioned earlier in this research, this makes it easier for us to connect correct CWEs to CVE -entries and form a better view on what vulnerabilities are relevant to embedded systems. This field is not in the MITRE CVE -datasets, making this a clear improvement in how NVD stores CVE information when compared to MITRE.

NVD also has the advantage over MITRE in how they allow the search filtering of the datasets on their portal. MITRE allows users to search based on the the CVE ID, or keywords that might appear on the entries. NVD allows the same, but also has enables user to filter based on the date when the CVE was created or updated, vendor, product, or most importantly for this research, the related CWE ID. NVD also allows users to display statistics of the search criteria used, for example how the distribution of a specific CWE ID has been over the years. Even though this comparison of search functionalities is not strictly related to the actual data stored in the datasets, it is a feature of the NVD and MITRE portals that aids with filtering and analyzing the data.

Another improvement for how the information is presented on both MITRE and NVD would be to better separate valid and current CVE -entries from those that are currently undergoing revisions or reanalysis. As vulnerabilities and defense mechanisms are constantly evolving in IT, it is a natural requirement that the entries should be kept up to date whether they are still relevant or not, but the currently this information can only be seen by accessing the description of the individual entries, which can make a mass analysis of the datasets difficult as it requires either more advanced automation when analyzing the entries, or more manual work to vet out the non-relevant entries. This also applies to the RESERVED and DISPUTED entries, as even though it is important to keep the CVE numbers reserved for them in order to not to distort the validity of the data (for

example by retroactively reducing the amount of new CVE -entries reported at a certain year), they also can be difficult to remove from data used in large analyses.

Further issues with CVE datasets from both NVD and MITRE come from their unevenness (Neuhaus & Zimmermann, 2010). What this means that the quality of the CVE -entries can have a great amount of variance on the quality of the reported information. Some of the entries contain very limited information, increasing the difficulty of determining if the vulnerability is actually related to embedded systems for the purposes of this research. This can be seen very well in the amount of **not defined** -entries listed in an earlier chapter. These entries contained varying amount of information concerning the CVE, so with many of them it was possible to deduce whether they were relevant for this research or not, but there were also entries that were not possible to categorize in the scope of this research, and as such were discarded. Neuhaus & Zimmermann suggest that further manual validation of CVE -entries would increase their reliability, as now for example identical vulnerabilities can be reported due to different vendors reporting them with slightly different terminology, thus artificially increasing the number CVE -entries.

7.2 Limitations and future work

Some of the limitations were mentioned earlier in chapter 5 during the discussion on the reliability and validity of this research. Now we go into more detail about them and other weaknesses in this research.

The first limitation was in the chosen datasets. Both NVD and MITRE store CVE -information in the same format, and regularly synchronize the information between them according to the information on their website. This makes comparing the datasets from the two sites redundant in many ways, as the data in almost the same. During the filtering process there were small differences in the amount of hits for some of the keywords, but they were not enough to make a significant impact on the results. For future research, other data sources should be considered in addition CVE -based, as the this would provide another point of view on vulnerability trends. For example, Exploit Database (exploit-db.com) is CVE compliant in how they store vulnerability information, so comparison between their datasets with MITRE and NVD should be relatively easy as the information is in the same format. This would increase the validity and reliability of future studies, as well as providing a more comprehensive view of CVEs.

Another potential weakness is in the chosen method of filtering the datasets. Since the keyword-based search relied on specific words appearing in the CVE -entry for it to be flagged as valid for this research, this a noticeable change of producing false results. This is why a manual filtering was conducted after the keyword search to pick out any remaining non-valid entries, but this still leaves the chance that some valid entries related to embedded systems were not picked by the keyword search, as they did not contain any of the used keywords. Adding more keywords could help with the filtering, though already with the amount of

keywords used in this research, significant number of the flagged entries were duplicates, so new keywords would need to be chosen carefully. Also, using a blacklist of keywords along with the whitelist, to filter out entries where a specific keyword was mentioned instead of the other way around, would help to produce more accurate results. Comparison to other similar research to see how the results match is another effective way to improve the research, though at least while gathering reference material for this research it was difficult to find more than one or two similar ones.

More advanced research could also go into more detail about the specific vulnerabilities and weaknesses connected to them, as this research mostly focused on the general trends of embedded systems vulnerabilities. Future research could also conduct practical tests to see how valid specific CVE -entries, as this was out of scope for this research.

8 CONCLUSION

In this research the trends of cybersecurity vulnerabilities related to embedded systems was studied in comparison to vulnerability trends in general. The research was done by retrieving vulnerability data in CVE -format from NVD and MITRE official CVE -datasets. Research was done by conducting a literature review and a quantitative data analysis. Most of the used literature was academic nature, but also official non-academic sources were used to explain some of the concepts and terms presented in this research, especially those concerning CVEs and CWEs. Most notable findings in this research were that embedded system vulnerabilities have been growing on a yearly basis, with increased growth in the last few years used in this analysis, and most of the current vulnerabilities are related to firmware and mobile devices. Most significant weaknesses for embedded systems are buffer overflow and denial-of-service, which also are very common for non-embedded systems based on the vulnerability information stored by MITRE and NVD. This means that embedded systems vulnerabilities do follow general vulnerability trends, with the largest categories for CVE and CWE being similar, and differences appearing in the smaller categories. Based on this research, it is likely that the amount of reported vulnerabilities for embedded systems will continue to rise along with reported vulnerabilities in general, but it is difficult to predict whether the ratio to vulnerabilities in general will grow smaller or larger. Also the security requirements for embedded systems are continuously rising (Zaddach & Costin, 2013), which also indicates that embedded systems security will continue to be rise in importance.

This study introduced the concepts of CVE (Common Vulnerabilities and Exposures) and CWE (Common Weakness Enumeration), and how they relate to the cybersecurity in general, and the cybersecurity of embedded systems. These concepts were used to describe how the cybersecurity trends for embedded systems have evolved between 2010 and 2018, and how they compare to general cybersecurity trends. Previous study on CVE -entries has been conducted multiple times, but only a few studies were found focusing specifically on embedded system vulnerabilities by using CVE and CWE as the basis of the analysis. Most significant study conducted on a similar subject was done by Papp et al. (2015), and this was used as the basis of this research.

Most of the material found online related to embedded system vulnerabilities in non-academic by its nature, focusing more on providing technical information on the subject to aid in preventing vulnerabilities from being exploited and explaining its background, instead of providing peer-reviewed academic analysis on the subject. Though research focusing on firmware of embedded systems was found and used in this research to reinforce the main conclusion that firmware is one of the most vulnerable parts of embedded systems (Costin et al., 2014, 2017; Zaddach & Costin, 2013) This made gathering and categorizing the research data relatively easy, as also provided an easily accessible sources of significant amounts of data. But it also meant that finding similar research to form

a frame of reference to this research was more difficult, which can be seen in parts of this research that explain the concepts of CVE and CWE, as most of the information there is not from academical, but instead directly from the primary sources of CVE and CWE data.

The amount of data used in this research was sufficient to draw conclusions on embedded system vulnerabilities, and these observations were detailed in the previous chapter. The limitations of this study were on the chosen research method, as in the scope of this research, doing a more detailed analysis of the obtained data was not feasible. This leaves the conclusions of this research as relatively general. The accuracy of the data could also be improved by using more than a whitelist of keywords to filter the data, for example by also using a blacklist. More sources for the CVE -data would also improve the accuracy, as the used datasets (MITRE and NVD) provided very similar results when analyzing the entries. Future research on the subject should include more recent data on the analysis, as well as including different data sources. More analysis also on the reasons of why specific vulnerabilities are weaknesses are prevalent in embedded systems should also be done. A more detailed look on the reasons why specific years on the used data had no noticeable increase in reported CVE -entries, and why other had a significant jump, should also be considered.

Results on this study should be used to give a general view on embedded system vulnerabilities and how they have evolved over the years, in comparison to vulnerability entries in general. The results could form a basis from which a more detailed research could be conducted on specific aspects of the vulnerability trends.

REFERENCES

About CPE (2013, March 22) Retrieved May 14, 2018, from https://cpe.mitre.org/about/

About CVE (2018, January 17) Retrieved May 10, 2018, from https://cve.mitre.org/about/index.html

About CWE. (2018, March 30). Retrieved May 14, 2018, from https://cwe.mitre.org/about/index.html

About OVAL (2014, May 13) Retrieved May 17, 2018, from https://oval.mitre.org/about/

Anthi, E., Williams, L., Slowinska, M., Theodorakopoulos, G., & Burnap, P. (2019). A Supervised Intrusion Detection System for Smart Home IoT Devices. *IEEE Internet of Things Journal*, *6*(5), 9042–9053. https://doi.org/10.1109/JIOT.2019.2926365

Buttner, A., & Ziring, N. (2009). CPE Specification 2.2 Common Platform Enumeration (CPE) – Specification. Retrieved from http://cpe.mitre.org/files/cpe-specification_2.2.pdf

Chang, Y. Y., Zavarsky, P., Ruhl, R., & Lindskog, D. (2011). Trend analysis of the CVE for software vulnerability management. *Proceedings - 2011 IEEE International Conference on Privacy, Security, Risk and Trust and IEEE International Conference on Social Computing, PASSAT/SocialCom 2011*, 1290–1293. https://doi.org/10.1109/PASSAT/SocialCom.2011.184

Choi, B. C., Lee, S. H., Na, J. C., & Lee, J. H. (2016). Secure firmware validation and update for consumer devices in home networking. *IEEE Transactions on Consumer Electronics*, *62*(1), 39–44. https://doi.org/10.1109/TCE.2016.7448561

Costin, A., Zaddach, J., Francillon, A., & Balzarotti, D. (2014). A Large-Scale Analysis of the Security of Embedded Firmwares. *USENIX Security Symposium*, 95–110. Retrieved from https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/costin%5Cnhttps://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-costin.pdf

Costin, A., Zarras, A., & Francillon, A. (2015). Automated Dynamic Firmware Analysis at Scale: A Case Study on Embedded Web Interfaces. https://doi.org/10.1145/2897845.2897900

Costin, A., Zarras, A., & Francillon, A. (2017). Towards automated classification

of firmware images and identification of embedded devices. In *IFIP Advances in Information and Communication Technology* (Vol. 502, pp. 233–247). Springer. https://doi.org/10.1007/978-3-319-58469-0_16

Elmiligi, H., Gebali, F., & Watheq El-Kharashi, M. (2016). Multi-dimensional analysis of embedded systems security. *Microprocessors and Microsystems*, *41*, 29–36. https://doi.org/10.1016/j.micpro.2015.12.005

Fournaris, A. P., & Sklavos, N. (2014). Secure embedded system hardware design - A flexible security and trust enhanced approach. *Computers and Electrical Engineering*, *40*(1), 121–133. https://doi.org/10.1016/j.compeleceng.2013.11.011

Guo, M., & Wang, J. A. (2009). An Ontology-based Approach to Model Common Vulnerabilities and Exposures in Information Security. *ASEE Southest Section Conference*. Retrieved from http://icee.usm.edu/icee/conferences/ASEE-SE-2010/Conference Files/ASEE2009/papers/PR2009034GUO.PDF

Gürgens, S., Rudolph, C., Maña, A., & Nadjm-Tehrani, S. (2010). Security engineering for embedded systems. *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems - S&D4RCES '10*, 1. https://doi.org/10.1145/1868433.1868443

Hintze, D., Hintze, P., Findling, R. D., & Mayrhofer, R. (2017). A Large-Scale, Long-Term Analysis of Mobile Device Usage Characteristics. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, *1*(2), 1–21. https://doi.org/10.1145/3090078

Hou, J. B., Li, T., & Chang, C. (2017). Research for Vulnerability Detection of Embedded System Firmware. In *Procedia Computer Science* (Vol. 107, pp. 814–818). https://doi.org/10.1016/j.procs.2017.03.181

Humayed, A., Lin, J., Li, F., & Luo, B. (2017). Cyber-Physical Systems Security - A Survey. *IEEE Internet of Things Journal*, *4*(6), 1802–1831. https://doi.org/10.1109/JIOT.2017.2703172

Jormakka, O. (2019). *Approaches and challenges of automatic vulnerability classification using natural language processing and machine learning techniques*. Retrieved from https://jyx.jyu.fi/handle/123456789/66196

Kim, L. W., & Villasenor, J. D. (2014). Dynamic function replacement for system-on-chip security in the presence of hardware-based attacks. IEEE Transactions on reliability, 63(2), 661-675.

Knight, J. C. (2002). Safety critical systems: challenges and directions. *Proceedings of the 24rd International Conference on Software Engineering (ICSE), 2002. IEEE.*, 547–550. https://doi.org/10.1145/581339.581406

Kuhn, R., Raunak, M., & Kacker, R. (2017). It Doesn't Have to Be Like This: Cybersecurity Vulnerability Trends. *IT Professional*, (November), 66–70. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8123486&isnumber=8123452

Lau, F., Rubin, S. H., Smith, M. H., & Trajković, L. (2000). *Distributed denial of service attacks*. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics* (Vol. 3). https://doi.org/10.1109/ICSMC.2000.886455

Lee, E. A. (2008). Cyber Physical Systems: Design Challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)* (pp. 363–369). https://doi.org/10.1109/ISORC.2008.25

McLoughlin, I. (2008). Secure embedded systems: The threat of reverse engineering. *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, 729–736. https://doi.org/10.1109/ICPADS.2008.126

Mell, P, & Grance, T. (2002). Use of the common vulnerabilities and exposures (cve) vulnerability naming scheme. *NIST Special Publication*, *September*, 1–4. Retrieved from http://www.dtic.mil/docs/citations/ADA407728%0Ahttp://tim.kehres.com/docs/nist/sp800-51.pdf

Mell, Peter, Scarfone, K., & Romanosky, S. (2007). A Complete Guide to the Common Vulnerability Scoring System Version 2.0. *FIRSTForum of Incident Response and Security Teams*, 1–23. Retrieved from http://www.nazimkaradag.com/wp-content/uploads/2014/11/cvss-guide.pdf

Narayanan, V., & Xie, Y. (2006). Reliability concerns in embedded system designs. *Computer*, *39*(1), 118–120. https://doi.org/10.1109/MC.2006.31

Neuhaus, S., & Zimmermann, T. (2010). Security trend analysis with CVE topic models. In *Proceedings - International Symposium on Software Reliability Engineering, ISSRE* (pp. 111–120). https://doi.org/10.1109/ISSRE.2010.53

Noergaard, T. (2013). *Embedded systems architecture : a comprehensive guide for engineers and programmers*. Newnes. Retrieved from https://books.google.fi/books?hl=fi&lr=&id=96jSXetmlzYC&oi=fnd&pg=PP1&dq=embedded+systems+architecture&ots=3o_ePKRgUV&sig=em2czf4x3AvqjvTmqIjggkex96M&redir_esc=y#v=onepage&q&f=true

Ozment, A. (2007). Vulnerability Discovery and Software Security, 139. Retrieved from http://andyozment.com/papers/ozment_dissertation.pdf

Papp, D., Ma, Z., & Buttyan, L. (2015). Embedded systems security: Threats, vulnerabilities, and attack taxonomy. *2015 13th Annual Conference on Privacy, Security and Trust, PST 2015*, 145–152. https://doi.org/10.1109/PST.2015.7232966

Parameswaran, S., & Wolf, T. (2008). Embedded systems security—an overview. *Des Autom Embed Syst*, *12*, 173–183. https://doi.org/10.1007/s10617-008-9027-x

Radack, S., & Kuhn, R. (2011). Managing Security Using the Security Content Automation Protocol How SCAP Helps Organizations Manage Security and Comply With Reporting Requirements. *IT Professional*, 9–11. Retrieved from https://ws680.nist.gov/publication/get_pdf.cfm?pub_id=907372

Ravi, S., Raghunathan, A., Kocher, P., & Hattangady, S. (2004). Security in embedded systems: Design challenges. *ACM Trans.Embed.Comput.Syst.*, *3*(3), 461–491. Retrieved from http://www.cs.ucsb.edu/~sherwood/cs290/papers/secure_embeded_kocher.pdf

Ruwase, O., & Lam, M. S. (2004). *A Practical Dynamic Buffer Overflow Detector*. *Proceedings of the 11th Annual Network and Distributed System Security Symposium*. https://doi.org/10.1145/780822.781150

Scarfone, K., & Mell, P. (2009). An Analysis of CVSS Version 2 Vulnerability Scoring 1. Retrieved from https://ws680.nist.gov/publication/get_pdf.cfm?pub_id=903020

Tripathi, A., & Singh, U. K. (2012). Taxonomic Analysis of Classification Schemes in Vulnerability Databases. *2011 6Th International Conference on Computer Sciences and Convergence Information Technology (Iccit)*, 686–691.

Ukil, A., Sen, J., & Koilakonda, S. (2011). Embedded security for internet of things. *Proceedings - 2011 2nd National Conference on Emerging Trends and Applications in Computer Science, NCETACS-2011*, 50–55. https://doi.org/10.1109/NCETACS.2011.5751382

Vai, M., Nahill, B., Kramer, J., Geis, M., Utin, D., Whelihan, D., & Khazan, R. (2015). Secure architecture for embedded systems. *2015 IEEE High Performance Extreme Computing Conference, HPEC 2015*, 1–5. https://doi.org/10.1109/HPEC.2015.7322461

Zaddach, J., & Costin, A. (2013). Embedded Devices Security and Firmware Reverse Engineering. *Black Hat USA*, 9. Retrieved from https://media.blackhat.com/us-13/US-13-Zaddach-Workshop-on-Embedded-Devices-Security-and-Firmware-Reverse-Engineering-WP.pdf

Zhang, C., Vahid, F., & Najjar, W. (2003). A highly configurable cache architecture

for embedded systems. *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium On*, 136–146. https://doi.org/10.1109/ISCA.2003.1206995

Zhang, S., Caragea, D., & Ou, X. (2011). An empirical study on using the national vulnerability database to predict software vulnerabilities. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 6860 LNCS, pp. 217–231). https://doi.org/10.1007/978-3-642-23088-2_15

Zheng, Y., & Zhang, X. (2013). *Path sensitive static analysis of web applications for remote code execution vulnerability detection*. *Proceedings - International Conference on Software Engineering*. https://doi.org/10.1109/ICSE.2013.6606611

APPENDIX 1 MITRE CVE -DATASET EXAMPLE ENTRY

```
<item type="CAN" name="CVE-2017-9231" seq="2017-9231">
<status>Candidate</status>
<phase date="20170524">Assigned</phase>
<desc>XML external entity (XXE) vulnerability in Citrix XenMobile Server 9.x and 10.x before 10.5 RP3
      allows attackers to obtain sensitive information via unspecified vectors.</desc>
<refs>
<ref source="CONFIRM" url="https://support.citrix.com/article/CTX220138">https://support.citrix.com/article/CTX220138</ref>
<ref source="BID" url="http://www.securityfocus.com/bid/98995">98995</ref>
<ref source="SECTRACK" url="http://www.securitytracker.com/id/1038704">1038704</ref>
</refs>
<votes>
</votes>
<comments>
</comments>
</item>
```

APPENDIX 2 NVD CVE -DATASET EXAMPLE ENTRY

```
<entry id="CVE-2017-0002">
  <vuln:vulnerable-configuration id="http://nvd.nist.gov/">
    <cpe-lang:logical-test operator="OR" negate="false">
      <cpe-lang:fact-ref name="cpe:/a:microsoft:edge"/>
    </cpe-lang:logical-test>
  </vuln:vulnerable-configuration>
  <vuln:vulnerable-software-list>
    <vuln:product>cpe:/a:microsoft:edge</vuln:product>
  </vuln:vulnerable-software-list>
  <vuln:cve-id>CVE-2017-0002</vuln:cve-id>
  <vuln:published-datetime>2017-01-10T16:59:00.133-05:00</vuln:published-datetime>
  <vuln:last-modified-datetime>2017-01-17T21:59:18.813-05:00</vuln:last-modified-datetime>
  <vuln:cvss>
    <cvss:base_metrics>
      <cvss:score>6.8</cvss:score>
      <cvss:access-vector>NETWORK</cvss:access-vector>
      <cvss:access-complexity>MEDIUM</cvss:access-complexity>
      <cvss:authentication>NONE</cvss:authentication>
      <cvss:confidentiality-impact>PARTIAL</cvss:confidentiality-impact>
      <cvss:integrity-impact>PARTIAL</cvss:integrity-impact>
      <cvss:availability-impact>PARTIAL</cvss:availability-impact>
      <cvss:source>http://nvd.nist.gov</cvss:source>
      <cvss:generated-on-datetime>2017-01-11T09:49:58.903-05:00</cvss:generated-on-datetime>
    </cvss:base_metrics>
  </vuln:cvss>
  <vuln:cwe id="CWE-264"/>
  <vuln:references xml:lang="en" reference_type="VENDOR_ADVISORY">
    <vuln:source>MS</vuln:source>
    <vuln:reference href="http://technet.microsoft.com/security/bulletin/MS17-001" xml:lang="en">MS17-001</vuln:reference>
  </vuln:references>
  <vuln:references xml:lang="en" reference_type="UNKNOWN">
    <vuln:source>BID</vuln:source>
    <vuln:reference href="http://www.securityfocus.com/bid/95284" xml:lang="en">95284</vuln:reference>
  </vuln:references>
  <vuln:references xml:lang="en" reference_type="UNKNOWN">
    <vuln:source>SECTRACK</vuln:source>
    <vuln:reference href="http://www.securitytracker.com/id/1037573" xml:lang="en">1037573</vuln:reference>
  </vuln:references>
  <vuln:summary>Microsoft Edge allows remote attackers to bypass the Same Origin Policy via vectors involving the about:blank URL and data: URLs, aka "Microsoft Edge Elevation of Privilege Vulnerability."</vuln:summary>
</entry>
```

APPENDIX 3 CWE -DATASET EXAMPLE ENTRY

```
<Weakness ID="28" Name="Path Traversal: '..\filedir'" Abstraction="Variant" Structure="Simple" Status="Incomplete">
   <Description>The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize "..\" sequences that can resolve to a
   <Extended_Description>
      <xhtml:p>This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.</xhtml:p>
      <xhtml:p>The '..\' manipulation is the canonical manipulation for operating systems that use "\" as directory separators, such as Windows. However, it is also useful for bypassing path t
   </Extended_Description>
   <Related_Weaknesses>
      <Related_Weakness Nature="ChildOf" CWE_ID="23" View_ID="1000" Ordinal="Primary"/>
      <Related_Weakness Nature="ChildOf" CWE_ID="23" View_ID="699" Ordinal="Primary"/>
   </Related_Weaknesses>
   <Applicable_Platforms>
      <Language Class="Language-Independent" Prevalence="Undetermined"/>
      <Operating_System Class="Windows" Prevalence="Undetermined"/>
   </Applicable_Platforms>
   <Modes_Of_Introduction>
      <Introduction>
         <Phase>Implementation</Phase>
      </Introduction>
   </Modes_Of_Introduction>
   <Common_Consequences>
      <Consequence>
         <Scope>Confidentiality</Scope>
         <Scope>Integrity</Scope>
         <Impact>Read Files or Directories</Impact>
         <Impact>Modify Files or Directories</Impact>
      </Consequence>
   </Common_Consequences>
   <Potential_Mitigations>
      <Mitigation Mitigation_ID="MIT-5.1">
         <Phase>Implementation</Phase>
         <Strategy>Input Validation</Strategy>
         <Description>
            <xhtml:p>Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Re
            <xhtml:p>When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inpu
            <xhtml:p>Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, espe
            <xhtml:p>When validating filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weakne
            <xhtml:p>Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a blacklist, which may be incomplete (CWE-184). For e
         </Description>
      </Mitigation>
      <Mitigation Mitigation_ID="MIT-20">
         <Phase>Implementation</Phase>
         <Strategy>Input Validation</Strategy>
         <Description>Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180). Make sure that the application does not
      </Mitigation>
```