Creating a Configuration Security Layer for Embedded Devices

A research-based on the case study of a widely used Embedded Device

Master thesis submitted to Delft University of Technology
in fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in Embedded Systems,
Faculty of Electrical Engineering, Mathematics and Computer Science

January 10, 2020

Student ID: 4633474
Lazaros Lazaridis
Master's Student at TU Delft / Software Engineer at Atos, Greece

Graduation Committee

Chairperson	: Prof.dr. P. Hartel, Section Cyber Security
First Supervisor	: Dr.ir. C. H. Gañán, Section Organisation & Governance
Second Supervisor	: Dr.ir. Mitra Nasri, Section Real-Time Systems

Preface

This thesis project concludes my master's degree in Embedded Systems at the Delft University of Technology. My research was carried out in collaboration with Atos. I worked closely with the Quality Assurance team and the Platform team for the OpenScape Business product. During my research I investigated ways to protect embedded/IoT devices; devices that are currently being deployed in the field.

And yet, this was more than just a thesis project for me. Trite as it might sound, it was a journey. I was offered an opportunity to expand my horizons, but, more importantly, I was lucky enough to assist in preventing the threat that embedded devices could potentially pose. By doing so, I fulfilled a longtime wish of mine: to contribute to society through my work. I could not have asked for much more.

Then, of course, there is the human aspect of the story. I feel more than grateful for meeting such amazing people here at Atos. Through this journey, I evolved both as an engineer and as a human being. I kindly thank my daily Atos supervisor, Grigoris Dedes, for his guidance and total support throughout the project, and Vassilis Stoumpos for giving me the opportunity to work on this project for Atos. Panos Katsikogiannis, Nasos Priftis and Panagiotis Tamtamis for being my technical mentors, and at the same time a source of inspiration. I want to thank the Platform and Quality Assurance teams for their indispensable contribution to the experiment, and for their willingness to assist me in any request I ever had. In TU Delft, I want to thank Carlos Hernandez Gañán for believing in the idea behind my project and offering me the opportunity to make it happen; Peter Hartel and Mitra Nasri for their feedback which has been invaluable and most constructive. Last but not least, I simply must express my gratitude towards my family, the most wonderful family I could have ever hoped for; I warmly thank them for their support, much needed, for their pleasant holiday gatherings, and for believing in me when I certainly did not. I want to thank my roommates for their patience. I want to thank Dask, Lander, River, Joni, Nedor, Mano, Detska, Baron, Tzo^2, Viko; you are all part of my family. Giota, I am more than lucky to have your love, smile, and support throughout this whole time, from the very beginning. If everyone had people like the ones I have in my life, earth would be a much better place.

Abstract

As software security expert Bruce Schneier argues, the pervasive vulnerability of embedded systems today is structurally similar to the security crisis of PCs in the mid-1990s—only much worse. Embedded devices are ideal malware targets for several reasons. Firstly, Internet-connected devices are inherently more exposed to remote exploitation. Furthermore, embedded systems are notoriously difficult to update, regularly leading to unpatched vulnerabilities. Last but not least, many such devices operate in a mostly unattended fashion, which means that the timely discovery of compromise is unlikely.

Hardening is the process of securing a system by reducing its surface of vulnerability. Hardening of the already deployed embedded devices that are connected to the internet is examined in this research. A method capable of automatically generating and enforcing security configuration based on the embedded system's set of functions has been designed and implemented. The proposed system is dynamic, automatic, and seamless. Hardening level of such devices is measured through recognized security benchmarks. The objective is to harden the product as much as possible while maintaining its full functionality.

This study concerns embedded systems using custom Linux-distribution software. The thesis was conducted in cooperation with Atos. OpenScape Business series X (OSBiz X) was used as a case study. OSBiz X is an embedded system used as a telephony center, with more than 150.000 systems already deployed worldwide.

Implementing the system described above on OSBiz X significantly increased the hardening-level of the product while its functionality remained intact. Due to the case study's scenario results, Atos plans to integrate the proposed system into the next version of OSBiz X's official release. Finally, ongoing research about other internal organization's products that could greatly benefit from this approach is being conducted.

CONTENTS

List of Figures

List of Tables

Nomenclature

CIS	Center of Internet Security
cL	Custom Linux
CPU	Central processing unit
DDoS	Distributed Denial of Service
HW	Hardware
IoT	Internet of Things
NCP	National Checklist Program
OS	Operating System
OSBiz	Open Scape Business
QA	Quality Assurance
RAM	Random access memory
SDHC	Secure Digital High Capacity
SSH	Secure Shell
SW	Software
UC	Unified Communication
YP	Yocto Project

1 Introduction

1.1 Research Context

Embedded systems are usually a combination of computer hardware and software designed for a specific function, or they might operate within a more extensive system. Some particular examples of embedded systems include digital cameras, controllers for the operation of a car engine, chips within telecom switching equipment, control systems embedded in process plants, and many more. The importance of embedded systems is increasing mainly because of their capabilities, interconnectivity but also due to the growth of technology. The set created by all those embedded devices connected to the Internet is known as the Internet of Things (IoT). Cisco expects 500 billion devices connected to the Internet by 2030 [1]. Assuming a world population of 8.5 billion as predicted by the United Nations [2], this corresponds to 60 devices per person. IoT enables a data exchange that was never there before.

Technological advances that have spurred the development of these electronic systems resulted in more sophisticated security attacks. It has been observed that the cost of lack of security in electronic systems can be very high. The concept of security is often perceived by embedded system designers as the addition of features "such as specific cryptographic algorithms and security protocols" to the system [3]. Developers may face challenges, including the Original Design Manufacturer (ODM) supply chain, limited memory, small stack, and the problem of pushing firmware updates securely to an endpoin. Embedded Application Security is often not a high priority for embedded developers when they are producing devices such as routers, managed switches, medical devices, Industrial Control Systems (ICS), VoIP phones, IoT devices, and wearables due to other challenges outside of development [4].

In reality, embedded security is an entirely new metric that designers should consider throughout the design process, along with other parameters such as cost, performance, and power [5]. Deploying non-secure embedded devices can have disastrous consequences, as demonstrated by large-scale botnets, such as Reaper [6], Mirai, Hajime, and BrickerBot [7], [8]. Embedded devices are ideal malware targets for several reasons. Firstly, Internet-connected devices are inherently more exposed to remote exploitation. Furthermore, embedded systems are notoriously difficult to update, regularly leading to unpatched vulnerabilities. Last but not least, many such devices operate in a mostly unattended fashion, which means that the timely discovery of compromise is unlikely.

Moreover, the General Data Protection Regulation, a regulation that fundamentally reshapes how people handle data across every sector, directly affects embedded systems. The components that are relevant to the security of this regulation are [9]:

- Storage systems
- Wireless chips and wireless modules
- Embedded boards
- Microcontrollers
- Power supplies
- Sensors
- Special security modules

All the above are essential components that affect almost every embedded system. The motivation for increased security does not only target embedded systems that are going to be manufactured and deployed from now on, but also devices that are already deployed and need to comply with the users' needs and the formal regulations.

Interventions to limit the spread of embedded systems/IoT malware are required. One way to face malware that targets embedded/IoT systems is to make the devices self-aware in a way that they can defend themselves by adjusting their configuration. Since those systems have processing capabilities and Internet-connectivity, they also can generate a protective shell that can prevent them from future-attacks. There are many considerations to this practice since the processing, along with the storage of those devices, is usually constrained. Therefore, organizations owning embedded systems need to know at what extent embedded systems can protect themselves against external attacks.

1.2 Problem Statement

Security in Internet-connected embedded/IoT systems is evolving, and it has been evolving in recent years. This becomes clear when analyzing IoT research articles that explicitly mention the term "security" [10]. Regrettably, this evolution arrived too late as many of the already deployed IoT devices lack even the simplest form of security, resulting in numerous successful attacks on IoT devices in recent years.

It is essential also to note the popularity Linux distributions gained the recent years when it comes to embedded/IoT devices. This trend in favor of Linux is due to several benefits one might obtain using the specific Operating System. Linux is enhanced with a versatile, vastly customizable kernel where one can choose only the features needed. Additionally, Linux support almost any given hardware peripheral or layout. Another great benefit is the robust features encircled in Linux OS like portability, multitasking, multi-user support, multiprocessor support, virtual memory, filesystem and network protocol support and most importantly robust security features. On top of that, Linux is free and enjoys the acceptance of the OpenSource community. A survey conducted by the Eclipse IoT Working Group, AGILE IoT, IEEE, and the Open Mobile Alliance using a sample of 502 IoT developers; This survey revealed that 71.8% use Linux for their IoT devices [11]. However, this popularity is not entirely beneficiary, since it makes Linux an attractive target for miscreants.

Previous research focused primarily on how to design secure embedded systems. Those studies analyzed in-depth techniques that can be employed to provide embedded systems with extra resources and enable them to defend themselves. There are proposals on the decoupling of security and functional aspects in the process of designing the system [12]. Additionally, researchers have explored the concept of adding architectural support via the combination of the Trusted Platform Module (TPM), TrustZone, and reconfigurable Technology[13].

Furthermore, the traditional authentication techniques are not ideal for use in embedded devices since they incur significant overhead for authentication. This inefficacy has spurred research in identifying new ways for light-weight authentication schemes [14], [15]. Muhammad Tausif et al. analyzed and evaluated thirteen lightweight-cryptographic algorithms trying to investigate the relationships between the structural elements of an algorithm and its performance [16].

More recent studies focus on how to make Internet-connected devices more secure by exploring various concepts. Bruno Bogaz Zarpelão et al. created a review of 18 papers where Intrusion Detection Systems are utilized to provide security mechanisms in embedded/IoT devices. However, the results showed that these schemes were not mature yet [17]. Hiroshi Tsunoda et al. reflect on the societal model where parents protect human babies and infants to the IoT world by introducing designated Guardian devices. These

Guardian devices are essentially gateways acting as an intermediary between IoT objects and external entities [18].

A topic investigated from different angles throughout the literature is the employment of well-defined security configurations for Operating Systems as a method of protecting embedded systems. The significant advantage of this method is that it can be utilized in systems that have already been deployed in the field, in contrast with techniques that need to be implemented in the design phase of the product. Hardening the devices' OS configuration methods have been tested against botnets like the Mirai and also in enhancing the security in Android devices. Furthermore, large organizations like Cisco and Amazon employ management methods providing security to embedded/IoT devices. Nevertheless, these methods do not come without limitations. Most of the techniques developed to harden the OS configuration of a device either require manual user actions or deploy configuration policies defined by a System Admin to sets of devices based on their properties.

The research problem of this thesis is the automatic implant of security features into connected embedded systems in order to harden already deployed IoT devices.

1.3 Research Objective and Questions

The objective of this research is to explore the extent to which a system can make already deployed devices self-aware in a way that they can harden their configuration, thus becoming more secure. The proposed system should (1) utilize the device's processing and storage capabilities; (2) not break the device functionality nor accessibility; (3) be automated; (4) provide security over the product's life cycle. In this study, the desired outcome is that the proposed system (1) is designed and implemented, and (2) is utilized in a use-case.

To achieve the defined objective, the research question is formulated as follows:

How and to what extent can deployed-embedded systems, using custom Linux distributions, increase their own overall security by hardening their configuration without affecting their functionality?

To adequately answer the main research question, the following sub-questions should be answered.

SQ1. What mechanisms are being used to secure the configuration of an interconnected device's OS?

To make the configuration of an OS secure, it is of crucial importance to investigate the state-of-the-art technologies that are being currently employed in this direction.

SQ2. To what extent can a system-configuration of devices using Linux be considered secure? Is there a way to quantify this value, thus measuring the system's configuration security level?

Secure configuration implies a stable, relatively predictable environment in which a system can function without disruptions or fear of disturbance. It is of crucial importance to define what is considered secure in order to have a clear target. This question aims to investigate metrics to quantify the security level of a Linux configuration.

SQ3. What information is needed in order to generate a more secure configuration tailored to a specific custom Linux-based embedded system?

I need to determine the information that needs to be collected to generate a hardening shell tailored for each system. The whole methodology is dependent upon this information since every decision the hardening-system makes is dependent upon the data obtained from the embedded system itself.

SQ4. To what extent would the functionality of a system be affected after the hardening of its security?

The primary interest of the described methodology is to increase the security level of Linux's system configuration without affecting its functionality. Thus, it is vital to determine a way to check if the system's functionality is affected.

SQ5. How can we design and develop a prototype of configuration-hardening in systems using custom Linux Distributions?

This question investigates the feasibility of designing and developing a system with the characteristics described above.

SQ6. How does the prototype apply in an actual embedded system, and to what extent does the hardened system comply with its security, functionality, and accessibility specifications?

This sub-question addresses the interaction of the proposed methodology with a real-world use-case. The results obtained from the real-world use-case need to be analyzed in-depth to provide insight information related to the trade-off between security compliance/functionality/serviceability of the proposed methodology.

1.4 Research Approach

To answer the questions discussed previously, research is divided into three parts: (i) desk research, (ii) system design and development, and (iii) case-study. Each part is discussed in this section.

1.4.1 Desk research

During the desk research, an extensive literature review is conducted. The literature review starts with identifying the challenges faced currently by the interconnected devices. Next, the state of embedded systems' security is reviewed. Insights on what challenges concerning the interconnected devices have been addressed and how they are addressed are obtained through the first parts of the literature review. The literature review continues with the vulnerabilities and the attacks able to target IoT/embedded devices. The attacks most used in the real-world are further investigated. The state-of-the-art methodologies that have been developed to protect embedded/IoT devices are explored. Finally, the ways that have been employed to defend embedded/IoT devices already deployed on the field will be investigated (SQ1).

These steps are crucial to give an insight into what has already been implemented in both academic and industrial levels, enabling the possibility to develop a system that will take into consideration the limitations faced by the previously developed methodologies.

In short, the results of the desk research are: (1) an overview of the challenges faced by the embedded/IoT devices along with their security state; (2) the vulnerabilities of the devices and the attacks deployed against them; (3) on-the-field methods employed in protecting those devices.

1.4.2 System design and development

The proposed method is designed and developed by taking into consideration the results of the desk research phase and the research objectives. During the design & development phase, I formulate the proposed methodology of the thesis. First, how secure configuration is defined for Linux OS is determined. Then, the information that needs to be retrieved from a system regarding its configuration to decide whether or not it is secure or not must be obtained (SQ3). Additionally, information regarding a system's settings that are essential for its functionality must be acquired (SQ4). Designing a method to obtain the above information and, based on that, deciding what parameters should be changed to increase overall security, but considering the system's functionality, is next (SQ5). To properly develop the designed methodology, the appropriate tools, must be selected. The tools used are chosen based on the requirements described during the design of the proposed system. Next, the actual development of the proposed mechanism is implemented.

The results of this section are: (1) the design of the proposed mechanism; (2) the selection of the appropriate tools needed to develop the designed methodology; (3) the development of the proposed method.

1.4.3 Case-Study

A real-world embedded system will be utilized as a use-case. The developed methodology will be employed and tested in a real-world device. It is of crucial importance to get insight on how the proposed method behaves when deployed on an actual embedded device; that is because the results obtained will enable the accurate evaluation of the prototype developed during this research. Open Scape Business X will be used as the target device since it encircles every attribute considered in the Problem Statement section.

The results of the case-study will essentially be the results obtained from applying the prototype to the real-world use-case (SQ6). In order to retrieve the benefits of the proposed system –if any--, we will compare the security level of the device, its functionality, and its serviceability after and before the deployment of the prototype system to Open Scape Business X.

1.5 Scientific and Societal Relevance

From a scientific perspective, this thesis contributes to existing research in the field of security of interconnected embedded/IoT devices. Previous studies focus on ways to secure embedded/IoT systems by introducing security features or by adding extra resources; this aims to improve security during the design phase of the product. Additionally, research focused on protecting already deployed devices requires human intervention, while this study investigates how to enable embedded/IoT devices to harden themselves automatically.

This research has a direct societal impact as it is conducted within a European multinational IT company, Atos. Atos is a leading organization in digital transformation. Digital transformation is the integration of digital technology into all areas of a business resulting in fundamental changes to how companies operate and how they deliver value to customers. Embedded systems are the next big thing in the technology sector. Companies are searching for ways to utilize digital technologies to ample their competitive edge. Since, Atos' philosophy is to transform other entities and bring them up-to-speed with the latest technology, being able to do so with enterprises that produce embedded systems is one of Atos main objectives. Additionally, collaboration with Atos creates significant opportunities. Firstly, a successful product of Atos is used as a case-study providing vital information on how the proposed methodology functions in a real-world scenario. Second, Atos has customers looking for security solutions for their devices. Collaboration with Atos consists an excellent opportunity for the future of this research since the customer-base of Atos is directly interested in such a technology.

In societal terms, this research also contributes to mitigate cybercrime via embedded/IoT systems and enable them to affect society positively. Interconnected devices impact society through various channels. For example, smart cities can play a significant role in tackling climate change; intelligent health devices can improve the health care sector; IoT systems can improve water conservation and save critical species and provide even more benefits for society that have not yet been discovered. Nevertheless, if those devices are used maliciously, they can compromise all the sectors that benefit from them under normal circumstances. To ensure that this is not the case, generating effective methodologies that increase their security level is of crucial importance. Most importantly, since this research focuses on devices already deployed on the field, it promotes the sustainability of those devices and helps in their conformation with the latest security guidelines; in this way, the devices are able to continue functioning and providing their useful services without being a potential hazard.

1.6 Thesis Outline

The remainder of this thesis is structured as follows. Chapter 2 provides a literature review related to the topic. Essentially, it discusses background information and the security state on embedded systems/IoT, along with the prior work on the hardening of a device's Operating System configuration. Chapter 3 describes the design of the proposed prototype. Chapter 4 presents the selection and the usage of the tools utilized for the development of the prototype. Chapter 5 provides information related to the actual construction of the prototype's components. Chapter 6 describes the use-case where the prototype will be deployed. Chapter 7 describes the monitoring, hardening, and assessment procedures along with the results obtained from the case study. Last, Chapter 8 provides conclusions, discusses limitations, the quality of the research, and finally presents recommendations for future work.

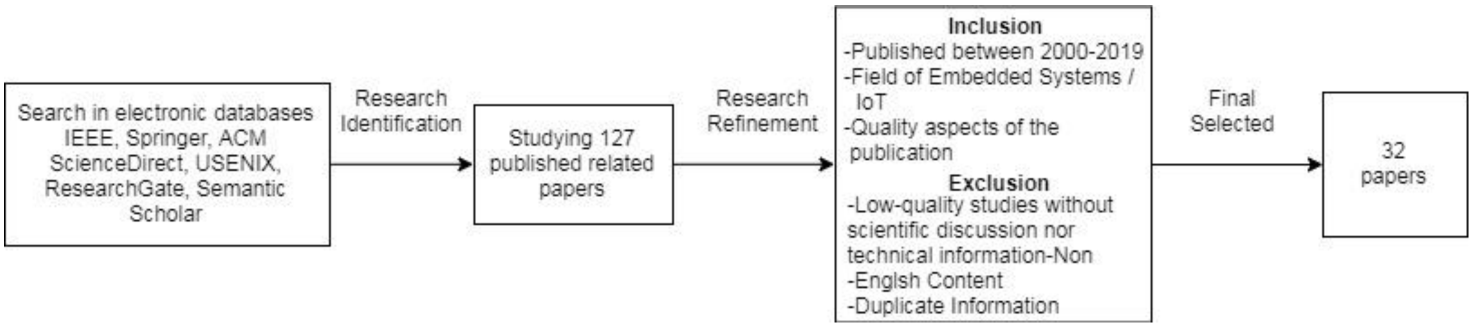
2. Literature Review

This chapter provides information on several topics related to this research. The purpose is to give some background on the inter-connected devices and their security state. Additionally, the previous related research that has already been conducted regarding methods employed to secure deployed on the field devices is investigated.

The structure of this Chapter goes as follows. First, in Section 2.1, the search strategy followed in this research is described. Next, the security challenges for the interconnected devices, along with the reason behind why these systems are more probable to be exploited in comparison with conventional computers, are described. In Section 2.3, the state of Embedded Systems' security is discussed. Following, the vulnerabilities of those systems, along with the attacks being utilized against them, are analyzed. Next, the most used attacks in the context of the real-world are presented. Last, ways to secure already deployed interconnected-systems are investigated. More specifically, ways to harden a device's Operating System are considered.

2.1 Search Strategy

This Section provides a review based on the Systematic Literature Review method (SLR) as a research study assessment for classifying the security challenges in interconnected devices, their vulnerabilities and attack vectors, and the solutions against them for already deployed systems. Taking into account the alternatives and other synonyms of the vital components, the following exploration string is formulated: ("Embedded Systems" OR "ES" OR "Deployed Embedded Systems" OR "Deployed ES") OR ("Internet of Things" OR "IoT" "Deployed Internet of Things" OR "Deployed IoT") AND ("Security") OR ("Attacks" OR "Vulnerabilities" OR "Defense" OR "Protection")



Search in electronic databases IEEE, Springer, ACM ScienceDirect, USENIX, ResearchGate, Semantic Scholar
Research Identification
Studying 127 published related papers
Research Refinement
Inclusion
-Published between 2000-2019
-Field of Embedded Systems / IoT
-Quality aspects of the publication
Exclusion
-Low-quality studies without scientific discussion nor technical information-Non
-Englsh Content
-Duplicate Information
Final Selected
32 papers

Figure 1 – Search Strategy

The Systematic Literature Review paper presents inclusive answers to questions related to the research. More specifically, the following Analytical Questions must be answered.

- What are the challenges faced by the Embedded/IoT devices that connect directly to the Internet?
- What is the current security state of those devices?
- Which are the vulnerabilities of the Embedded/IoT devices?
- What types of attacks are launched against them?
- With which ways are the currently deployed systems being defended?

Following the probing questions, the inclusion and exclusion criteria were applied to complete the research selection procedure. To effectively apply inclusion and exclusion criteria, the title, abstract, introduction, and conclusion sections of each paper were read. The procedure resulted in 32 papers that helped in the analysis and the answer to the analytical questions mentioned above.

Figure 2 presents the databases used, the number of the initially selected papers, the inclusion-exclusion criteria, and the number of the finally chosen papers for my research.

The inclusion principles considered are the following:

- Studies available online between 2000 and 2019.
- Works related to the field of Embedded and IoT devices.
- Researches using existing quality aspects.

The exclusion principles considered are the following:

- Works not written in the English Language.
- Studies contained very similar content with others.
- Low-quality studies without scientific discussion nor the technical information

Below, a figure depicting the distribution of the research papers by the publisher is attached. Most papers were retrieved from IEEE and during 2014-2017. This makes sense when considering that the most significant bridges in IoT devices, such as botnets attacking online services, occurred during this period.

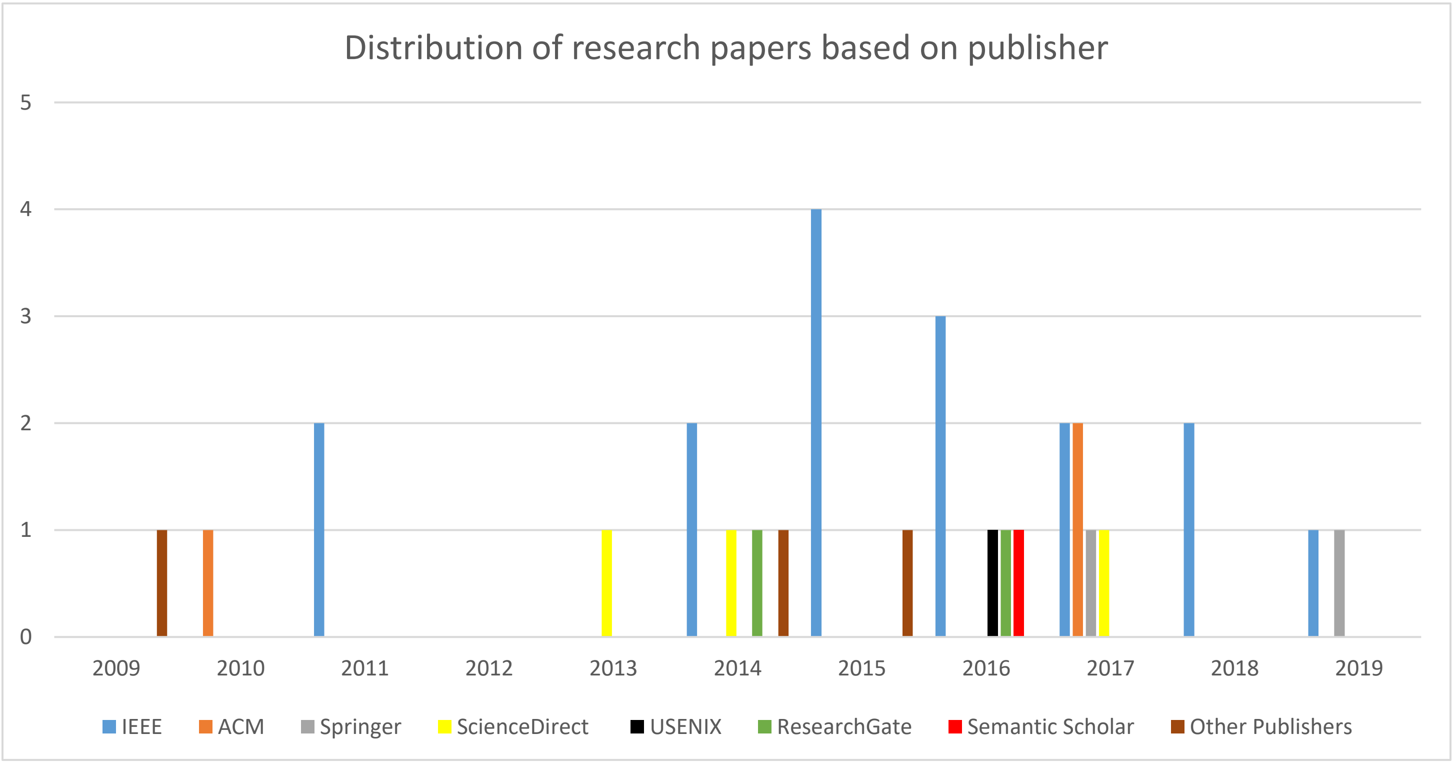


Figure 2 – Research Papers distribution over time-based on publisher

2.2 Security challenges for the interconnected devices

The biggest concern around embedded systems/IoT devices with network capabilities is their security aspect. Data stored or transferred via such devices is usually personal, enterprise, or consumer, making them sensitive. The Federal Trade Commission (FTC) report highlights the following potential security risks for interconnected devices that might get exploited to harm consumers. First, a lack of security can enable unauthorized access to intruders, who then can manipulate or tamper the data stored and transferred via the device. Second, security vulnerabilities on those devices might allow their compromise. When a group of those devices is compromised, attacks may be launched through them against targets connected to the Internet. Third, there is a possibility than when unauthorized persons get access to interconnected devices, and they might create risks to physical security [19]. A vivid example is the Jeep Cherokee hack executed by Charlie Miller & Chris Valasek, which could directly lead to physical harm [20].

Security challenges faced in interconnected embedded/IoT devices are notoriously harder to deal with than these of the standard PCs. The difficulty in anticipating these challenges is mainly because the focus of system engineering for embedded systems usually lies in resource-efficiency, cost reduction, and functionality. Non-functional properties, if considered in the development of embedded systems, are mainly concerned with safety or performance. Security issues are either neglected, added as an afterthought, or minimized due to cost or efficiency conditions in the development [21]. One important reason for this situation is that in the past, many embedded systems only had minimal connectivity and operated in controlled environments. Thus, security was usually sufficiently provided by the physical protection of the embedded devices. This situation has dramatically changed as the evolution of embedded systems has led them towards being devices connected via the Internet, wireless communication, or other interfaces as well.

2.3 State of Embedded Systems security

Before discussing how to defend embedded systems, we first need to consider the current state of embedded systems' security. With each passing day, embedded systems are getting smaller and smarter, enabling us to get more things done than before. As more functionalities are integrated into smaller device footprints, there is an upsurge in the security concerns as well. Device vendors tend to add new features that often crowd out the necessary security features, thus launching devices that underwent insufficient security testing.

The poor state that embedded systems security is currently at can be observed through the incidents that have occurred over the years. Attacks such as the 2007 TJX Security Breach and the 2009 Heartland Payment Systems resulted in millions of credit card numbers and other confidential consumer information to be compromised [22]. TJX breach occurred due to weak forms of security for wireless LANs that transmitted data between price-checking devices, cash registers, and computers at a store in Minnesota [23]. In 2011 "external intrusion" reported by Sony led to an estimate 170$ million cost as acknowledged by Sony executives [24]. In July 2015, a team of researchers was able to take total control of a Jeep SUV using the vehicle's CAN bus. By exploiting a firmware update vulnerability, they hijacked the car over the Sprint cellular network and discovered they could make it speed up, slow down, and even veer off the road [25]. A proof of concept for the emerging Internet of Things (IoT) hacks. The most exceptional example of the lack of security that characterizes IoT devices is the Mirai Botnet. An essential factor in the success of Mirai was the fact that most vendors do not follow security best practices in the IoT industry [7]. For instance, many devices are shipped with default passwords. This security flaw made it feasible to log in to hundreds of thousands of devices with a dictionary attack (using a small list of known default usernames and passwords). Furthermore, many IoT devices are shipped with some ports opened by default, accessible to anyone, even though that is unnecessary for the device to function.

2.4 Vulnerabilities in embedded/IoT devices

Vulnerabilities affecting devices directly connected to the Internet are evolving continuously. Knowing the weaknesses that may lie on a system is the first step towards finding ways to defend it. Thus, a taxonomy of the vulnerabilities of the IoT and, therefore, of the internet-connected devices with the same characteristics, such as embedded systems with network capabilities, is of essential importance. Nataliia Neshenko et al. have recently published an exhaustive survey on IoT vulnerabilities that contains an extensive taxonomy of the vulnerabilities one might find in IoT devices. The taxonomy examines IoT vulnerabilities within the scope of Layers, Security Impact, Attacks, Countermeasures, and Situational Awareness Capabilities [26].

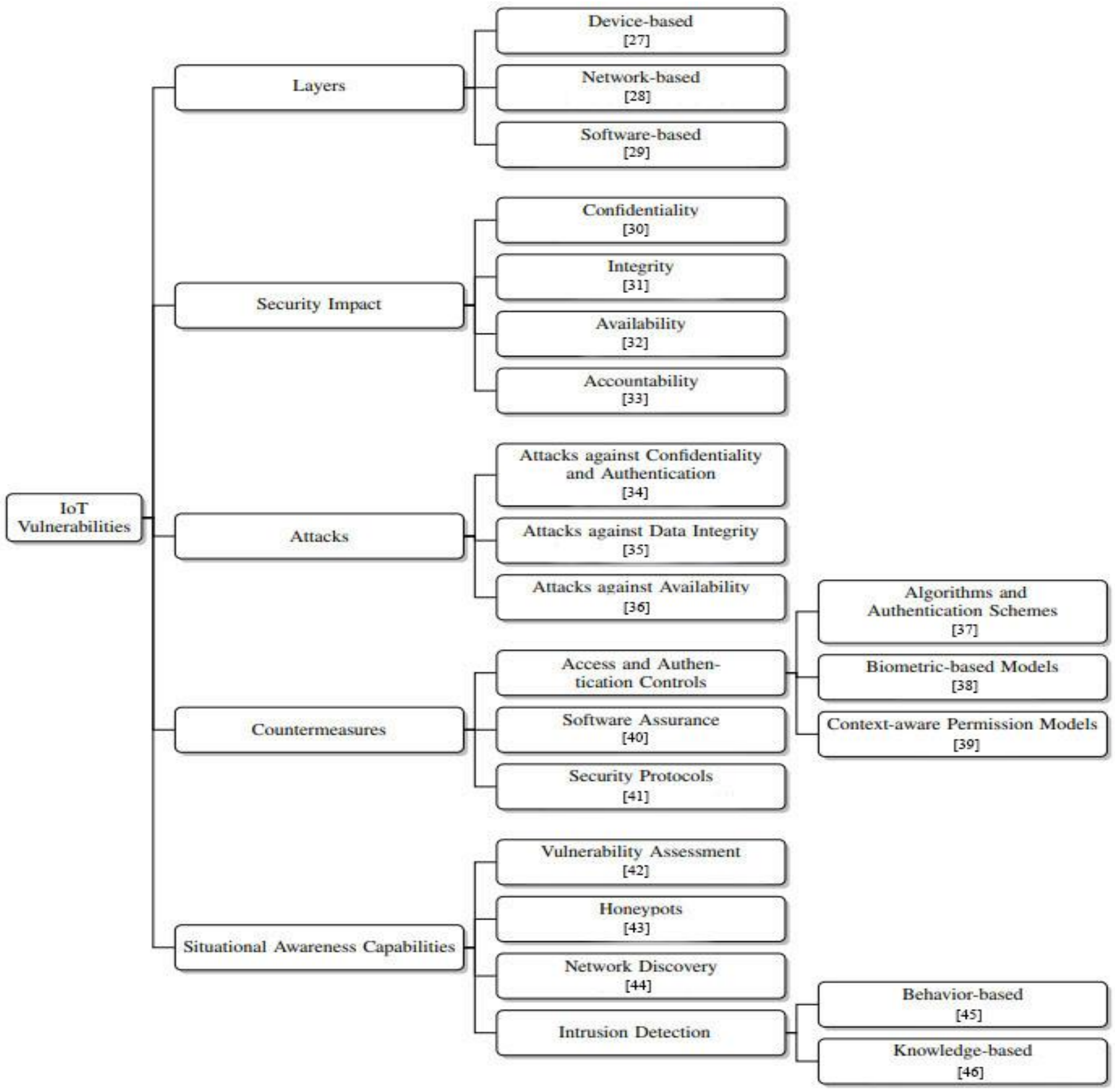

IoT Vulnerabilities
Layers
Device-based [27]
Network-based [28]
Software-based [29]
Security Impact
Confidentiality [30]
Integrity [31]
Availability [32]
Accountability [33]
Attacks
Attacks against Confidentiality and Authentication [34]
Attacks against Data Integrity [35]
Attacks against Availability [36]
Countermeasures
Access and Authen-tication Controls
Algorithms and Authentication Schemes [37]
Biometric-based Models [38]
Context-aware Permission Models [39]
Software Assurance [40]
Security Protocols [41]
Situational Awareness Capabilities
Vulnerability Assessment [42]
Honeypots [43]
Network Discovery [44]
Intrusion Detection
Behavior-based [45]
Knowledge-based [46]

Figure 3 – IoT Vulnerabilities [26]

- **Layers** examine the vulnerabilities associated with the components that comprise the actual device and its contents. Thus, layers can be further divided into three sub-classes. Device-based weaknesses are linked with the hardware elements of the device. Network-based are mainly due to communication protocols. Finally, Software-based refers to the vulnerabilities that occur due to the firmware or the software of the device.
- **Security Impact** classifies the vulnerabilities of those devices based on their impact on core security principles like confidentiality, integrity, availability, and accountability.
- **Attacks** are the third category describing the ways vulnerabilities might get exploited. These are the means to inflict the security impacts. More accurately, the three sub-categories of this category are attacks against confidentiality & authentication, attacks against data integrity, and attacks against availability.
- **Countermeasures** category classifies the remediation techniques utilized to mitigate the IoT vulnerabilities. Countermeasures are further divided into Access and Authentication Controls (firewalls, algorithms, authentication schemes, file-permissions), Software Assurance (methods to assert integrity constraints), Security protocols (lightweight protocols used in embedded/IoT devices).
- **Situation Awareness Capabilities** encircles the methods used to capture information related to the malicious activity of the device. Sub-classes contain Intrusion Detection Systems (IDS), Honeypots, Network Discovery techniques, and Vulnerability Assessment.

2.5 Attacks in Embedded Systems

To protect any system, the first thing one should do is to analyze the potential attacks that could be deployed to the system. So here, possible attacks, explicitly targeting embedded systems are listed.

Possible attacks: Attacks against ES fall under two major categories. The first one is Physical and Side-Channel Attacks, while the second is Logical Attacks. The first category refers to the situation where the attacker has direct access to the ES using a physical connection. These kinds of attacks can either be invasive (the attacker can access these devices directly using a physical link) or non-invasive (attacks that do not need to access the system from the inside). The second category is about attacks where the attacker tries to penetrate the software or the cryptographic security system. Below, the categories mentioned above will be expanded [47] [48].

Physical & Side-Channel Attacks (invasive)

- **Micro-probing:** Analyzing the integrated circuits using microscopes or even special equipment known as microprobing station.
- **Reverse Engineering:** The reproduction of the ES product following a detailed examination of its construction and composition, to simulate and learn its functions.

Physical & Side-Channel Attacks (non-invasive)

- **Timing analysis:** A try to violate the cryptosystem by analyzing the time it takes to execute computations.
- **Power analysis:** Attackers gather information by measuring the energy consumed. It can be implemented in two ways, simple power analysis (SPA) and differential power analysis (DPA).
- **Electromagnetic analysis:** The attacker can detect information by measuring the electromagnetic radiation emitted by the system.

- **Fault injection:** There are different ways in which a fault can be induced in a microcontroller. Methods used to implement fault injection could be a particle accelerator, a laser or light. A more invasive injection is when someone inserts a glitch directly on the power supply pin. That way, the processor might misinterpret an instruction or a variable.

Logical Attacks

- **Code Injection:** The exploitation of a computer bug that is caused by processing invalid data. The injection is used by an attacker to introduce (or "inject") code into a vulnerable computer program and change the course of execution. The result of a successful code injection can be disastrous.
- **Software Exploitation and Buffer Overflows:** In software exploitation, attack a chunk of data, or a sequence of commands take advantage of the vulnerability to cause unintended behavior to computer software or hardware. Typically, it is a flaw in the programming.
- **Crypto and protocol Weaknesses:** Using deprecated transport layer protocols such as TLS1.0, which is known to be vulnerable to attacks such as BEAST and POODLE, is a highly severe security threat. A well-equipped attacker can easily extract vital information from the ES using this weakness.
- **Control Hijacking:** Execute arbitrary code on target by hijacking application control flow.

The above analysis is being displayed comprehensively in the below figure.

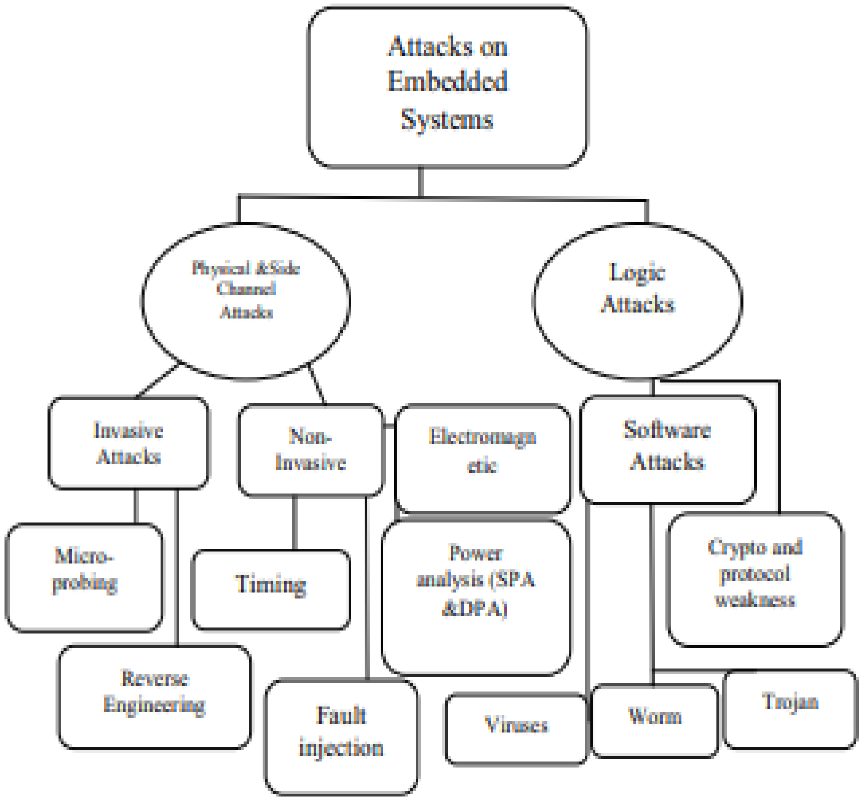


Figure 4 – Attacks on Embedded Systems Taxonomy [48]

2.6 Most used attacks and vulnerabilities in Embedded Systems

The above analysis considers every possible attack that can be launched against an embedded system. Supplementary to this information, a systematic review of the existing threats and vulnerabilities in embedded systems based on publicly available data has been conducted by Dorottya Papp, Zhendong Ma, and Levente Buttyan. This survey focuses on two sets of data, the exposures of attacks on embedded systems in security conferences and literature, and the published vulnerabilities specific to embedded systems. Based on the data, an attack taxonomy to systematically identify and classify frequent attacks against embedded systems is derived. This taxonomy has a particular weight as it reflects the attacks that happen against Embedded Systems that are functioning in the field. Furthermore, the sequences of these types of attacks are also included in this survey. Below the figure depicting the results of this review is illustrated.

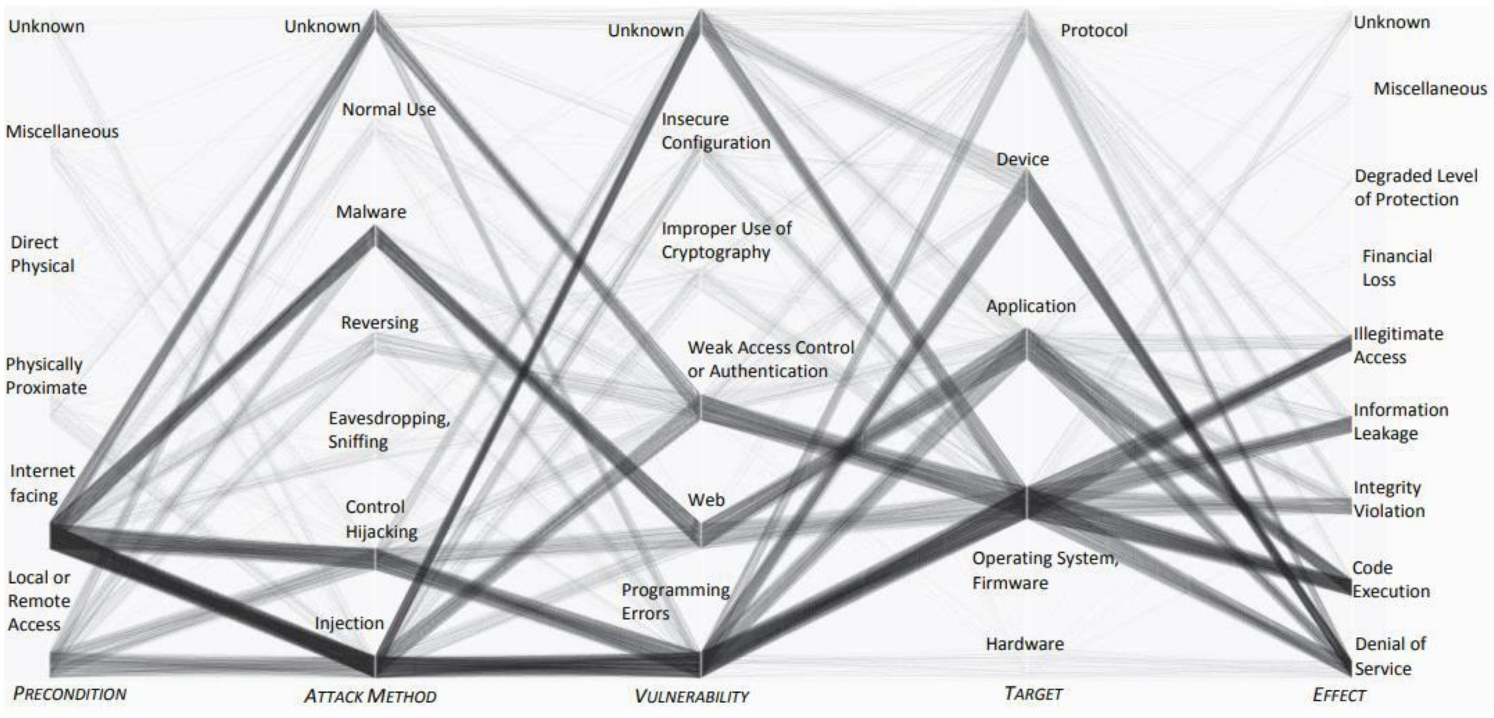


Figure 5 – Most used attacks and vulnerabilities in ES [49]

The above attack taxonomy provides information on how embedded systems are being attacked and the results of these attacks. Moreover, structured knowledge can assist the analysis and design of systems, including or based on embedded devices during the system's development lifecycle.

2.7 Security enhancement on deployed-systems via Configuration Management

This section provides insight into the ways Configuration Management has been utilized to face security threats in embedded/IoT systems. Configuration Management highly depends on the Operating System running on the target device. In the past years, several methodologies tried to utilize the OS Configuration to anticipate security vulnerabilities in embedded/IoT systems effectively.

Adjusting configuration of the OS in critical control systems to enhance their security is a topic of interest for many years. J. Holcomb proposed a method in which, via connecting to a system, it is possible to audit the system's OS configuration and generate a report after the audit, displaying the status of each audit check as well as a summary. [50]

Cheng-Liang Kuo and Chung-Huang Yang investigated a security design for configuration management specifically for Android devices. Android devices mostly are Linux-based embedded systems. Their main objective is to make an Android device conform with Google Android OS Benchmark issued by CIS (Center of Internet Security). They developed an application through which the Android user can scan his security configurations online and generate a report with the security's compliance score while she/he can also check and repair her/his phone's configuration to increase its security level. One weak point of this approach is that the final decision of repairing or not a configuration option is upon the end-user. Usually, end-users do not have the appropriate background to decide whether a configuration option would affect their user-interaction with the device if changed [51].

Boheung Chung, Jeongyeo Kim, and Youngsung Jeon in their published work, late in 2016, proposed a methodology that provides an on-demand security configuration to set or change the security functionalities of a device. Their methodology uses security profiles from other devices that have similar hardware to configure the functional security modules and then generate a device image transferred to the target device via its update process or transmit them directly. The limitations of this method are that it assumes that the device does have an update mechanism, and it needs the system to reboot for the configuration changes to apply [52].

3 Prototype Design

The goal of this research is to enhance the way of defense for deployed embedded systems. A key element to properly defend an embedded system is to generate a system profile accurately. To do so, a monitor application is utilized. There are numerous monitor applications/tools for Linux, such as command-line tools (top, Htop, monit), network monitoring tools (ntopng, ngrep, tcpdump), and Linux server monitoring tools (Linux Dash, ps, glances). However, these tools are made to fulfill a different purpose. Thus, the development of an application to accurately monitor and properly log just the system's elements that are of interest for the system's defense was necessary.

In this chapter, the design, functionality, and way of deployment of three packages are presented. Those packages are a monitor package, cL-Monitoring, a hardening package, cL-Hardening, and an assessment package, cL-Assessment, where cL stands for custom-Linux.

The first one, cL-Monitoring, is used to monitor the system and create its system-profile. cL-Hardening utilizes as input the profile generated by the cL-Monitoring and generates a hardening configuration script along with a script that can revert these hardening configuration settings. The last one, cL-Assessment, must check the system's configuration generating a percentage of compliance with a given check-list or benchmark that will include how OS's configurations be adjusted in order for them to be considered secure.

The following requirements are defined:

Requirement 1

The monitoring package should collect information to track the embedded system's vital points that are important for its defense along with system-specific information that changes between custom-Linux distributions. This information will be used to generate detailed system-maps.

Requirement 2

During the monitoring process, the system should be fully functional. The duration of the monitoring phase depends solely on the functionality of the embedded system that will be examined.

Requirement 3

Hardening package must be able to apply the desired hardening to the embedded systems that it will be deployed without affecting the system's functionality and performance. It should also provide the means to revert the applied configurations.

Requirement 4

A report should be generated indicating which policies are adjusted and which remain un-adjusted based on a security check-list that will be utilized as a baseline. An overall compliance score against the same security check-list should also be provided.

Requirement 5

In the generic case, constrictions due to the agnostic nature of the embedded systems apply. These constrictions are primarily related to three dimensions. First, the tools that will be used to implement the proposed methodology need to be available at as most Linux Distributions as possible. Second, the size of the recommended packages needs to be minimized as the available size of the system is unknown. Third, the CPU utilization must be kept low since otherwise, it might interfere with the system's normal function.

From the requirements, the functions that the proposed system needs to perform become clear. Those functions are depicted in the Figure below. Functions illustrated below are described in more detail in the remainder of this chapter.

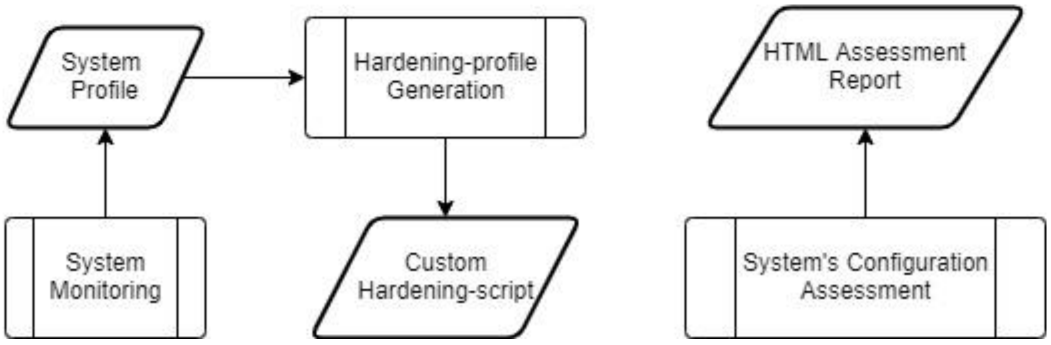


Figure 6 – Proposed system design

3.1 Secure configuration for Linux OS

OS configuration settings are OS elements that can be altered via the software itself. OS configuration settings are affecting the security of the system directly. An example of a Linux OS setting is allowing VFAT filesystems to be mounted on it.

Almost no system, if any, is 100% secure. Configuration settings can vastly improve a system's security, but this often comes at the expense of reducing its functionality. Additionally, there are security settings that might have a positive and negative impact on the system's security at the same time. For example, setting the lockout time for any failed authentication attempt to 1 hour would be very secure against brute force attacks, but at the same time it could affect negatively legitimate users by locking them out for mistyping a password, and it could also comprise a potential denial-of-service attack vector against users.

To increase a device's security level without disturbing its functionality via adopting the OS's configuration, one must consider how each alteration in the configuration settings can affect the device's functionality. The process of increasing a device's security via adopting its configuration settings is known as system hardening.

3.2 System-specific information collection

Creating a custom-Linux distribution implies that the Operating System is designed to cover particular needs. Every custom-Linux can differ from another on four major elements:

1. The bootloader. Even though boot-loaders tend to have tiny size foot-print and are relatively simple, they play a crucial role in the boot process. The bootloader is responsible for loading the Linux kernel with optional kernel parameters and the Linux initial RAM disk. Numerous bootloaders can be used along with a Linux, so detecting the one utilized is very important.
2. System architecture. Different configurations might be applied between 32bit and 64bit architectures.
3. System initialization. Nowadays, two system initialization techniques are mainly utilized for Linux based systems, SysV, and systemd. Knowing which one is being used by the system is crucial to configure it.
4. Package manager. Detecting which package manager is being used by the custom Linux-Distro can play a significant role when packages are to be added or removed from the system.

Additionally, each of these distributions can have a unique set of configurations. The ultimate goal is to tailor the hardening to be perfect for each system separately. Thus, to properly harden the configuration of these systems, first, they need to get mapped in full detail.

3.3 cL-Monitoring (design)

To create an appropriately tailored system-map, a monitoring package will be designed and implemented. The purpose of this package is to continuously monitor the device's vital points while the device is functioning in real-conditions. The main functionality of the monitoring package is to record which system's attributes are being utilized. Once an adequate number of samples has been collected, a file containing the system's profile is generated.

For the monitoring procedure, several constrictions apply. Since the examined system, in the generic case, will be unknown, the tools that will be utilized to implement the monitoring procedure must be selected wisely. That is the reason monitoring will rely mostly on tools like grep, find, iptables and modprobe, which are all very basic for any Linux Distribution and will examine Linux specific attributes and files that exist in all systems such as dmesg, modules.builtin and configuration files located mostly in pre-defined paths. Additionally, the checks that cL-Monitoring conducts should be smart, in a sense that the monitoring process should be able to tackle small variations that may exist from system to system.

cL-Monitoring should essentially be a mechanism able to monitor the system's vital functionalities during its operation. To effectively do so, bash-scripts responsible for the system's monitoring that are executed periodically and asynchronously, depending on the checks they conduct, are employed. The bash-scripts mentioned above, must extract the necessary information and record it in a structured way. The way information is recorded plays a significant role since it enables the automatic parsing of the records and the extraction of crucial conclusions through them.

3.4 cL-Hardening (design)

The cL-Hardening package will be responsible for generating the desired hardening for the target embedded system. The hardening of the system is directly related to the system's secure configuration and its functionalities.

Hardening package must be able to generate a hardening script tailored for the specific device that has been previously monitored. The tailored hardening script must be eligible for the monitored device and all other devices of the same kind.

Hardening should be able to be applied in real-time, do not disrupt the device's functionality at all, and do not affect the services provided by the device after its enforcement. To generate the cL- Hardening package, the system map generated by cL-Monitoring will be used as an input.

3.5 cL-Assessment (design)

cL-Assessment should investigate the system and generate a report, including a score indicating the security level of the system. To define the security level of a system, a security check-list/benchmark should be used as a baseline to decide whether a specific configuration is considered secure or not. Additionally, cL-Assessment must be able to provide its results in an easily readable format that will present each configuration separately and inform the reader about each system's configuration security status. Last, an overall compliance score against the security check-list/benchmark must also be included in the generated report.

4 Tools

4.1 Image/Package development Tool Selection

The main objective of this section is to select the appropriate tools that will allow the development of the packages described in the Prototype Design section and provide a reference point for the security measures that should be enforced into any custom Linux based distribution.

Since the proposed approach in this thesis aims to target any custom Linux distribution, the available options for the image/package development solution to be utilized are limited. An investigation amongst the two most mainstream image/package development solutions for embedded Linux distributions was conducted. These options are Yocto and Buildroot. The central aspect that makes Yocto and Buildroot build-systems suitable for my research is their ability to cross-compile and build a full Linux system for a chosen target device. Thus, the proposed packages can be built for any custom Linux distro by just adjusting the target device parameters in the build system's environment.

4.1.1 Yocto

Yocto can create custom Linux distributions tailored for the needs of the target device. The Yocto Project is an open-source collaboration project able to support developers create custom Linux based systems tailored for embedded/IoT devices regardless of the product's hardware architecture. Yocto provides a flexible toolset and a development environment, including shared technologies, configurations, software stacks, and best practices enabling embedded device developers to create custom Linux distributions adjusted to their needs.

Yocto products:

- Root filesystem image, Kernel, and Bootloader tailored based on a device's characteristics.
- Cross-development toolchains able to build images that run on the target hardware
- Package feeds. Package feeds are an intermediary step in the build procedure. The Yocto Project provides classes to generate different package types (rpm, ipk or deb)

Advantages	**Disadvantages**
Widely adopted across the industry	Steep learning curve
Architecture agnostic	Project workflow is difficult to grasp
Package management system	Resource-intensive
Highly customizable and expandable	Unfamiliar environment for non-embedded system developers
Minimal native tooling required	
Comprehensive toolchain capabilities	
Shared State Cache allowing incremental builds	
Build from scratch, from source	
Widely used in the industry	
Well documented, training courses	

Table 1 – Yocto Project advantages and disadvantages

4.1.2 Buildroot

Buildroot is a tool created to automate and simplify the process of building a complete custom Linux distribution for an embedded system via cross-compilation.

Buildroot products:

- Boot images designed for the target embedded device
- ''Firmware Generator'': it has no binary packages nor a package management system
- A compilation toolchain. Practically, a set of tools that allows the code compilation for the target system.

Advantages	Disadvantages
Quick to get started & easy to grasp	Configuration changes require a complete rebuild
Active developer community	Resource-intensive
Broad architecture support	Unfamiliar environment for non-embedded system developers
Comprehensive Documentation	Updates are not possible via packages
Strong focus on simplicity	Updates require a full system update
Open community	
Build from scratch from source	
Widely used in the industry	
Well documented, training courses	

Table 2 – Buildroot pros advantages and disadvantages

4.1.3 Yocto vs. Buildroot

Yocto and Buildroot are both able to provide the same end product: a root filesystem image, a bootloader, a kernel, and a compatible toolchain for an Embedded system. Nevertheless, differences are making these projects unique. The table below describes the significant differences between the two build-systems.

Yocto	Buildroot
Yocto focuses on being versatile and supports a wide range of embedded systems	Buildroot focuses on simplicity
Yocto's output is "a distribution." It provides a root FS image, individual packages, and a Software Development Kit.	Buildroot's output is a root FS image. That is why it is called a "firmware generator."
Yocto keeps configuration information in multiple parts.	Buildroot puts all configuration information in one file.
Yocto has a bit more challenging of a learning curve. The leading utility, BitBake, is a 60,000-line Python program, and recipes are written in a combination of Python, shell scripts, and a BitBake-specific language.	Buildroot strives to remain simple; Petazzoni said every change proposed for the core is analyzed in terms of its "usefulness to complexity ratio."

Yocto utilizes layers to enable a strict separation between the core of the build, the board support package (BSP), and custom modifications. Thus, it allows a configuration to scale properly	Buildroot avoids the usage of collections of layers for its build mostly because they tend to be case-specific

Table 3 – Yocto vs. Buildroot

4.1.4 Selection

Based on the characteristics of the two build-systems analyzed above, I chose to proceed with one of them. The choice was based on the needs of the thesis and the differences of the build-systems. The criteria on which the decision was based are described below:

1. It is highly preferred to be able to deploy the packages described in the Prototype Design section directly into the system over replacing the whole system's image to include extra packages.
2. The challenging learning curve is seen as positive in the context of the thesis project since one of the purposes of the project is to explore in-depth new technologies.

Based on the build-systems' properties and the needs of the thesis project, Yocto was selected as the build-system tool to be utilized for the thesis project.

4.2 Yocto

To create the necessary packages for the implementation of the proposed system, the Yocto Project will be utilized. Yocto can create custom Linux distributions and build packages for the already deployed devices. It is an open-source collaboration project that provides templates, tools, and methods to help in the creation of custom Linux-based systems for embedded products, regardless of their hardware architecture. Moreover, the Yocto Project provides a flexible toolset and a development environment that allows embedded device developers across the world to collaborate through shared technologies, software stacks, configurations, and best practices used to create these tailored Linux images. Yocto Project has the aim and objective of attempting to improve the lives of developers of customized Linux systems supporting the ARM, MIPS, PowerPC, and x86/x86-64 architectures.

The general project idea is that one can use either Upstream Software Projects or local ones, apply custom configuration and policies to them, and then via Yocto's tools, produce output packages of these sources which are used to produce an overall Software Image along with a Software Development Kit.

Glossary of Terms and Acronyms:

• **OpenEmbedded**: The build system architecture promoted by the Yocto Project. During a build process, the build system tracks dependencies and performs a native or cross-compilation of the package. As the first step in a cross-build setup, the framework attempts to create a cross-compiler toolchain (i.e., SDK) suited for the target platform.

• **Build System - "Bitbake":** a scheduler and execution engine which parses instructions (recipes) and configuration data. It then creates a dependency tree to order the compilation, schedules the compilation of the included code, and finally executes the building of the specified, custom Linux image (distribution). BitBake is a make-like build tool. BitBake recipes specify how a particular package is built. They include all the package dependencies, source code locations, configuration, compilation, build, install, and remove instructions. Recipes also store the metadata for the package in standard variables. Related recipes are consolidated into a layer. During the build, process dependencies are tracked, and native or cross-compilation of the package is performed.

• **Metadata:** A vital element of the Yocto Project is the meta-data, which is used to construct a Linux distribution, contained in the files that the build system parses when building an image. In general, Metadata includes recipes, configuration files, and other information referring to the build instructions themselves, as well as the data used to control what things get built and to affect how they are built. The meta-data also includes commands and data used to indicate what versions of the software are used, and where they are obtained from, as well as changes or additions to the software itself (patches or auxiliary files) which are used to fix bugs or customize the software for use in a particular situation. OpenEmbedded Core is an essential set of validated metadata.

• **Recipe:** The most common form of metadata. A recipe will contain a list of settings and tasks (instructions) for building packages, which are then used to build the binary image. A recipe describes where you get source code and which patches to apply. Recipes describe dependencies for libraries or other recipes, as well as configuration and compilation options. Recipes are stored in layers.

• **Layer:** A collection of related recipes. Layers allow you to consolidate related metadata to customize your build and isolate information for multiple architectures builds. Layers are hierarchical in their ability to override previous specifications. You can include any number of available layers from the Yocto Project and customize the build by adding your layers after them. The Layer Index is searchable for layers within the Yocto Project.

• **Classes:** Class files (.bbclass) contain information that is useful to share between recipe files.

• **Configuration Files:** Files that hold global definitions of variables, user-defined variables, and hardware configuration information. They tell the build system what to build and put into the image to support a particular platform.

• **Packages:** The output of the build system used to create your final image.

The OpenEmbedded build system uses a "workflow" to accomplish image and SDK generation.

The following figure overviews that workflow.

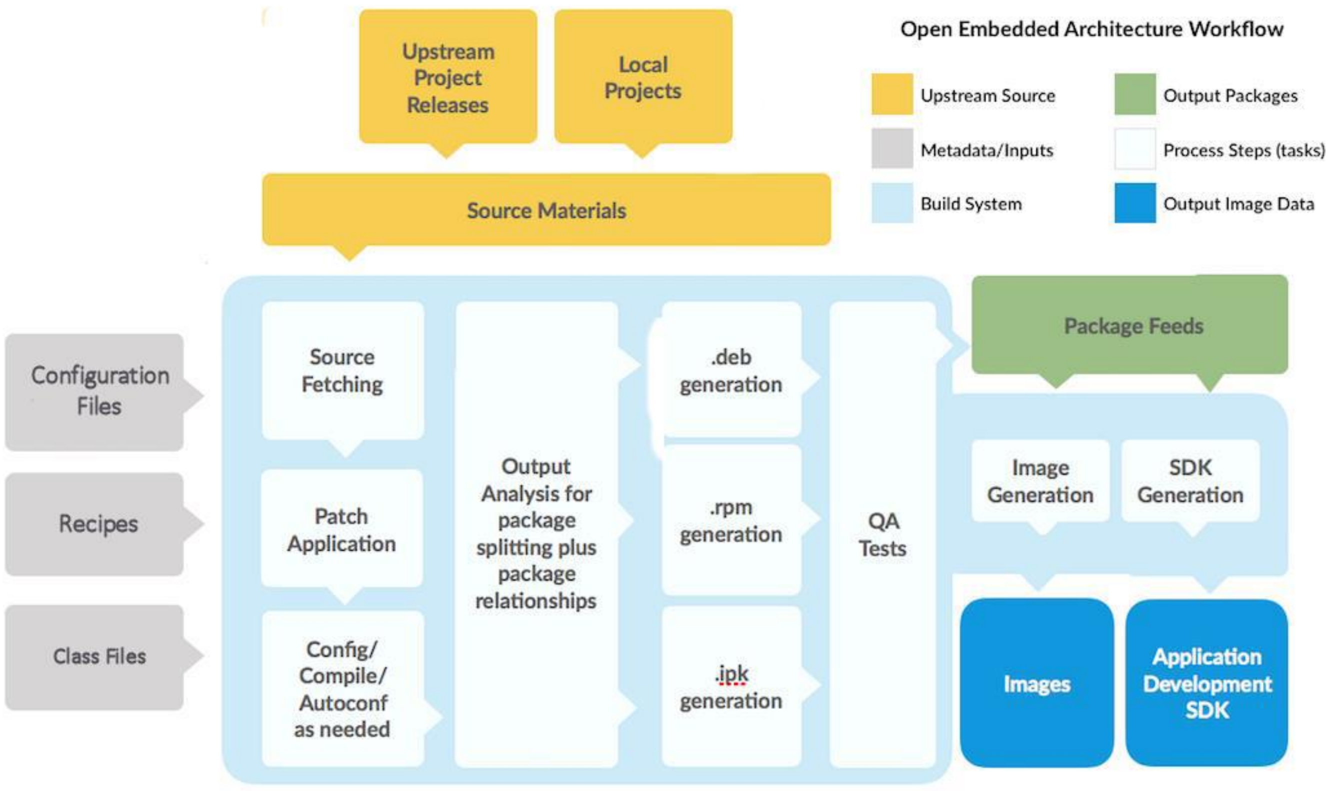


Figure 7 - The OpenEmbedded build system “workflow.”

Yocto Features:

- **Widely Adopted Across the Industry**
- **Architecture Agnostic**: Yocto Project supports Intel, ARM, MIPS, AMD, PPC, and other architectures. Most ODMs, OSVs, and chip vendors create and supply BSPs that support their hardware.
- **Flexibility**: Through customization and layering, a project group can leverage the base Linux distribution to create a distribution that works for their product needs.
- **Images and Code Transfer Easily:** Yocto Project output can quickly move between architectures without moving to new development environments.
- **Suitable for Constrained Embedded devices:** Unlike a full Linux distribution, you can use the Yocto Project to create precisely what you need for embedded devices. You only add the feature support or packages that you need for the device.
- **Comprehensive Toolchain Capabilities:** Toolchains for supported architectures satisfy most use-cases. However, if your hardware supports features that are not part of a standard toolchain, you can easily customize that toolchain through the specification of platform-specific tuning parameters.

• **Uses a Layer Model:** The Yocto Project layer infrastructure, groups the related functionality into separate bundles. You can incrementally add these grouped functionalities to your project as needed.

• **Supports Partial Builds:** You can build and rebuild individual packages as needed.

• **Releases According to a Strict Schedule:** Major releases occur on a six-month cycle predictably in October and April.

• **Rich Ecosystem of Individuals and Organizations:** For open source projects, the value of community is critical. Support forums, expertise, and active developers who continue to push the Yocto Project forward are readily available.

• **License Manifest:** The Yocto Project provides a license manifest for review by people who need to track the use of open-source licenses (e.g. legal teams).

Challenges

- **Steep Learning Curve:** The Yocto Project has a steep learning curve and has many different ways to accomplish similar tasks. It can be challenging to choose how to proceed when varying methods exist by which to achieve a given task.
- **Understanding What Changes You Need to Make For Your Design Requires Research:** Beyond the simple tutorial stage, understanding what changes need to be made for your particular design can require a significant amount of research and investigation.
- **Working in a Cross-Build Environment Can Feel Unfamiliar**: When developing code to run on a target, compilation, execution, and testing done on the actual target can be faster than running a BitBake build on a development host and then deploying binaries to the target for a test.
- **Initial Build Times are Significant:** Long initial build times are unfortunately unavoidable due to a large number of packages initially built from scratch for a fully functioning Linux system. Once that initial build is completed, however, the shared-state (state) cache mechanism Yocto Project uses keeps the system from rebuilding packages that have not been "touched" since the last build.

To utilize the Yocto, a transition period has to take place. During this period, the production process will be transformed. The final goal is to build the overall image of the organization’s custom-Linux distribution through Yocto. It is a complicated procedure due to YP’s steep learning curve, but once it is accomplished, then the benefits, YP, makes it worth the effort [53].

4.3 Security Checklist/Benchmark Selection

To adequately assess the security level of the custom-based Linux distributions that the target systems are using, a security configuration checklist (also called a lockdown, hardening guide, or benchmark) will be utilized. A configuration checklist is essentially a series of instructions or procedures for configuring an IT product to an operational environment. Furthermore, it verifies that the product has been configured correctly and can identify unauthorized changes to the product.

To identify which security checklist fits best the examined case of this research, National Checklist Program Repository is used. The National Checklist Program (NCP), defined by the NIST SP 800-70, is the U.S. government repository of publicly available security checklists (or benchmarks) that provide detailed low-level guidance on setting the security configuration of operating systems and applications.

NCP includes security checklists issued by various recognized third parties, software vendors, and governmental authorities. These authorities are Brohium, Center of Internet Security (CIS), CyberESI, Defense Information Systems Agency, Department of Defense, HP, Intel, Kyocera Mita America Inc, Microsoft Corporation, MITRE, National Security Agency, NIST National Vulnerability Database, NIST Computer Security Division, Red Hat, Technology Infrastructure Subcommittee (TIS), and Vanguard Integrity Professionals Inc.

To satisfy the needs of this research, a checklist tailored for custom-Linux distributions was the target I was looking for. Unfortunately, such a list was not included in the National Checklist Program Repository (https://nvd.nist.gov/ncp/repository). Nevertheless, such a security checklist was found at one of the authorities that are included in NCP's repository. CIS issues such a Benchmark titled 'CIS Independent Linux Benchmark.' It fits the needs of the research perfectly, and as there was no other security checklist issued by a well-acknowledged authority targeting custom-Linux distributions, CIS was selected as the authority to validate the security level of the embedded/IoT devices this research targets [54].

4.4 CIS-CAT

CIS – Center for Internet Security

The Center for Internet Security (CIS) is a 501(c)(3) nonprofit organization formed in October 2000. CIS is dedicated to enhancing the cybersecurity readiness and response among public and private sector entities. Utilizing its strong industry and government partnerships, CIS combats evolving cybersecurity challenges on a global scale and helps organizations adopt key best practices to achieve immediate and effective defenses against cyber-attacks.

CIS employs a closed crowdsourcing model to identify and refine security measures, with individuals developing recommendations that are shared with the community for evaluation through a consensus decision-making process. At the national and international level, CIS plays a vital role in forming security policies and decisions by maintaining the CIS Controls and CIS Benchmarks and hosting the Multi-State Information Sharing and Analysis Center (MS-ISAC).

CIS Controls and CIS

Benchmarks CIS Controls and CIS Benchmarks provide global standards for internet security and are a recognized worldwide standard and best practices for securing IT systems and data against attacks. CIS maintains "The CIS Controls," a popular set of 20 security controls, "which map to many compliance standards." Those security tools are also applicable to the Internet of things and embedded systems. Through an independent consensus process, CIS Benchmarks provide frameworks to help organizations bolster their security.

4.4.1 CIS Independent Linux Benchmark

Especially, for custom Linux Distributions, there is a specific Benchmark (CIS Independent Linux Benchmark), which provides prescriptive guidance for establishing a secure configuration posture for Linux systems running on x86 and x64 platforms. This well accepted by the Internet Security community benchmark will be used to assess the custom Linux distribution's security level.

The benchmark was created using a consensus review process comprised of subject matter experts. Consensus participants provide perspective from a diverse set of backgrounds, including consulting, software development, audit and compliance, security research, operations, government, and legal [55].

CIS-benchmark offers the possibility to evaluate servers or workstations. For now, it will be used in the Embedded Systems workstations category since it fits better. The profiles that have been developed for Linux Workstations are the following [56]:

Profile Definitions
Level 1 - Workstation
Items in this profile intend to:
1. be practical and prudent
2. provide a clear security benefit
3. not inhibit the utility of the technology beyond acceptable means

Level 2 - Workstation - This profile extends the "Level 1 - Workstation" profile.
Items in this profile exhibit one or more of the following characteristics:
1. are intended for environments or use-cases where security is paramount
2. acts as a defense in depth measure
3. may negatively inhibit the utility or performance of the technology

The Benchmark is divided into sections. Each section contains recommendations aiming at OS hardening. These recommendations are grouped, so each section refers to a specific aspect of the Operating System. Below there is a summary of the benchmark's sections.

1 - Initial Setup
- File System Configuration
- Configure Software Updates
- File System Integrity
- Secure Booting Settings
- Process Hardening

2 – Services
- inetd
- Special Purpose Services
- Service Clients

3 – Network Configuration
- Network Parameters Host/Router
- IPv6
- TCP Wrappers
- Firewall Configuration

4 – Logging and Auditing
- auditd
- Configure Logging

5 - Access, Authentication and Authorization
- Configure cron
- SSH Server Configuration
- PAM
- User Accounts

6 – System Maintenance

- System File Permissions
- User & Group Settings

CIS's benchmark is also utilized as an assessment tool on a Linux System. A score can be generated, indicating the percentage of the system's compliance with the benchmark. A passing score is based on the organization's requirements and policies. Successfully implementing every security setting is considered unrealistic for most of the already deployed products as these settings interfere with products' functionality. After a CIS report is produced and all applicable security recommendations have been implemented according to the organization's requirements, it is recommended to include an exception report to document the justification as to why some recommendations were not applied.

This is the tool that will be used to assess the embedded system's security policies as it can be used for every system using a custom Linux distribution. More specifically, the Benchmark that will be used is CIS Distribution Independent Linux version 2.0.0 issued on the 16th of July 2019, and it is illustrated below. A table explaining the colors used to highlight each Benchmark's entry is attached before the benchmark.

4.4.2 Investigation of the proposed hardening configurations

Enforcing every hardening configuration proposed by CIS's benchmark for Distribution Independent Linux will almost definitely break the device's functionality since some of the system's configurations must remain intact for the system to continue its proper function.

Thus, the benchmark must be analyzed to determine which hardening configurations can be applied to the examined device without affecting its functionality.

There are three scenarios for every hardening configuration suggestion.

- The impact on the system's functionality for applying the hardening configuration can be neglected as it most certainly will not affect the system's functionality. Thus, the specific hardening configuration can be applied to every system.
- The impact on the system's functionality for applying the hardening configuration might heavily depend on the Linux system's functionality. Thus, it is required to investigate the system in order to determine whether the specific configuration ca be applied to it or not.
- The impact on the system's functionality for applying the hardening configuration cannot be determined and it is possible that it could potentially harm the system's functionality. Thus, the specific hardening configuration must be prohibited for every system.

To determine at which category every hardening configuration proposed by CIS falls under, every configuration must be examined separately. The Benchmark is divided into sections and every section contains grouped recommendations that refer to a specific aspect of the Operating System. Hardening configurations that belong to the same section might have some common characteristics that enable their examination via a common way.

Initially, I conducted an investigation of each section and sub-section included in the CIS benchmark to categorize every hardening configuration proposed by CIS based on the impact it could have on the functionality of a Linux system. The investigation is described in detail in the below table:

Section	Investigation Result
1.Initial Setup	**Hardening configurations proposed in this section highly depend on every system's functionality. Thus, system's investigation is required in most cases to determine whether the hardening of a specific configuration might affect the system's functionality.**
1.1 - Filesystems Configuration	Every Linux OS may utilize different filesystems. It is possible to determine what filesystems are essential for the system's function and eliminate the rest of them to reduce the attack surface Re-partitioning a live system might result to undesired behavior; thus configurations related to dynamic partitions are prohibited. Nevertheless, if the separate partitions do already exist; their special options (e.g. noexec, nosuid) might be enforced to further increase system's protection depending on whether the system works as expected with and without them
1.2 - Configure SW Updates	This section is not scored. However, configuring package manager repositories for custom Linux distributions highly depends on the way the system was designed. So, the mechanism cannot easily predict which upstream repository fits each distribution, if any
1.3 - Filesystem Integrity Checking	IDS are generally intense resource consuming programs. Thus, the installation of an IDS or not highly depends on the system's specifications and needs. Due to the variations that may be found between systems, the installations/configurations proposed in this section are prohibited
1.4 - Secure Boot Settings	Properly configuring bootloader's permissions and setting a password in order to prevent unauthorized edits to the bootloader's configuration can be considered safe to be applied to any Linux system
1.5 - Additional Process Hardening	Every process must be investigated and if it is not used by the system it should be hardened. If it cannot be determined whether the system utilizes or not a specific process, then its hardening should be prohibited
1.6 - Mandatory Access Control	Due to the fact that both SELinux and AppArmor can degrade the system's performance their installation is prohibited. However, if they are already installed their proper configuration can safely be applied based on the CIS's recommendations
1.7 - Warning Banners	Warning banners do not affect system's functionality thus their proper configuration can safely be applied to all custom Linux systems
2.Services	**Investigation must be conducted for every service and if it is being utilized by the system then it should remain as it is otherwise the remediation suggested by CIS must be followed.**
2.1 - inetd Services	
2.2 - Special Purpose Services	
2.3 - Service Clients	
3.Network Configuration	**Investigation to determine which network parameters (kernel parameters, access list control, firewall settings, network protocols) are being utilized by the system must be conducted. Based on the result of the investigation the respective network parameters should be hardened if they are not essential for the system's function or remain intact otherwise. Network files' related permissions can safely be hardened. Also, packets logging for various reasons as suggested by CIS is also considered safe for the system in terms of not affecting its functionality.**

3.1 - Network Parameters [Host]	
3.2 - Network Parameter [Host-Router]	
3.3 - TCP Wrappers	
3.4 - Uncommon Network Protocols	
3.5 - Firewall Configuration	
4.Logging & Auditing	**Most of the configurations under this section are related to data collection and they vastly depend on whether there is available disk space in the system to do so.**
4.1- Configure auditd	If there is enough available disk space on the system then these configurations can safely be adjusted otherwise, they should not change
4.2 - Configure Logging	Depending on the available space investigation rsyslog might be installed. Proper configurations of journald and rsyslog can safely be applied. Nevertheless, a remote log host cannot be configured automatically
4.3 – Ensure logrotate is configured	Not scored. Even though it is a very important aspect of the logging procedure since it might enable the collection of more information while it the same time it might also decrease the space reserved for logging purposes. Logrotate's further investigation and proper configuration depending on the system must be conducted
5.Access, Authentication & Authorization	**The impact of adjusting most of the configurations under this section cannot be determined via investigation so they are primarily divided into those who can safely be applied to any system and to those who are prohibited to be applied to any system.**
5.1 - Configure Cron	Configurations related to file permissions can safely be applied
5.2 - SSH Server Configuration	SSH configurations related to file permissions can safely be applied. SSH configurations that might affect the system's interaction with other systems and the behavior of the system due to those adjustments cannot be verified; thus they must be prohibited
5.3 - Configure PAM	Not scored section
5.4 - User Accounts and Environment	Configurations that might alter the system's behavior even on the long term must be prohibited
5.5 - Ensure Root Login is restricted	Not scored section
5.6 – Ensure su command is restricted	Access to su command must be restricted in any case. Nevertheless, enforcing a root password to a system which has none even it is an essential protection measure it might affect the way it interacts with other systems. Thus, its alternation is prohibited
6.System Maintenance	**The impact of adjusting most of the configurations under this section cannot be determined via investigation so they are primarily divided into those who can safely be applied to any system and to those who are prohibited to be applied to any system.**
6.1 - System File Permissions	Properly adjusting file permissions' configurations can safely be applied
6.2 - User and Group Settings	The configurations in this sub-section are related to local users and groups. Applying the suggested remediations or not highly depends on the nature of the remediation so the individual check of each remediation is suggested

Table 4 – CIS's benchmark configuration settings investigation

Taking into consideration:

1. The investigation included in the above table

2. Careful examination-analysis of each policy separately
3. Experience that was obtained after every hardening configuration and its impact was tested with a custom based Linux system

Every hardening configuration, included in CIS's benchmark, was assigned a color. Each color indicates if the respective hardening configuration can be applied or not to every Linux system and under which constrictions. The colors that are selected to indicate the status of each policy suggested by CIS are green, gold, red and cyan.

Green indicates that the specific configuration policy can safely be applied for all target embedded/IoT devices. The respective configuration hardens the system without affecting its functionality, performance, or serviceability. This category is fundamental since it contains all configurations that can safely be applied to a system and increase its security level without the risk of disrupting the service it provides. This category contains mainly configurations that have to do with the access permissions on Linux specific files such as *passwd* or *shadow*, proper warning banners 'configuration, and SSH configurations that ensure that basic good-practices such as maximum login attempts.

Gold color indicates the policies that require for the device to be investigated in order to determine whether it is safe for its functionality to harden the specific configurations. This is the most exciting category since it varies from device to device. Essentially, the proposed mechanism monitors the system to determine if the configurations under this category can be hardened without affecting the system's functionality or not. This category contains the majority of the policies suggested by CIS for custom Linux distributions.

Policies whose impact cannot be determined or if hardened will most probably affect the system's functionality are marked with red. This category contains policies that the proposed mechanism is not able to decide whether the respective policies would improve the system's security if hardened without affecting the system's functionality. For example, the decision to disable the system when audit logs are full or not is a decision that should be made from the system admin since it heavily depends on whether it is essential for the system to function without logging capabilities or not. Also, configurations that require changes in the partition scheme of the system (e.g., *Ensure separate partitions exists for /var/tmp*) are also marked with red since the hardening is applied in real-time and performing such actions is not recommended for live systems.

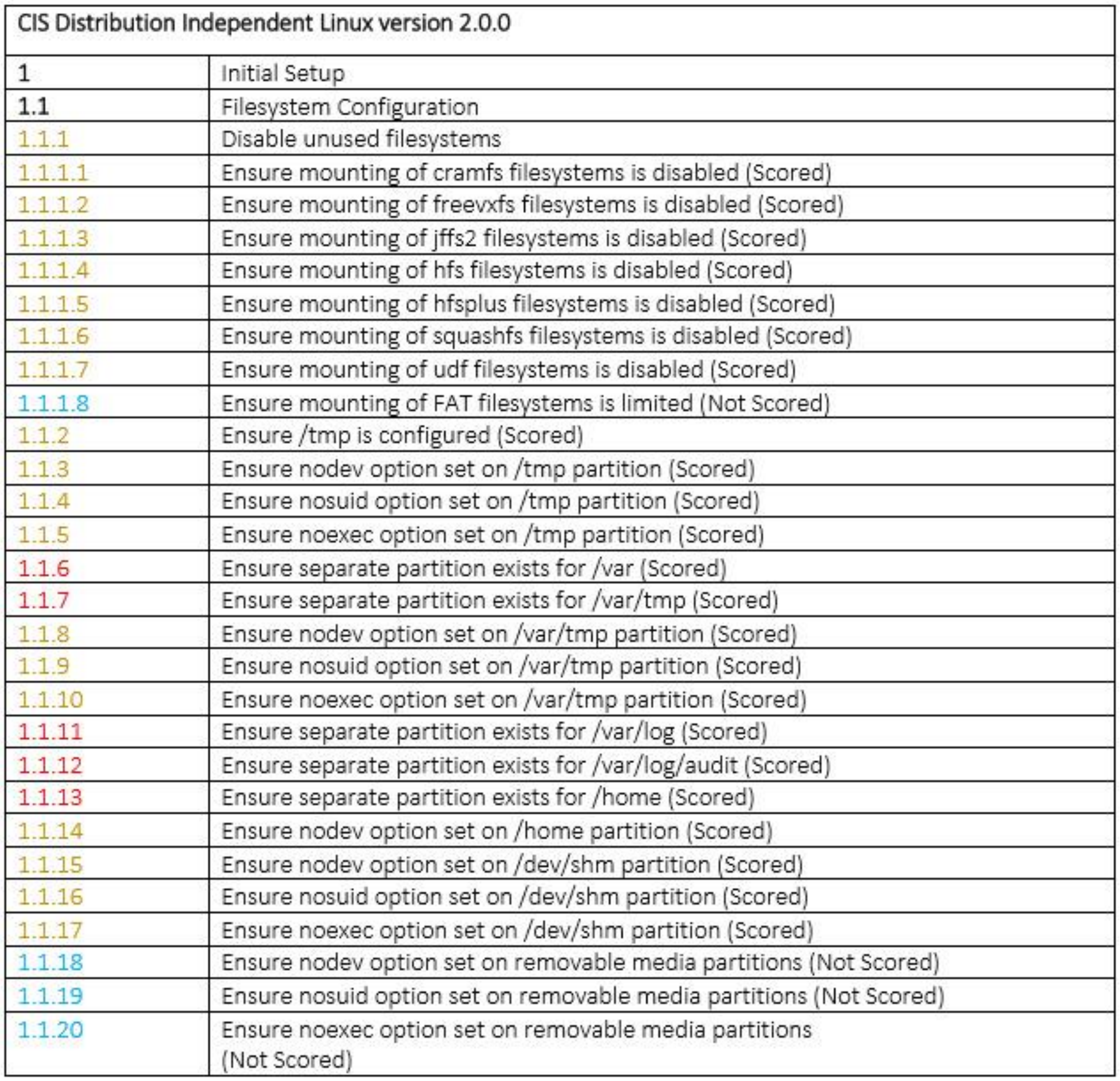
Finally, with cyan policies that are not scored are marked. These policies are not taken into account for the proposed mechanism.

A table containing the meaning of each color assigned to every policy proposed by CIS is demonstrated below:

Color	Meaning
🟩	Apply of this policy is considered safe for all systems
🟨	Apply of this policy needs to be investigated for the specific system
🟥	Apply of this policy is prohibited for all systems
🟦	The policy is not scored

Table 5 – Benchmark memorandum

The results of the above described exploration procedure are concluded in the table below:



CIS Distribution Independent Linux version 2.0.0	
1	Initial Setup
1.1	Filesystem Configuration
1.1.1	Disable unused filesystems
1.1.1.1	Ensure mounting of cramfs filesystems is disabled (Scored)
1.1.1.2	Ensure mounting of freevxfs filesystems is disabled (Scored)
1.1.1.3	Ensure mounting of jffs2 filesystems is disabled (Scored)
1.1.1.4	Ensure mounting of hfs filesystems is disabled (Scored)
1.1.1.5	Ensure mounting of hfsplus filesystems is disabled (Scored)
1.1.1.6	Ensure mounting of squashfs filesystems is disabled (Scored)
1.1.1.7	Ensure mounting of udf filesystems is disabled (Scored)
1.1.1.8	Ensure mounting of FAT filesystems is limited (Not Scored)
1.1.2	Ensure /tmp is configured (Scored)
1.1.3	Ensure nodev option set on /tmp partition (Scored)
1.1.4	Ensure nosuid option set on /tmp partition (Scored)
1.1.5	Ensure noexec option set on /tmp partition (Scored)
1.1.6	Ensure separate partition exists for /var (Scored)
1.1.7	Ensure separate partition exists for /var/tmp (Scored)
1.1.8	Ensure nodev option set on /var/tmp partition (Scored)
1.1.9	Ensure nosuid option set on /var/tmp partition (Scored)
1.1.10	Ensure noexec option set on /var/tmp partition (Scored)
1.1.11	Ensure separate partition exists for /var/log (Scored)
1.1.12	Ensure separate partition exists for /var/log/audit (Scored)
1.1.13	Ensure separate partition exists for /home (Scored)
1.1.14	Ensure nodev option set on /home partition (Scored)
1.1.15	Ensure nodev option set on /dev/shm partition (Scored)
1.1.16	Ensure nosuid option set on /dev/shm partition (Scored)
1.1.17	Ensure noexec option set on /dev/shm partition (Scored)
1.1.18	Ensure nodev option set on removable media partitions (Not Scored)
1.1.19	Ensure nosuid option set on removable media partitions (Not Scored)
1.1.20	Ensure noexec option set on removable media partitions (Not Scored)

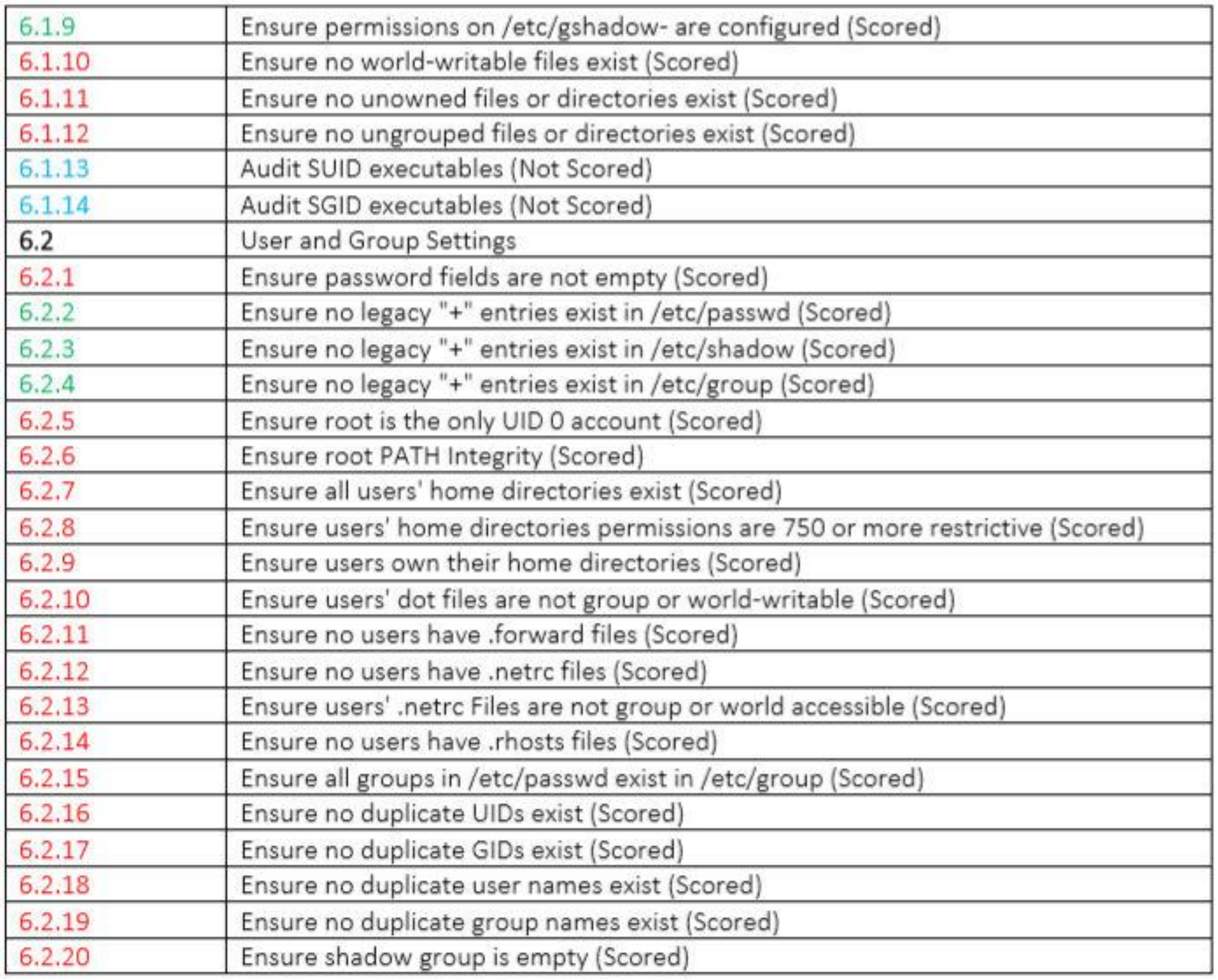
1.1.21	Ensure sticky bit is set on all world-writable directories (Scored)
1.1.22	Disable Automounting (Scored)
1.1.23	Disable USB Storage (Scored)
1.2	Configure Software Updates
1.2.1	Ensure package manager repositories are configured (Not Scored)
1.2.2	Ensure GPG keys are configured (Not Scored)
1.3	Filesystem Integrity Checking
1.3.1	Ensure AIDE is installed (Scored)
1.3.2	Ensure filesystem integrity is regularly checked (Scored)
1.4	Secure Boot Settings
1.4.1	Ensure permissions on bootloader config are configured (Scored)
1.4.2	Ensure bootloader password is set (Scored)
1.4.3	Ensure authentication required for single user mode (Scored)
1.4.4	Ensure interactive boot is not enabled (Not Scored)
1.5	Additional Process Hardening
1.5.1	Ensure core dumps are restricted (Scored)
1.5.2	Ensure XD/NX support is enabled (Scored)
1.5.3	Ensure address space layout randomization (ASLR) is enabled (Scored)
1.5.4	Ensure prelink is disabled (Scored)
1.6	Mandatory Access Control
1.6.1	Ensure Mandatory Access Control Software is Installed
1.6.1.1	Ensure SELinux or AppArmor are installed (Scored)
1.6.2	Configure SELinux
1.6.2.1	Ensure SELinux is not disabled in bootloader configuration (Scored)
1.6.2.2	Ensure the SELinux state is enforcing (Scored)
1.6.2.3	Ensure SELinux policy is configured (Scored)
1.6.2.4	Ensure SETroubleshoot is not installed (Scored)
1.6.2.5	Ensure the MCS Translation Service (mcstrans) is not installed (Scored)
1.6.2.6	Ensure no unconfined daemons exist (Scored)
1.6.3	Configure AppArmor
1.6.3.1	Ensure AppArmor is not disabled in bootloader configuration (Scored)
1.6.3.2	Ensure all AppArmor Profiles are enforcing (Scored)
1.7	Warning Banners
1.7.1	Command Line Warning Banners
1.7.1.1	Ensure message of the day is configured properly (Scored)
1.7.1.2	Ensure local login warning banner is configured properly (Scored)
1.7.1.3	Ensure remote login warning banner is configured properly (Scored)
1.7.1.4	Ensure permissions on /etc/motd are configured (Scored)
1.7.1.5	Ensure permissions on /etc/issue are configured (Scored)
1.7.1.6	Ensure permissions on /etc/issue.net are configured (Scored)
1.7.2	Ensure GDM login banner is configured (Scored)
1.8	Ensure updates, patches, and additional security software are installed (Not Scored)
2	Services
2.1	inetd Services
2.1.1	Ensure chargen services are not enabled (Scored)
2.1.2	Ensure daytime services are not enabled (Scored)

2.1.3	Ensure discard services are not enabled (Scored)
2.1.4	Ensure echo services are not enabled (Scored)
2.1.5	Ensure time services are not enabled (Scored)
2.1.6	Ensure rsh server is not enabled (Scored)
2.1.7	Ensure talk server is not enabled (Scored)
2.1.8	Ensure telnet server is not enabled (Scored)
2.1.9	Ensure tftp server is not enabled (Scored)
2.1.10	Ensure xinetd is not enabled (Scored)
2.2	Special Purpose Services
2.2.1	Time Synchronization
2.2.1.1	Ensure time synchronization is in use (Not Scored)
2.2.1.2	Ensure ntp is configured (Scored)
2.2.1.3	Ensure chrony is configured (Scored)
2.2.1.4	Ensure systemd-timesyncd is configured (Scored)
2.2.2	Ensure X Window System is not installed (Scored)
2.2.3	Ensure Avahi Server is not enabled (Scored)
2.2.4	Ensure CUPS is not enabled (Scored)
2.2.5	Ensure DHCP server is not enabled (Scored)
2.2.6	Ensure LDAP server is not enabled (Scored)
2.2.7	Ensure NFS and RPC are not enabled (Scored)
2.2.8	Ensure DNS Server is not enabled (Scored)
2.2.9	Ensure FTP Server is not enabled (Scored)
2.2.10	Ensure HTTP server is not enabled (Scored)
2.2.11	Ensure IMAP and POP3 server is not enabled (Scored)
2.2.12	Ensure Samba is not enabled (Scored)
2.2.13	Ensure HTTP Proxy Server is not enabled (Scored)
2.2.14	Ensure SNMP Server is not enabled (Scored)
2.2.15	Ensure mail transfer agent is configured for local-only mode (Scored)
2.2.16	Ensure rsync service is not enabled (Scored)
2.2.17	Ensure NIS Server is not enabled (Scored)
2.3	Service Clients
2.3.1	Ensure NIS Client is not installed (Scored)
2.3.2	Ensure rsh client is not installed (Scored)
2.3.3	Ensure talk client is not installed (Scored)
2.3.4	Ensure telnet client is not installed (Scored)
2.3.5	Ensure LDAP client is not installed (Scored)
3	Network Configuration
3.1	Network Parameters (Host-Only)
3.1.1	Ensure IP forwarding is disabled (Scored)
3.1.2	Ensure packet redirect sending is disabled (Scored)
3.2	Network Parameters (Host and Router)
3.2.1	Ensure source-routed packets are not accepted (Scored)
3.2.2	Ensure ICMP redirects are not accepted (Scored)
3.2.3	Ensure secure ICMP redirects are not accepted (Scored)
3.2.4	Ensure suspicious packets are logged (Scored)
3.2.5	Ensure broadcast ICMP requests are ignored (Scored)

3.2.6	Ensure bogus ICMP responses are ignored (Scored)
3.2.7	Ensure Reverse Path Filtering is enabled (Scored)
3.2.8	Ensure TCP SYN Cookies is enabled (Scored)
3.2.9	Ensure IPv6 router advertisements are not accepted (Scored)
3.3	TCP Wrappers
3.3.1	Ensure TCP Wrappers is installed (Not Scored)
3.3.2	Ensure /etc/hosts.allow is configured (Not Scored)
3.3.3	Ensure /etc/hosts.deny is configured (Not Scored)
3.3.4	Ensure permissions on /etc/hosts.allow are configured (Scored)
3.3.5	Ensure permissions on /etc/hosts.deny are configured (Scored)
3.4	Uncommon Network Protocols
3.4.1	Ensure DCCP is disabled (Scored)
3.4.2	Ensure SCTP is disabled (Scored)
3.4.3	Ensure RDS is disabled (Scored)
3.4.4	Ensure TIPC is disabled (Scored)
3.5	Firewall Configuration
3.5.1	Configure IPv6 ip6tables
3.5.1.1	Ensure IPv6 default-deny firewall policy (Scored)
3.5.1.2	Ensure IPv6 loopback traffic is configured (Scored)
3.5.1.3	Ensure IPv6 outbound and established connections are configured (Not Scored)
3.5.1.4	Ensure IPv6 firewall rules exist for all open ports (Not Scored)
3.5.2	Configure IPv4 iptables
3.5.2.1	Ensure default-deny firewall policy (Scored)
3.5.2.2	Ensure loopback traffic is configured (Scored)
3.5.2.3	Ensure outbound and established connections are configured (Not Scored)
3.5.2.4	Ensure firewall rules exist for all open ports (Scored)
3.5.3	Ensure iptables is installed (Scored)
3.6	Ensure wireless interfaces are disabled (Not Scored)
3.7	Disable IPv6 (Not Scored)
4	Logging and Auditing
4.1	Configure System Accounting (auditd)
4.1.1	Configure Data Retention
4.1.1.1	Ensure audit log storage size is configured (Scored)
4.1.1.2	Ensure the system is disabled when audit logs are full (Scored)
4.1.1.3	Ensure audit logs are not automatically deleted (Scored)
4.1.2	Ensure auditd is installed (Scored)
4.1.3	Ensure auditd service is enabled (Scored)
4.1.4	Ensure auditing for processes that start prior to auditd is enabled (Scored)
4.1.5	Ensure events that modify date and time information are collected (Scored)
4.1.6	Ensure events that modify user/group information are collected (Scored)
4.1.7	Ensure events that modify the system's network environment are collected (Scored)
4.1.8	Ensure events that modify the system's Mandatory Access Controls are collected (Scored)
4.1.9	Ensure login and logout events are collected (Scored)
4.1.10	Ensure session initiation information is collected (Scored)

4.1.11	Ensure discretionary access control permission modification events are collected (Scored)
4.1.12	Ensure unsuccessful unauthorized file access attempts are collected (Scored)
4.1.13	Ensure use of privileged commands is collected (Scored)
4.1.14	Ensure successful file system mounts are collected (Scored)
4.1.15	Ensure file deletion events by users are collected (Scored)
4.1.16	Ensure changes to system administration scope (sudoers) is collected (Scored)
4.1.17	Ensure system administrator actions (sudolog) are collected (Scored)
4.1.18	Ensure kernel module loading and unloading is collected (Scored)
4.1.19	Ensure the audit configuration is immutable (Scored)
4.2	Configure Logging
4.2.1	Configure rsyslog
4.2.1.1	Ensure rsyslog is installed (Scored)
4.2.1.2	Ensure rsyslog is installed (Scored)
4.2.1.3	Ensure logging is configured (Not Scored)
4.2.1.4	Ensure rsyslog default file permissions configured (Scored)
4.2.1.5	Ensure rsyslog is configured to send logs to a remote log host (Scored)
4.2.1.6	Ensure remote rsyslog messages are only accepted on designated log hosts. (Not Scored)
4.2.2	Configure journald
4.2.2.1	Ensure journald is configured to send logs to rsyslog (Scored)
4.2.2.2	Ensure journald is configured to compress large log files (Scored)
4.2.2.3	Ensure journald is configured to write log files to persistent disk (Scored)
4.2.3	Ensure permissions on all logfiles are configured (Scored)
4.3	Ensure logrotate is configured (Not Scored)
5	Access, Authentication, and Authorization
5.1	Configure cron
5.1.1	Ensure cron daemon is enabled (Scored)
5.1.2	Ensure permissions on /etc/crontab are configured (Scored)
5.1.3	Ensure permissions on /etc/cron.hourly are configured (Scored)
5.1.4	Ensure permissions on /etc/cron.daily are configured (Scored)
5.1.5	Ensure permissions on /etc/cron.weekly are configured (Scored)
5.1.6	Ensure permissions on /etc/cron.monthly are configured (Scored)
5.1.7	Ensure permissions on /etc/cron.d are configured (Scored)
5.1.8	Ensure at/cron is restricted to authorized users (Scored)
5.2	SSH Server Configuration
5.2.1	Ensure permissions on /etc/ssh/sshd_config are configured (Scored)
5.2.2	Ensure permissions on SSH private host key files are configured (Scored)
5.2.3	Ensure permissions on SSH public host key files are configured (Scored)
5.2.4	Ensure SSH Protocol is set to 2 (Scored)
5.2.5	Ensure SSH LogLevel is appropriate (Scored)
5.2.6	Ensure SSH X11 forwarding is disabled (Scored)
5.2.7	Ensure SSH MaxAuthTries is set to 4 or less (Scored)
5.2.8	Ensure SSH IgnoreRhosts is enabled (Scored)
5.2.9	Ensure SSH HostbasedAuthentication is disabled (Scored)

5.2.10	Ensure SSH root login is disabled (Scored)
5.2.11	Ensure SSH PermitEmptyPasswords is disabled (Scored)
5.2.12	Ensure SSH PermitUserEnvironment is disabled (Scored)
5.2.13	Ensure only strong Ciphers are used (Scored)
5.2.14	Ensure only strong MAC algorithms are used (Scored)
5.2.15	Ensure only strong Key Exchange algorithms are used (Scored)
5.2.16	Ensure SSH Idle Timeout Interval is configured (Scored)
5.2.17	Ensure SSH LoginGraceTime is set to one minute or less (Scored)
5.2.18	Ensure SSH access is limited (Scored)
5.2.19	Ensure SSH warning banner is configured (Scored)
5.2.20	Ensure SSH PAM is enabled (Scored)
5.2.21	Ensure SSH AllowTcpForwarding is disabled (Scored)
5.2.22	Ensure SSH MaxStartups is configured (Scored)
5.2.23	Ensure SSH MaxSessions is set to 4 or less (Scored)
5.3	Configure PAM
5.3.1	Ensure password creation requirements are configured (Scored)
5.3.2	Ensure lockout for failed password attempts is configured (Not Scored)
5.3.3	Ensure password reuse is limited (Not Scored)
5.3.4	Ensure the password hashing algorithm is SHA-512 (Not Scored)
5.4	User Accounts and Environment
5.4.1	Set Shadow Password Suite Parameters
5.4.1.1	Ensure password expiration is 365 days or less (Scored)
5.4.1.2	Ensure minimum days between password changes is 7 or more (Scored)
5.4.1.3	Ensure password expiration warning days is 7 or more (Scored)
5.4.1.4	Ensure inactive password lock is 30 days or less (Scored)
5.4.1.5	Ensure all users last password change date is in the past (Scored)
5.4.2	Ensure system accounts are secured (Scored)
5.4.3	Ensure default group for the root account is GID 0 (Scored)
5.4.4	Ensure default user umask is 027 or more restrictive (Scored)
5.4.5	Ensure default user shell timeout is 900 seconds or less (Scored)
5.5	Ensure root login is restricted to system console (Not Scored)
5.6	Ensure access to the su command is restricted (Scored)
6	System Maintenance
6.1	System File Permissions
6.1.1	Audit system file permissions (Not Scored)
6.1.2	Ensure permissions on /etc/passwd are configured (Scored)
6.1.3	Ensure permissions on /etc/shadow are configured (Scored)
6.1.4	Ensure permissions on /etc/group are configured (Scored)
6.1.5	Ensure permissions on /etc/gshadow are configured (Scored)
6.1.6	Ensure permissions on /etc/passwd- are configured (Scored)
6.1.7	Ensure permissions on /etc/shadow- are configured (Scored)
6.1.8	Ensure permissions on /etc/group- are configured (Scored)



6.1.9	Ensure permissions on /etc/gshadow- are configured (Scored)
6.1.10	Ensure no world-writable files exist (Scored)
6.1.11	Ensure no unowned files or directories exist (Scored)
6.1.12	Ensure no ungrouped files or directories exist (Scored)
6.1.13	Audit SUID executables (Not Scored)
6.1.14	Audit SGID executables (Not Scored)
6.2	User and Group Settings
6.2.1	Ensure password fields are not empty (Scored)
6.2.2	Ensure no legacy "+" entries exist in /etc/passwd (Scored)
6.2.3	Ensure no legacy "+" entries exist in /etc/shadow (Scored)
6.2.4	Ensure no legacy "+" entries exist in /etc/group (Scored)
6.2.5	Ensure root is the only UID 0 account (Scored)
6.2.6	Ensure root PATH Integrity (Scored)
6.2.7	Ensure all users' home directories exist (Scored)
6.2.8	Ensure users' home directories permissions are 750 or more restrictive (Scored)
6.2.9	Ensure users own their home directories (Scored)
6.2.10	Ensure users' dot files are not group or world-writable (Scored)
6.2.11	Ensure no users have .forward files (Scored)
6.2.12	Ensure no users have .netrc files (Scored)
6.2.13	Ensure users' .netrc Files are not group or world accessible (Scored)
6.2.14	Ensure no users have .rhosts files (Scored)
6.2.15	Ensure all groups in /etc/passwd exist in /etc/group (Scored)
6.2.16	Ensure no duplicate UIDs exist (Scored)
6.2.17	Ensure no duplicate GIDs exist (Scored)
6.2.18	Ensure no duplicate user names exist (Scored)
6.2.19	Ensure no duplicate group names exist (Scored)
6.2.20	Ensure shadow group is empty (Scored)

Table 6 – CIS Distribution Independent Linux version 2.0.0 Benchmark (Colored) [56]

5 Prototype Implementation

In this Chapter, the prototype packages and their contents will be demonstrated in full detail, along with the way that they are produced. The source-code of the packages can be found upstream [57].

5.1 cL-Monitoring (Implementation)

Based on the design of cL-Monitoring described in Section 3.3 and the CIS's benchmark for independent Linux distributions the monitoring package must monitor, log and ultimately create a system-map that encircles the following vital points of a device:

- Filesystems used
- Existence of partitions and their special options
- Processes
- Access Control & Warning Banners
- Internet & Special Purpose Services
- Service Clients
- Network Parameters & TCP Wrappers
- Firewall Configuration
- System Accounts & Logging Configuration
- Cron, SSH & PAM Configuration
- User Accounts and Environment
- System File Permissions
- User and Group Settings

A scheme illustrating the structure of cL-Monitoring is attached below:

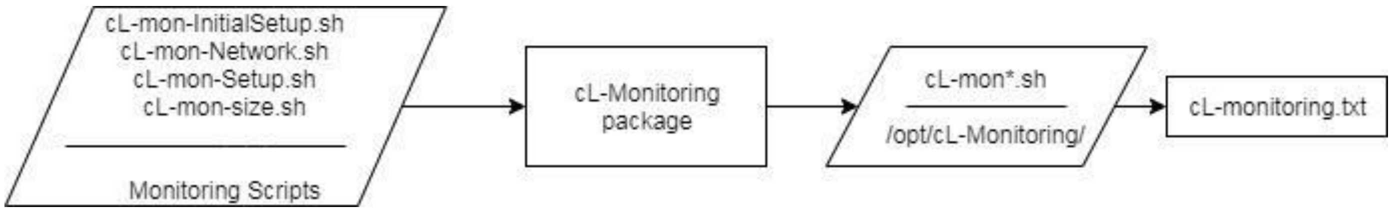


Figure 8 – custom Linux monitoring package

cL-Monitoring is a mechanism consisting of processes that register themselves into the system and are executed periodically and asynchronously to monitor the system's behavior. Those mechanisms were formed using common Linux commands that create a relatively small CPU overhead when they are executed in order to avoid disturbing the system's functionality. Below the four bash scripts that comprise the mechanism described above are presented in detail:

cL-mon-Setup.sh: Sets up firewall logging rules to monitor the traffic generated from and to the system along with the services that are being utilized. Ultimately, this mechanism exposes services that might start but never actually being used and ports that might be open while they are never utilized.

cL-mon-InitialSetup.sh: Checks the filesystems used by the system since the fewer filesystems, the less the attack surface of the system. Also, it monitors the system's partitions to determine whether special options can be applied to further protect them.

cL-mon-size.sh: Monitors the available free space on every drive of the system. The point is to determine the lowest available space in every drive. The system's available disk space is valuable for logging and update purposes.

cL-mon-Network.sh: It is responsible for checking if the firewall rules generated by cL-mon-Setup captured any network traffic along with the traffic's destination. Hardening procedure utilizes cL-mon-Network's output to minimize the attack surface of the system.

Those scripts provide crucial information as to what is needed by the system for it to function as expected, along with what increases the potential attack surface. Also, the available space is obtained. Available space refers to the lowest free disk space in each drive, and it is stored in *cL-mon-size.txt*. It is retrieved to determine if it is possible to be utilized, primarily for logging purposes.

The information retrieved from the above described mechanism is stored in *cL-monitoring.txt. cL-monitoring.txt* is a text file containing tags. These tags refer to the Policy number recommended by Cis-Cat's Benchmark. For example, an entry on cL-monitoring.txt could be '1.1.1.1', which maps to the 'Ensure mounting of cramfs filesystem is disabled' recommendation by CIS's Benchmark. Practically, if this tag is written into *cL-monitoring.txt,* it means that the system is utilizing cramfs filesystem, and it should not be disabled.

A table explaining how the system's settings and functionalities are being monitored is attached below.

System's settings	Monitoring Method
1.Initial Setup	
1.1 - File System Configuration	Monitor which filesystems are mounted on the system via mount command and dmesg message buffer of the kernel. Additionally, monitor the system's partitions, their special options, and the way the OS handles them
1.2 - Configure SW Updates	Check if package manager repositories & GPG keys are configured
1.3 - Filesystem Integrity Checking	No action – (IDS installation is suggested by CIS, but this heavily depends on the system's capabilities)
1.4 - Secure Boot Settings	Check bootloader's permissions, password, and single user's authentication
1.5 - Additional Process Hardening	Monitor whether additional processes like core-dump generation are utilized
1.6 - Mandatory Access Control	No monitoring action – (SELinux or AppArmor installation is not a decision that the mechanism can make; if they are installed then they are configured accordingly to CIS's recommendations)
1.7 - Warning Banners	No monitoring action – (Warning Banners must be configured according to CIS's recommendations)
2.Services	**For all Services, generate iptables logging rules and monitor whether there is traffic generated through the respective services.**
2.1 - inetd Services	
2.2 - Special Purpose Services	
2.3 - Service Clients	
3.Network Configuration	**For all Network Configurations, generate iptables logging rules and monitor whether there is traffic generated through the respective rules.**

3.1 - Network Parameters [Host]	
3.2 - Network Parameter [Host-Router]	
3.3 - TCP Wrappers	
3.4 - Uncommon Network Protocols	
3.5 - Firewall Configuration	
4.Logging & Auditing	**Monitor whether the system can support auditd, then configure it according to CIS's recommendations.**
4.1- Configure auditd	
4.2 - Configure Logging	
4.3 – Ensure logrotate is configured	
5.Access, Authentication & Authorization	**No Monitoring Action - (Configurations in this section are primarily static)**
5.1 - Configure Cron	
5.2 - SSH Server Configuration	
5.3 - Configure PAM	
5.4 - User Accounts and Environment	
5.5 - Ensure Root Login is restricted	
5.6 – Ensure su command is restricted	
6.System Maintenance	**No Monitoring Action - (Configurations in this section are primarily static)**
6.1 - System File Permissions	
6.2 - User and Group Settings	

Table 7 – System Monitoring mechanism

5.2 cL-Hardening (Implementation)

The cL-Hardening package is responsible for generating the desired hardening and applying it to the target embedded system. To generate the ultimate hardening script, the system map generated by cL-Monitoring package is needed as an input.

A scheme demonstrating how cL-Hardening package functions follow:

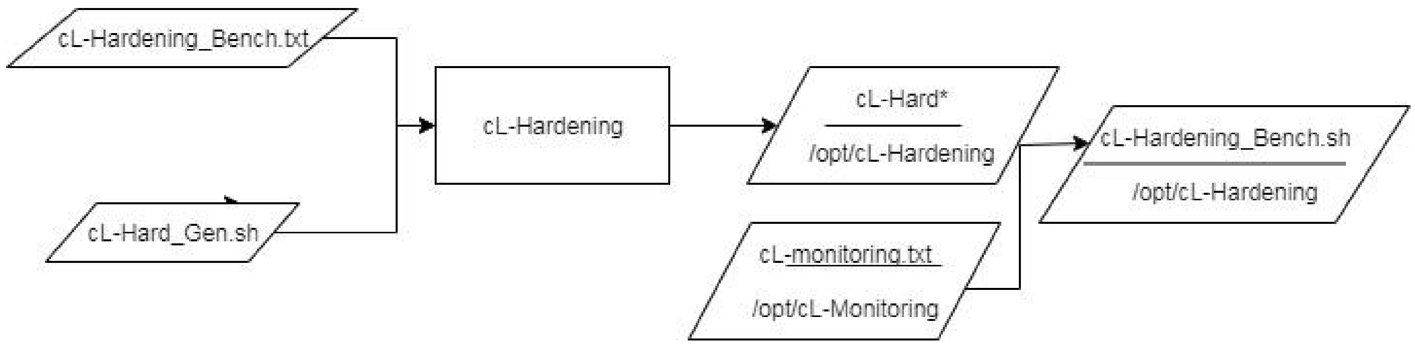


Figure 9 – custom Linux hardening package

cL-Hardening_Bench.txt: Contains every hardening configuration suggested by CIS and is not prohibited to be applied in any Linux system.

cL-Hard_Gen.sh: Generates the hardening script based on the system-map retrieved from cL-monitoring.txt.

cL-Hardening_Bench.sh: Is the hardening script, tailored for the device of which system-map was retrieved from.

5.3 cL-Assessment (Implementation)

cL- Assessment package consists of a process that investigates the system and generates an HTML report. Bash script were selected, to implement the assessment procedure, in the context of keeping the tools used as universal as possible for Linux distributions. The result is an HTML file containing a table with as many rows as the suggested configurations by CIS's benchmark and two columns. The first column contains the hardening recommendations, while the second one is marked as green (Pass) if the system is compliant with the specific policy or red (Fail) if it is not. The report also provides the overall hardening score checked against the CIS benchmark for custom Linux distributions. A simple diagram explaining its concept is displayed below:

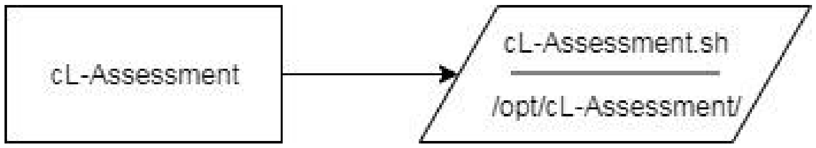


Figure 10 – custom Linux assessment package

To properly asses the compliance of the system, a different approach was used based on the recommendations suggested in each category of CIS's Benchmark for custom Linux distributions. The assessment was based on the core attributes of the six sections of the OS hardening. The table below presents how cL-Assessment confirms if the system follows a suggested CIS's policy or not. If a given recommendation from the CIS's Benchmark for independent Linux distributions is followed, then the check done from cL-Assessment returns a PASS indication meaning that the system's configuration is considered secure as it follows the policy proposed by CIS for the specific part, else cL-Assessment returns a FAIL indication meaning the exact opposite. Below a table can be found presenting how each section is being checked to see whether every policy proposed by CIS is followed.

1.Initial Setup	
1.1 - File System Configuration	Check if filesystem mounting is enabled & the special option on each partition
1.2 - Configure SW Updates	Check if package manager repositories & GPG keys are configured
1.3 - Filesystem Integrity Checking	Check if AIDE (IDS) is installed and run regularly
1.4 - Secure Boot Settings	Check bootloader's permissions, password, and single user's authentication
1.5 - Additional Process Hardening	Check where additional processes like XD/NX support and core dumps are hardened
1.6 - Mandatory Access Control	Check if SELinux or AppArmor are installed and configured
1.7 - Warning Banners	Check if login banners' messages and permissions are configured
2.Services	
2.1 - inetd Services	Check if inetd Services are disabled

2.2 - Special Purpose Services	Check if Special Purpose Services are disabled
2.3 - Service Clients	Check if specific Service clients are installed
3.Network Configuration	
3.1 - Network Parameters [Host]	Check if IP forwarding & packet redirection are disabled
3.2 - Network Parameter [Host-Router]	Check if suspicious packets are not accepted and/or logged
3.3 - TCP Wrappers	Check if TCP Wrappers is installed and permissions on /etc/hosts* are configured
3.4 - Uncommon Network Protocols	Check if uncommon network protocols are disabled
3.5 - Firewall Configuration	Check if IPv4 and IPv6 iptables are configured
4.Logging & Auditing	
4.1- Configure auditd	Check if auditd is installed and configured
4.2 - Configure Logging	Check if rsyslog/journald is installed, enabled and configured and logfiles' permissions
4.3 – Ensure logrotate is configured	Check if logrotate is configured
5.Access, Authentication & Authorization	
5.1 - Configure Cron	Check if a cron daemon is enabled and access permissions on /etc/cron* files
5.2 - SSH Server Configuration	Check if SSH is properly configured
5.3 - Configure PAM	Check if password generation requirements, lockout, reuse, and hashing are configured
5.4 - User Accounts and Environment	Check if /etc/shadow and /etc/passwdare properly configured
5.5 - Ensure Root Login is restricted	Check if root login is restricted to the system console
5.6 – Ensure su command is restricted	Check if access to su command is restricted
6.System Maintenance	
6.1 - System File Permissions	Check if /etc.* system file permissions are configured
6.2 - User and Group Settings	Check if settings related to users, groups and their directories' permissions are configured

Table 8 – System Assessment mechanism

5.4 Packages Creation via Yocto

When configured, Yocto can provide deb, ipk, rpm or tar packages to install the corresponding components into any custom Linux distribution. It can be utilized to create upstream repositories and tailor a package manager according to an embedded system's needs. In this section, the creation and the deployment of the cL packages via Yocto will be demonstrated. Additionally, the Yocto's configuration to implement cross-compilation is also included. Comprehensive information on how to install and get started with Yocto can be found at the Yocto Project Mega Manual [53].

5.4.1 Yocto configuration for cross-compilation

The following steps are followed to properly configure the host system:

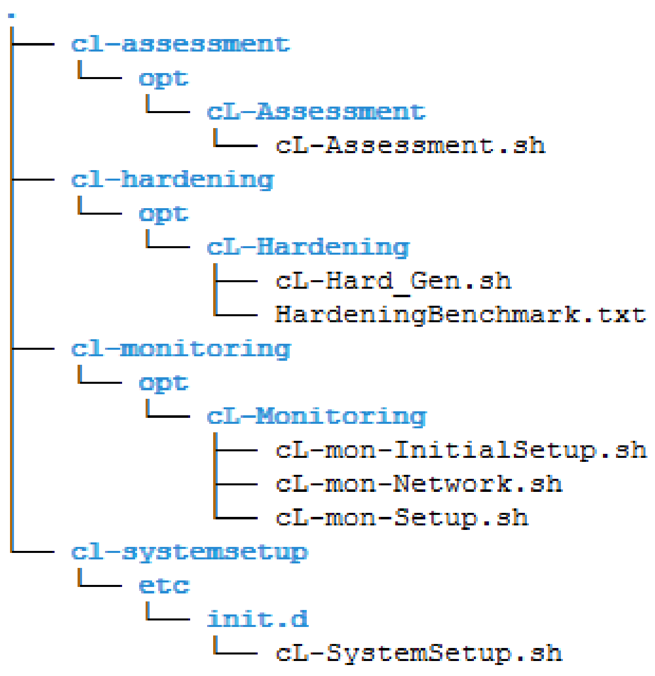
1. First step is to set up a host system to use for creating the packages for the target device. Detailed instructions on how to get started with Yocto can be found at the Yocto Project Mega-Manual. In our case, the host system is a physical machine on which Ubuntu 18.04 OS is installed. The hardware specifications of the host system are: 8 x Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz CPU, 8GB DDR4 RAM and 1TB disk space.
2. Next the appropriate packages Yocto needs were installed to the machine. More specifically the packages that must be installed are: *build-essential, chrpath, gawk, wget, git-core, diffstat, unzip, texinfo, gcc-multilib, socat, cpio, python, python3, python3-pip, python3-pexpect, xz-utils, debianutils, iputils-ping, python3-git, python3-jinja2, libegl1-mesa, libsdl1.2-dev, pylint3 and xterm.* These packages are required for Yocto to be able to function as expected for the host Ubuntu development system. Additionally, to utilize the build-system, host development machine needs to have Git version 1.8.3.1 or higher, tar 1.27 version or higher and Python version 3.4.0 or higher.
3. Once the setup of the host-system is complete, we need to get a copy of the Poky repository on the host machine. To do so, the following command can be used: *git clone git://git.yoctoproject.org/poky* . "Poky", is the name of the Yocto Project's Reference OS Kit or reference distribution. Poky contains the OpenEmbedded Build System (BitBake and OpenEmbedded-Core) along with a set of metadata to get us started building our own distro. By default, when one uses *git clone* to download the Poky directory it downloads the master branch. Optimally, for the packages to be compatible with the target-device for which the cross-compilation will take place the glibc recipe version of the Poky repository needs to be the same as the one already used in the system. Thus, we must first check what is the glibc version of the target device and then checkout the Poky branch which contains the same glibc version. For our use-case *morty* is the appropriate branch that must be used.
4. Finally, Yocto must be configured to cross-compile based on the specifications of our target-device. To specify the properties of the target-device *local.conf* should be adjusted appropriately. For our use-case the main variable that must be changed inside this file is the *MACHINE variable* and its value must be “qoriq”.

To better understand how Yocto works and how it should be configured in order to cross-compile one should continue the study of the Yocto Project using the Mega-Manual as her/his guide.

5.4.2 custom Linux packages generation

To generate a package via Yocto, a recipe file with bb (bitbake) extension is needed. These files have a specific structure and enable Yocto to identify the way the source code of the package should be configured and compiled along with the path that it should be installed at the target system.

The recipes created to generate the cL packages referred to in the previous chapters are uploaded upstream [57]. Below a picture depicting how Yocto delivers the packages at the target system is attached. The image examines all the generated packages. cl-assessment, cl-hardening, and cl-monitoring are installed in the opt directory as according to the Filesystem Hierarchy Standard, /opt is for "the installation of add-on application software packages." cL-SystemSetup is installed at /etc/init.d as it is a startup file designed to run at system boot.



```
.
├── cl-assessment
│   └── opt
│       └── cL-Assessment
│           └── cL-Assessment.sh
├── cl-hardening
│   └── opt
│       └── cL-Hardening
│           ├── cL-Hard_Gen.sh
│           └── HardeningBenchmark.txt
├── cl-monitoring
│   └── opt
│       └── cL-Monitoring
│           ├── cL-mon-InitialSetup.sh
│           ├── cL-mon-Network.sh
│           └── cL-mon-Setup.sh
└── cl-systemsetup
    └── etc
        └── init.d
            └── cL-SystemSetup.sh
```

Figure 11 – custom Linux proposed packages provided by Yocto

5.5 Performance Assessment

The mechanism's performance was measured against the system utilized as a use-case. The system's Hardware characteristics are important since they influence the results obtained for the proposed methodology; thus, they are presented in detail below:

CPU	Type: e500v2 Clock: 800 MHz Number of CPUs: 2
Memory	Type: DDR3 1333mhz Total Memory: 1Gb
Storage Type: SDHC Card	Memory: 8GB

Table 9 – use-case's HW specifications

To accurately measure the performance of the bash scripts' that compose the mechanism, each script was run 100 times, and its performance was measured using the time Linux command with the appropriate parameters that report the:

- Percentage of the CPU that this job got
- Total number of CPU-seconds that the process spent in user mode
- Total number of CPU-seconds that the process spent in kernel mode
- Elapsed real time

After that, the average value of the 100 runs for each parameter of every script was considered as its performance metric. Based on the methodology described above, the following results were obtained for each script separately.

Performance Metric/Script	cL-mon-InitialSetup.sh			cL-mon-Setup.sh			cL-mon-Network.sh		
	min	aver	max	min	aver	max	min	aver	max
CPU (%)	83	9	93	65	67,6	69	72	77,5	79
CPU usage (sec/user-mode)	1,95	2	2,11	8,21	8,71	9,19	1,54	1,64	1,72
CPU usage (sec/kernel-mode)	0,5	0,57	0,67	11,11	11,96	12,58	0,65	0,75	0,84
Elapsed real time (sec)	2,8	2,87	3,14	29,75	30,31	31,03	3,01	3,06	3,35

Performance Metric/Script	cL-mon-size.sh			cL-Assessment.sh			cL-Hardeni_Gen.sh		
	min	aver	max	min	aver	max	min	aver	max
CPU (%)	28	32	34	88	89,2	90	73	83	87
CPU usage (sec/user-mode)	0,1	0.15	0,21	16,24	16,71	17,22	0,2	0,25	0,31
CPU usage (sec/kernel-mode)	0.25	0.31	0,37	26,4	26,94	28,52	0,42	0,46	0,54
Elapsed real time (sec)	1,38	1,41	1,7	48	48,58	50,45	,082	0,85	1,04

Table 10 – Performance metrics of the mechanism's components

The above indexes make it possible to decide when a monitoring process fires, considering that the system's functionality-performance must remain unchanged while monitoring takes place.

cL-mon-Setup.sh: It can run only once per day. When the mechanism is deployed for the first time it sets up the appropriate monitoring-rules. Then, the script should only run to verify that the monitoring rules are still there. If rules have been removed for any reason, the mechanism re-establishes them.

cL-mon-Network.sh: Since cL-mon-Network primarily collects persistent information through iptables, its fire-off period would not affect its functionality if large. Additionally, since it only takes on average 1.23sec to complete with a CPU load of 71%, firing off once per hour would not affect the system's functionality.

cL-mon-size: It is desired to track changes in the available disk space of the system as often as possible. Thus, considering the low CPU consumption and the low time of completion for cL-mon-size enabling it to run once per minute is the choice in this case.

cL-mon-InitialSetup.sh: cL-mon-Initial-Setup is responsible for monitoring changes that occur in the filesystem. These changes might be logged in circular message buffers or in the filesystem itself, and they are dynamic. Thus, to ensure that all the information that the specific monitoring process logs will be captured, the decision is to run it once every minute, even if it creates a relatively high CPU load.

6 Case Study

6.1 OpenScape Business X -Overview

6.1.1 OpenScape Business X - Hardware

OpenScape Business X is an Embedded System that is being installed in the customer's property, and functions as a Telephony Center enhanced with many capabilities. OpenScape Business X series (also known as OSBiz X) mainly targets Small and Medium Enterprises offering Voice Unified Communication (UC) services. UC provides Web Collaboration, voice and fax message boxes, notification service, mobility, a Contact Center, and presence status functions. As a standalone system, it can support up to 500 subscribers, and in networked systems, up to 1000 subscribers can be connected. Below the three versions of OSBiz X are presented. The main difference between X3, X5, and X8 is the number of slots they have. These slots are being used for extension cards to be added. In practice, these cards are proprietary hardware that enables the connection of the system with analog and digital phones [58][59].

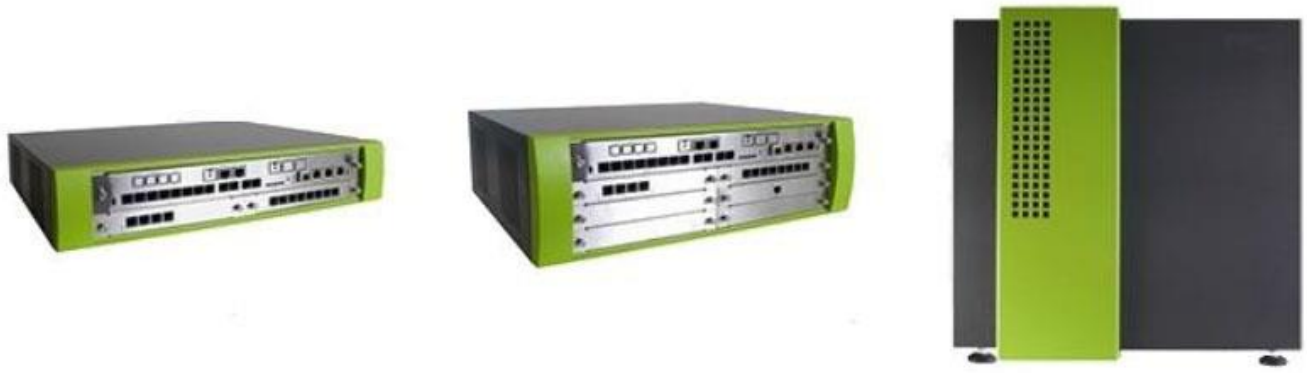


Figure 12 – OSBiz X series from left to right X3, X5 & X8

From the hardware perspective, the OSBiz X series run on a PowerPC architecture (qoriq p1021, p2020 e500v2 core) and are enhanced with native IP and restricted UC functionality. To support more users, extension cards are being utilized. These cards are inserted in the slots below the mainboard and provide ports to connect actual phone devices with the system. The storage device is an 8GB SDHC (Secure Digital High Capacity) card, which, apart from the system's software, contains also the BIOS. Furthermore, OSBiz X may be enhanced with a UC Booster Card (extra embedded system) or a Booster Server (a computer server). Both options improve the system's processing and storage capabilities while at the same time, they provide the UC Suite, which is the full set of UC services.

The OSBiz X mainboards are custom designed. The design of the mainboards takes place in Curitiba, Brazil, by a dedicated Hardware Design Team while the manufacturing takes place inside the European Union. Below an image showing the mainboard of an OSBiz X3 is illustrated.

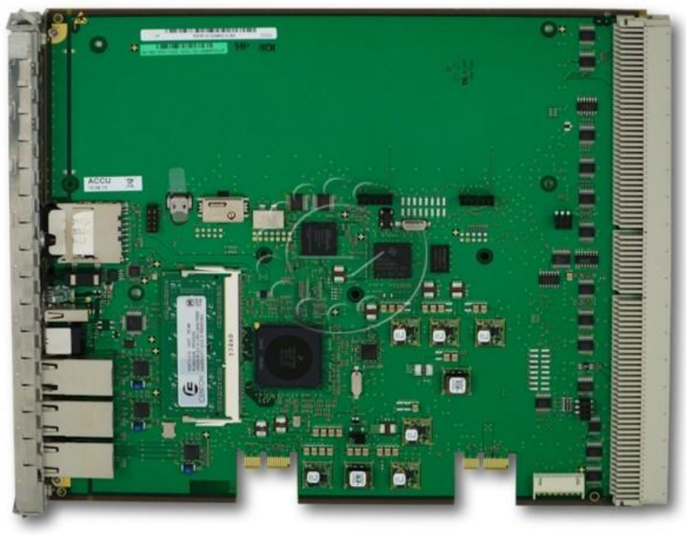
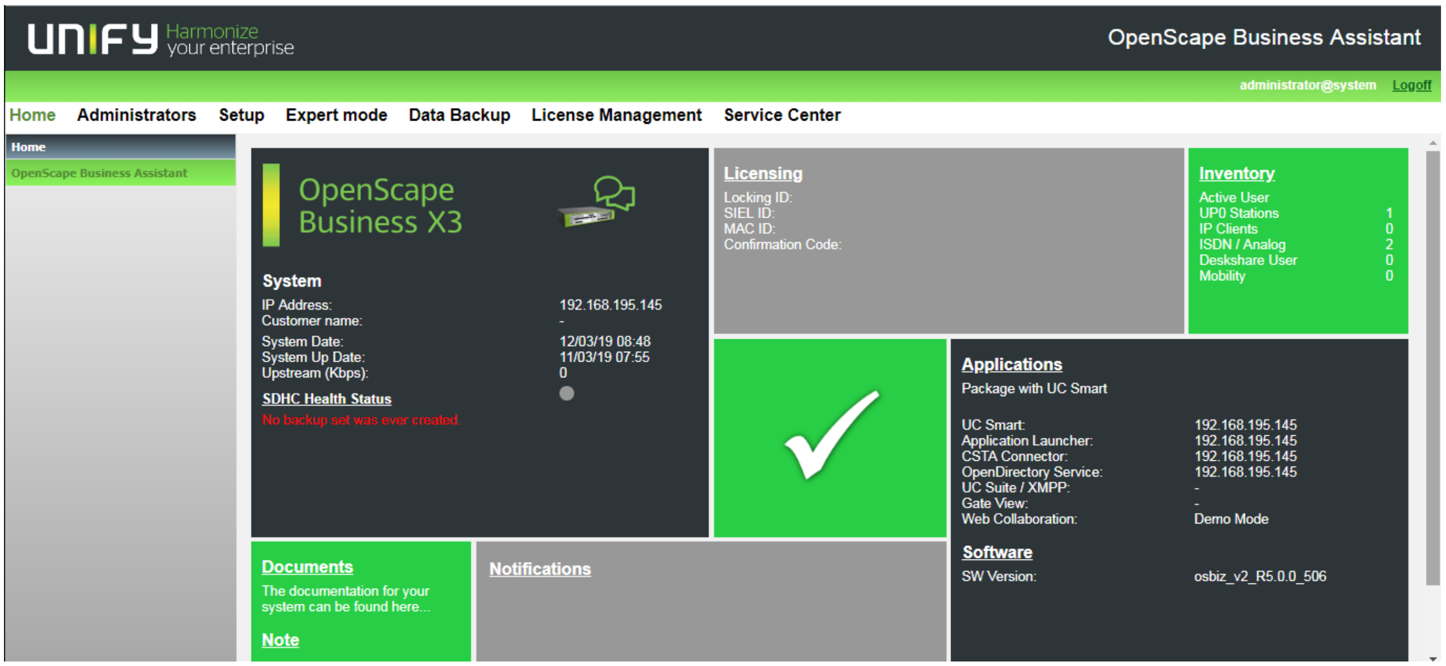


Figure 13 – OSBiz X3 Mainboard

6.1.2 OpenScape Business X - Software

From the software perspective, OpenScape Business X runs on a Debian-custom Linux distribution that includes a kernel based on the ones provided at kernel.org. Its functionality depends on 32bit Open-Source components and proprietary applications that have been tailored for PowerPC architecture. Additionally, third-party applications are also being used in the system. OSBiz X features a Web Interface called WBM (Web-Based Management) that enables users to have access and configure the system in real-time.



System

IP Address:	192.168.195.145
Customer name:	-
System Date:	12/03/19 08:48
System Up Date:	11/03/19 07:55
Upstream (Kbps):	0
SDHC Health Status	●

No backup set was ever created.

Licensing

Locking ID:
SIEL ID:
MAC ID:
Confirmation Code:

Inventory

Active User	
UP0 Stations	1
IP Clients	0
ISDN / Analog	2
Deskshare User	0
Mobility	0

Applications

Package with UC Smart

UC Smart:	192.168.195.145
Application Launcher:	192.168.195.145
CSTA Connector:	192.168.195.145
OpenDirectory Service:	192.168.195.145
UC Suite / XMPP:	-
Gate View:	-
Web Collaboration:	Demo Mode

Software

SW Version:	osbiz_v2_R5.0.0_506

The documentation for your system can be found here...

Figure 14 – Web-Based Management

OpenScape Business X is an "all in one solution" PBX (Private Branch Exchange) telephony system. It is a highly successful product and very well positioned in the global market. OSBiz X is a feature-rich system, highly competitive, and well accepted by the global market. In its six years journey, it has more than 5 (five) million licensed users and 150 (one hundred fifty) thousand installed systems on more than 70 countries. Moreover, it is a multi-award-winner product that excels in its area.

Figure 15 –OSBiz X highlights

The embedded system up to now was primarily being deployed in customers who chose not to have the system online. This way, the system was limited to function on a local network without having access to the global web. Thus, systems were a lot more protected. These days, internal research shows that more than 60% of the embedded systems deployed have Internet access. With this increased rate, the necessity for securing the embedded system has now become the number 1 priority as more and more incidents are occurring mainly due to this change. Currently, there are five million (5.000.000), active users, on the field. Since in this analysis, the central aspect is security, the information included below will be concerning that, which means that might be more details that will be neglected if they are not relevant to security.

6.2 OpenScape Business X Vulnerabilities

During its journey, OSBiz X has faced several attacks. Furthermore, some dangers have not been exploited yet, or the company is not aware of it. Below the sources feeding Atos with the system's vulnerabilities are presented.

- Affected users are reporting the security incidents directly to the company.
- A dedicated-security team is updating a list with all vulnerabilities discovered in open-source packages used in OSBiz X.
- Security incidents discovered by the Quality Assurance team.
- Penetration tests.

All incidents are of crucial importance, but reasonably the ones reported by the customers come first as they have already been exploited to some extent. Another reason these vulnerabilities come first is that they can directly affect the company's reputation and earnings at the same time.

To describe each vulnerability and how it can be exploited is beyond the scope of this thesis. A team of vulnerability managers updates the list with new entries as soon as they are made public. High importance vulnerabilities are handled with priority by security experts. This environment is very dynamic, but at the time of this thesis, 156 open items need to be handled.

Nevertheless, it would be interesting to have an overall image of the nature of these vulnerabilities. For the sake of that, the current vulnerability list was analyzed in detail, and the existing vulnerabilities that lie into OSBiz X have been categorized based on the convention described in Section 2.6. The results of this analysis are illustrated in the table below:

No of Vulnerabilities	**Precondition**	**Attack Method**	**Vulnerability**	**Target**	**Effect**
93	Local or Remote Access / Internet	Injection/Control Hijack	Insecure Configuration	Application /Operating System	Information Leakage / Code Execution / Illegitimate Access / Denial of Service
27	Local or Remote Access	Normal Use/Control Hijack	Insecure Configuration	Operating System	Information Leakage (Sensitive Data)
9	Internet Facing	Injections	Insecure Configuration	Application	Denial of Service
6	Internet Facing / Local or Remote Access	Control Hijack	Weak Access Control or Authentication	Application	Financial Loss (Toll fraud)
5	Direct Physical	Reverse Eng	Weak Access Control or Authentication	Device	Information Leakage (Sensitive Data)
4	Internet Facing	Normal Use	Web	Application	Information Leakage (Sensitive Data)
4	Local or Remote Access	Control Hijack	Programming Errors	Application	Illegitimate Access / Information Leakage
3	Local or Remote Access	Normal use	Weak Access Control or Authentication	Protocol	Information Leakage (Sensitive Data)
3	Internet Facing	Malware	Web	Application	Code Execution

Table 11 - Vulnerability Categorization

As derived from the table above, most vulnerabilities are either application or operating system specific. To be more precise, the first row is about vulnerabilities due to the out-of-date open source packages inside the system. The production system of OSBiz X makes it significantly challenging to update packages. So, this creates a pilling stack of vulnerabilities. In the second place, the vulnerabilities due to the insecure configuration of the Operating System can be found. The operating system needs to be configured carefully, but this can be tricky as the more hardened an operating-system the most probable its applications' functionality is negatively affected. Other vulnerabilities include internet threats such as injections and malware that are similar to the ones faced by standard PCs. Furthermore, there are programming errors that create security flaws. Finally, there are embedded system-specific vulnerabilities since direct physical access can enable intruders to find sophisticated ways to mess with the system.

7 Prototype and Results

In this section, the prototype and the results obtained when applied to OpenScape Business X will be presented. The method used to harden Open Scape Business X version 2 R7.0.1_87 system follows. Finally, the results obtained in terms of hardening and functionality after applying the prototype are illustrated.

7.1 Monitoring Procedure

The monitoring procedure is the most crucial since, based on its results, the system adjusts and creates a corresponding hardening mechanism. The monitoring scripts are initialized as asynchronous, periodic processes. They, in the background, check the components of the system that are being used when the system is fully functional. The most system functionalities are triggered during the monitoring-period, the most accurate the generated system-map will be. The monitoring period's duration vastly depends on the examined product. For Open Scape Business, it was decided to monitor the system for a total of five days. This decision was made as the Quality Assurance department needs three days for performing the majority of its test suites. Additionally, to ensure that serviceability was intact, the hardened product was handed to the developers to test its serviceability and functionality. There are times when the developer needs direct access to a user's product to patch it or investigate it for possible issues. The testing period from the development's department side was decided to be two days.

For the examined use-case cL-monitoring package was installed to OpenScape Business X. Then, the system was given to the Quality Assurance Department. There the system was in full test mode for three days. Manual, as well as automatic test suites, were executed. The comprehensive test reports can be found upstream [57]. After that, the system had to be tested from the developer's side. This investigation was conducted by the Platform & Drivers team who is responsible for the system's Operating System. For two days, all the common scenarios that may occur when a developer connects to a system remotely were reproduced. A list, including all the test cases executed, can be found upstream [57].

After the testing period, *cL-monitoring.txt* was complete. It contained all those CIS-CAT's recommendations that would possibly affect the system's functionality negatively if hardened. The actual *cL-monitoring.txt* is included upstream [57].

Furthermore, to accurately map the monitoring-procedure, before testing a particular version of the cL-monitoring package that includes a timestamp to identify the time a system's configuration was identified that will negatively affect the system's functionality if hardened was created. The results of the above-mentioned package are illustrated in the graph below. The package was deployed before the QA department started automatically and manually testing the system. The noticeable result here is that it can be observed that the graph stabilizes after 3 hours, indicating that the mechanism was able to identify which system's settings hardening could negatively affect. Nevertheless, the testing period should not be reduced due to size-monitoring. So, each system's functionality may require extra disk space. To identify the lowest point of disk space usage, all use-case scenarios must be performed.

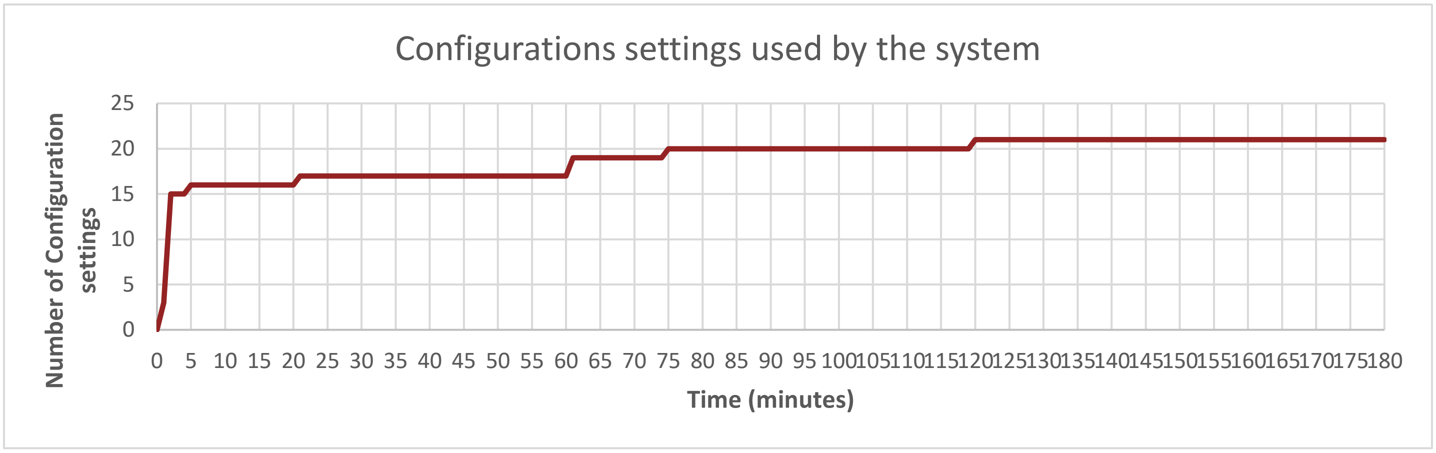


Figure 16 – Configuration Settings used by the target system

7.2 Hardening Procedure

When the monitoring procedure is complete, hardening package is able to use its output (cL-monitoring.txt) to generate a hardening shell tailored for the monitored device.

To do so, the hardening process initially considers every hardening configuration proposed in CIS's benchmark except from the prohibited ones (Table 6) for custom Linux distributions and from this set it subtracts the configurations that will possibly affect the system's functionality.

This process is implemented by *cL-Hardening_Bench.sh*. Its execution time on OSBiz-X was 27.3 seconds. The use-case device's functionality was not stopped following the hardening.

Immediately after that, the *cL-Assessment.sh* was run to obtain the corresponding report and check what hardening recommendations were applied.

The system was once again tested using the procedure described in Section 7.1. Quality Assurance department and Platform & Drivers team re-run all their automatic and manual test suites to verify the system's functionality and serviceability state. This was to investigate if the device's functionality and serviceability were affected due to the hardening.

7.3 Assessment Procedure

The assessment package was deployed to the examined OSBiz-X. The report generated by the assessment package before hardening is included in the upstream repository [57]. The format of the report is an HTML file. Following that, the assessment of the system once again took place after the Hardening procedure of the embedded system. The results obtained, after the hardening was applied, are also included upstream [57].

The goal of assessment report is to provide information related to the security configuration state of a given system. The checks that generate the corresponding report are included in the source code of cL-Assessment.sh script and are described in detail to the Section 5.3 where the implementation of cL-Assessment package is explained.

7.3.1 Compliance Assessment

The degree in which the proposed methodology answered the main research question could be measured by evaluating the OpenScape Business's X overall security level against the CIS's benchmark for custom Linux distributions along with its degree of functionality and serviceability.

To assess its overall security level, cL-Assessment is utilized. The results are

Section/Compliance Score	Before Hardening	After Hardening
Initial Setup	16/51	29/51
Services	31/34	31/34
Network Configuration	9/22	13/22
Logging & Auditing	6/29	20/29
Access, Authentication & Authorization	17/39	32/39
System Maintenance	18/31	21/31
Total Score	**97/206**	**146/206**

Tables 12 – Before vs. After Hardening Assessments

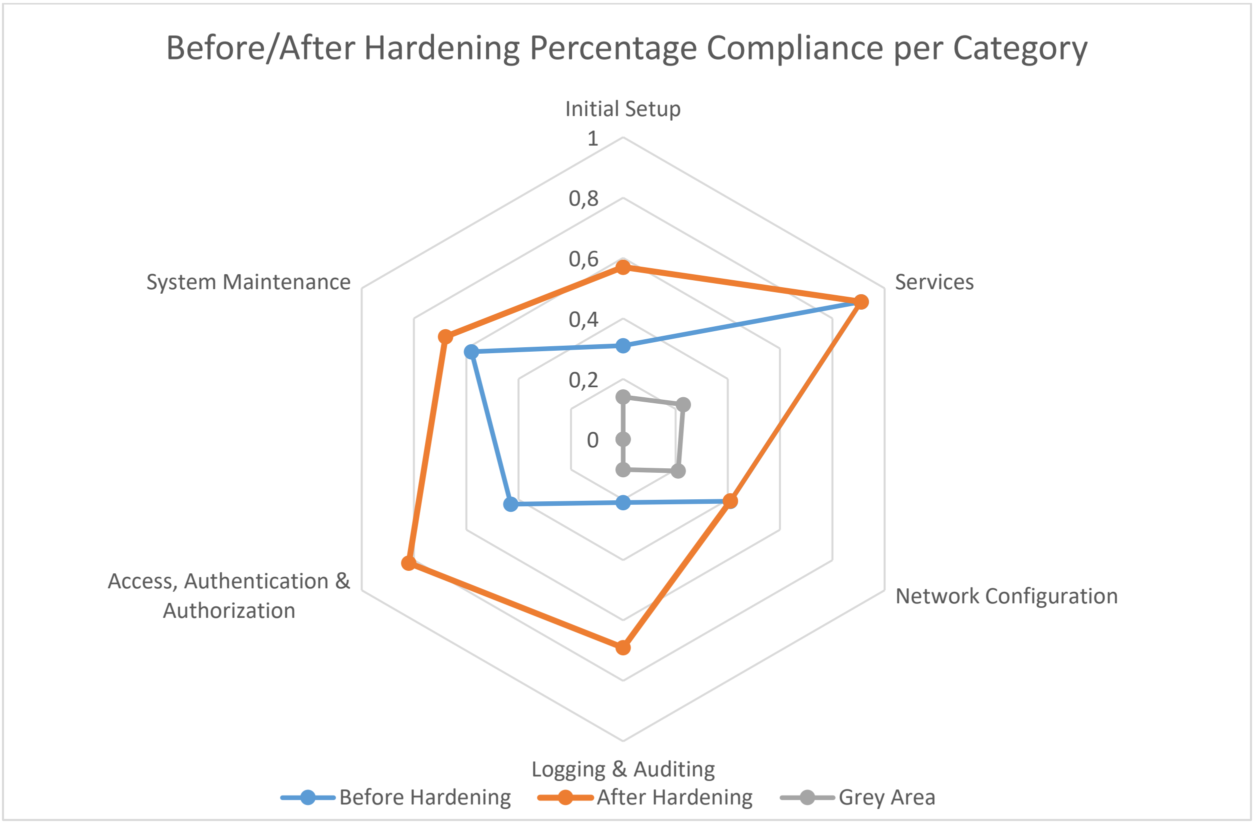


Figure 17 – Spider Graph Before vs. After Hardening and Grey Area

The grey points represent the percentage of configuration policies that are being utilized in each category by the target system. These configurations have been identified by cL-Monitoring as assisting in the target system's standard functionality. Nevertheless, these configuration policies are also in the CIS' benchmark with the ones that must be hardened for the system to be fully compliant with CIS's benchmark. The proposed mechanism favors functionality over security; thus, the configuration policies that assist in the system's standard functionalities remain intact.

7.3.2 Compliance Assessment Analysis

Results show increment in the percentage of the system's compliance with the CIS's Benchmark after the custom Linux hardening was applied to it. In detail, this percentage raised from 47,1% to 70.8%. Vast compliance improvements are observed in the Initial Setup, Logging & Auditing and Access, Authentication, and Authorization areas. Overall, every section has experienced compliance improvements to some degree. Initial Setup is one of the two sections that experienced a significant compliance adjustment.

7.3.2.1 Initial Setup

The initial setup encircles configurations related to the available to be mounted filesystems, partitions and their special options, and mandatory access control software. System monitoring revealed that while some filesystems where enabled inside the kernel, they were never being used, so they were disabled. Also, for existing system partitions like tmp, monitoring shows that some of the suggested special options can be applied, but not all of them since monitoring reported that noexec and nosuid special options are actually interfering with the system's function as it is. For mandatory access control software, no changes were made since the packages are not installed into the system.

7.3.2.2 Services

It can be observed that the Services section for OpenScape Business X is almost perfectly aligned with CIS's recommendations even before hardening. Atos has a dedicated team handling package vulnerabilities for OSBiz X. Services layer is one of the first layers that can be attacked in systems with Internet access, and the exploitation of this layer is mainly due to vulnerabilities. The Atos' security team has already been all over the aforementioned vulnerabilities handling them appropriately.

7.3.2.3 Network Configuration

OpenScape Business functions as a telephony center. Its functionality heavily depends on handling, redirecting, sending, and receiving network packets. Due to its nature, network configurations are relatively loose. The monitoring mechanism used in this thesis project, indicated that the network configurations have to remain loose and revealed that their hardening could most definitely lead to functionality degradation. On the other hand, some logging and validation methods through iptables were enabled since they do not affect the system's functionality, and they can enhance the way network traffic is monitored and handled.

7.3.2.4 Logging & Auditing

Analyzing the assessment reports, the logging and auditing section exhibits significantly improved compliance. OpenScape Business has its custom logging mechanisms, but they are mostly logging events that support the debugging procedure and collect global-system events. The logging and auditing section includes recommendations that focus on keeping track of intentional environment changes an intruder might perform to the system.

7.3.2.5 Access, Authentication & Authorization

Access, Authentication and Authorization sections presented an almost double compliance score when the system was hardened in comparison with before. Cron permissions and SSH related configurations are what made the difference in this section. From my personal experience, I know that SSH access is prohibited for the end-users, and only developers or authorized personnel might connect via SSH to any OSBiz system.

My perception is that because of that, system architecture did not take into account properly configuring SSH daemon apart from setting a robust password for the root user and relied on that. Changes in this section primarily add an extra layer of protection regarding who uses the SSH daemon.

7.3.2.6 System Maintenance

In the system maintenance section, one can notice that the system after hardening does not improve its compliance significantly. Two factors contribute to this situation. First, permissions on crucial files like passwd and group were already configured since they are fundamental prevention measures. Second, and more important is that this section includes recommendations that are system-specific and were decided not to be adopted. For example, ensuring that every user has a home directory is not always the case since some users, such as services that might have a user account, should not have a directory where they can create files. Detailed information on why it was decided to prohibit some hardening configurations proposed by CIS can be found at Subsection 4.4.1 .

7.3.3 Functionality Assessment

To assess its functionality degree, the Quality Assurance department was once again triggered. The tests that were run during the monitoring procedure were repeated after configuration hardening was applied. The decision to run the same test suites was made as the custom Linux Hardening script adapts to the data collected during the monitoring period. Running different types of tests would probably result in invalid results since the system was hardened under a specific environment.

The results obtained from the two test suites executed are displayed in the graphs below.

The first test suite, which runs 309 automatic test cases returned the following results:

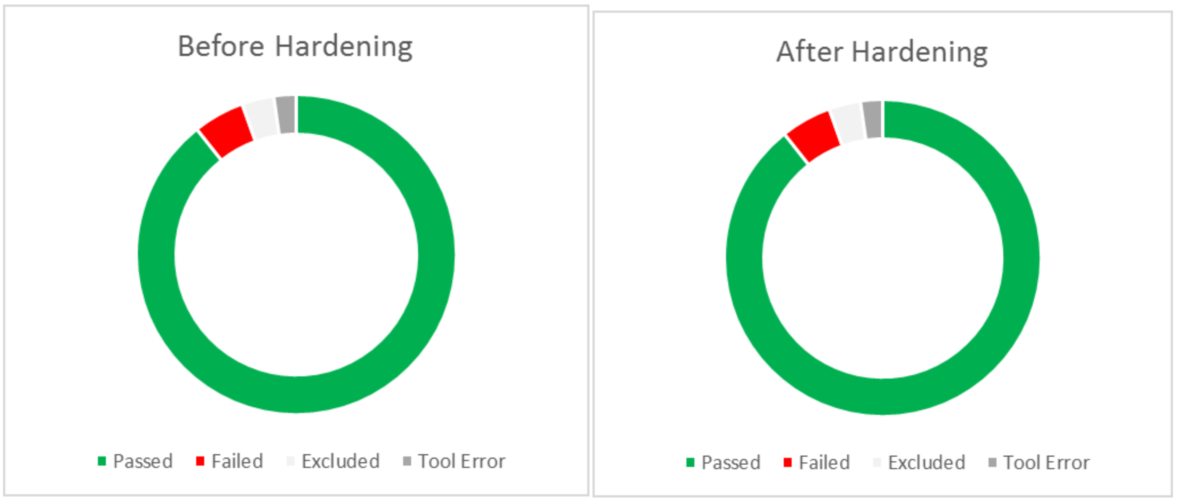


Figure 18 – HUSIM-V2 Results Before & After Hardening 309 test cases

The results are exactly the same before and after the hardening procedure, indicating that hardening has not tackled the OpenScape Business X's functionality. The complete test lists which were performed are included in the attached file named HUSIM-V2-309TestCases.xls

The second test suite was based on a Java Framework. This suite is also automatic and runs 103 test cases. Below the results are presented. Full test description along with a Pass/Fail Indication is also attached in an excel file named *JavaFrameWork-AutomaticTests-100TestCases.xlsx*

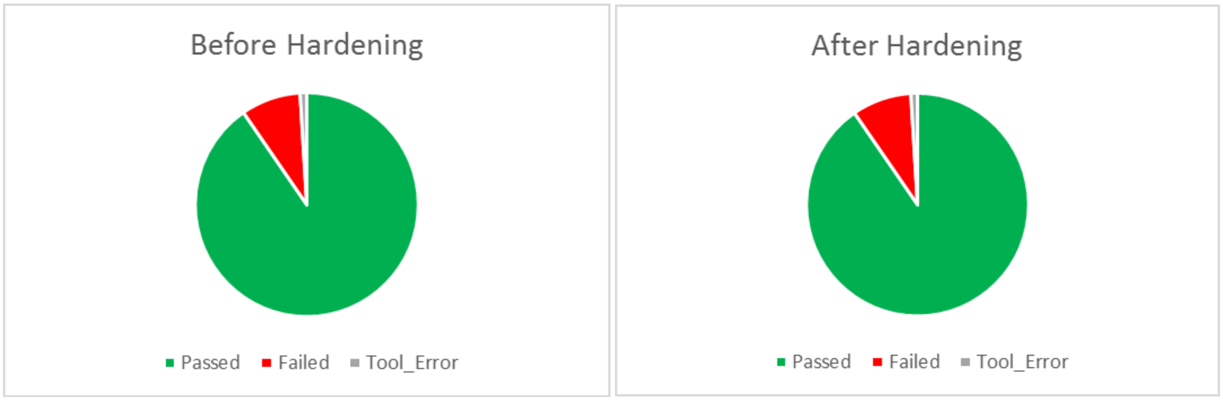


Figure 19 – JavaFrameWork-V2 Results Before & After Hardening – 103 test cases

In this test suite, the same behavior pattern was observed. The same results were obtained before and after the hardening was applied to the system. Furthermore, the exact same tests that passed or failed before the hardening are the ones that pass or fail after hardening is applied. More information is included in the excel file mentioned above.

From the above, it becomes clear that the system can adequately adjust to the given conditions. The test cases that were executed were chosen because apart from just using basic system functionalities, they trigger interactions with users using network interfaces. Keeping functionality intact is of crucial importance for the success of the proposed system since applying security configurations that break functionality can generate unpredicted behavior to the deployed systems.

7.3.4 Serviceability Assessment

To assess its serviceability, manual tests were performed by the Platform & Drivers Development Team.

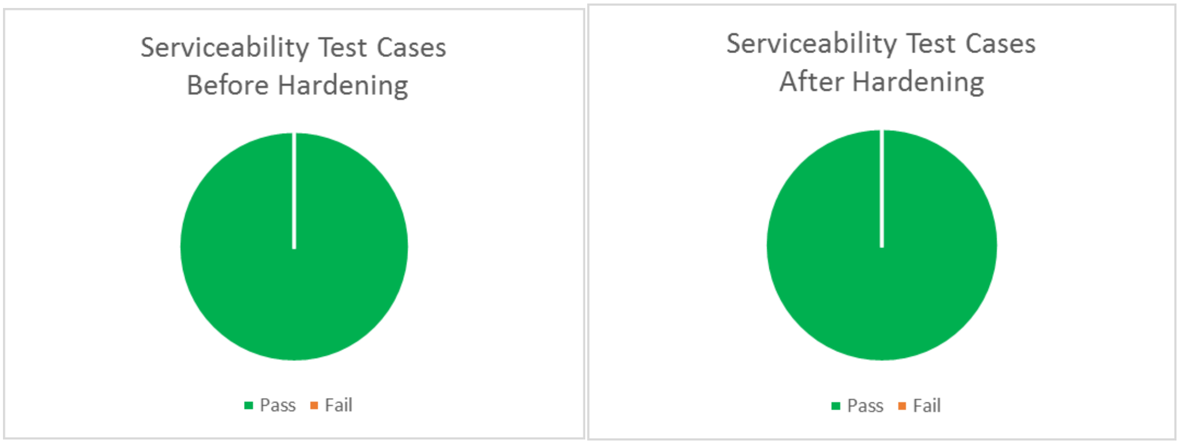


Figure 20 – Serviceability Results Before & After Hardening – 103 test cases

The complete test-list explaining the serviceability tests executed is included in the upstream repository [57]. All tests were passed successfully, indicating that there were no issues related to the serviceability. Furthermore, the banners configured from the hardening procedure were perceived as useful additions from the developers who tested the system.

Serviceability is of crucial importance since apart from the system’s functionalities the whole user’s experience in terms of using and servicing the product should not change due to the hardening. The ultimate goal is for the hardening to be as seamless as possible when perceived from the user side. This can be concluded only by verifying that both functionality and serviceability were not affected by the hardening procedure.

8 Conclusions and Discussion

8.1 Main Conclusion

Most embedded systems/IoT devices are composed of computing devices highly specialized to execute a particular purpose. The specialization and the nature of those systems results in constraints related to their energy consumption, performance, and cost. Consequently, those systems often have limited security-related capabilities when compared to standard operating systems.

This issue affects both the systems that will be deployed from now on and the systems that are already deployed in the real-world. The IoT is formed by a continuously growing number of interconnected embedded systems, and most of these systems lack even the simplest form of security features; this leads to numerous successful attacks deployed to them or even through them in recent years. The results of these attacks vary from compromising sensitive data to weaponizing data or even threatening the safety of individual users. Therefore, the need for providing security capabilities to already deployed devices becomes mandatory.

This thesis provides the first step towards developing an automated approach to increase the security level of interconnected devices without disrupting their functionality. Within Atos, a European multinational information technology service and consulting company, I have designed, implemented, and evaluated a novel approach based on hardening the security configurations of existing embedded/IoT devices to make them more robust against incoming attacks.

Unfortunately, applying hardening to OS's configurations often leads to disturbance of the target device's functionality since some of the 'non-secure' configurations are essential to the system's functionality. Thus, this thesis investigated an automated approach to hardening the target's configuration, while at the same time leaving its functionality intact

The core-idea of this proposal is to monitor the system, while it remains fully functional, and identify which of its configurations are essential. Then the goal is to create a hardening-shell that will harden system configurations non-essential to the system's function. Notably, the hardening will be based on the most recent and well-recognized policies established by the cyber-security community, and it will not affect configurations essential to the system's function.

To this end, a prototype mechanism, aiming to increase the overall security of a target embedded/IoT device, was designed and implemented. To explore its effects on a real-world example, the prototype was deployed in a product that is already present in the market, manufactured by Atos, named OpenScape Business X.

To examine if the proposed mechanism increases the security level of the target device, a benchmark created by the Center of Internet Security specifically for custom Linux distributions was utilized as a baseline. The results reveal that the proposed mechanism did increase the system's security compliance at a significant level while its functionality and accessibility remained undisturbed when the hardening was applied. Thus, it can be concluded that the proposed method can increase the overall security level of these types of systems without affecting their functionality.

Through the research, the information required to be collected from an embedded system to generate a hardening script tailored for it were identified, and a method to obtain them was developed. Utilizing the information mentioned above, a process to produce the actual hardening script was developed. Along with that, a process to evaluate the system's overall security level was proposed and implemented.

For the system used as a case study tests to identify its functionality and serviceability level, after the hardening occurred, were performed. These tests verified that the system remained fully functional after the hardening while at the same time its overall security level was increased.

Furthermore, due to the results obtained from applying the proposed mechanism in OSBiz X, Atos did adopt the solution provided by this research and planned its inclusion to the next SW version release of the product as no apparent drawbacks were observed.

8.1.1 Answering sub-questions

In this section, a conclusion for each of the six research questions will be provided. Based on these conclusions, I derived the answer to the main research question of this study.

SQ1. What mechanisms are being used to secure the configuration of an interconnected device's OS?

In Chapter 2, I formed a search strategy based on Systematic Literature Review that included the search for the ways that have been employed to defended deployed embedded/IoT device. During the research I came across different mechanisms that have been used to achieve the security configurations of devices directly connected to the Internet.

J. Holcomb investigated a method that can audit the system's OS configuration and generate a report after the audit — displaying the status of each audit check as well as a summary. Cheng-Liang Kuo and Chung-Huang Yang, explored a security design for configuration management targeting Android devices. Their main objective was to make an Android device conform with Google Android OS Benchmark issued by CIS while allowing the end-user to decide wheter security configurations that are not adjusted based on CIS's recommendations should be adjusted or not. Boheung Chung, Jeongyeo Kim, and Youngsung Jeon investigated the possibility of providing an on-demand security configuration to adjust the security configuration of the target devices.

Generally, it can be concluded that the mechanisms utilized to secure the configuration of a system do not automatically harden the system on their own. Instead, they tend to give the system Admin, or even the user, the possibility to do so based on an audit conducted to the system. Thus, a gap between identifying the non-secure configuration and automatically applying the remediations was observed, which could be due to the fact certain configurations might need to be left unchanged for the system to function properly.

SQ2. To what extent can a system-configuration of devices using Linux be considered secure? Is there a way to quantify this value, thus measuring the system's configuration security level?

Searching through the Internet, I found different ways to identify whether a Linux configuration is considered secure or not; I searched mainly in the format of checklists, benchmarks, or hardening guides. At the same time, no central checklist repository exists, and it was sometimes hard to figure out if the checklist is current or not.

To effectively do so, the National Checklist Program, defined by the National Institute of Standards and Technology (NIST) of the U.S. Department of Commerce, was used. NIST was utilized as it provides an easy-to-use repository of checklists that are validated in terms of quality and seasonableness.

It proved to be challenging to identify a checklist suitable for custom-Linux distributions as the majority of checklists are targeting specific flavors of Linux distributions. Nevertheless, through NIST I was able to identify certified vendors that provide checklists of high-quality, and by further exploring them, I located a Benchmark suitable for independent Linux distributions generated by the Center of Internet Security (CIS). Thus, CIS Distribution Independent Linux Benchmark version 2.0.0 released in July 2019 was selected as the baseline of accurately quantifying the security configuration level of a given custom Linux-based system.

SQ3. What information is needed to generate a more secure configuration tailored to a specific custom-Linux based embedded system?

In order to determine the information that will be utilized to assess the configuration setup of an embedded/IoT device and improve the security of its configuration setup while its functionality does not get disturbed, we need to collect all its configuration settings.

The main axes from which these data should be collected from:

- Filesystems used
- Existence of partitions and their special options
- Processes
- Access Control & Warning Banners
- Internet & Special Purpose Services
- Service Clients
- Network Parameters & TCP Wrappers
- Firewall Configuration
- System Accounts & Logging Configuration
- Cron, SSH & PAM Configuration
- User Accounts and Environment
- System File Permissions
- User and Group Settings

It is mandatory to check each category of the ones described above and, more specifically, every configuration policy that these categories encircle and might affect security. By checking these policies, valuable information related to whether a specific configuration is considered secure or not can be extracted. More importantly, by monitoring whether the system makes use of the functionality provided to it by a given configuration, one is able to decide whether the specific configuration can be hardened or it should remain as it is, in order to not disturb the system’s functionality. By collecting the information mentioned above, we can select a set of configuration options inside the system that can effectively get hardened without affecting the system’s functionality, thus enabling the generation of a more-secure configuration tailored for the target system.

SQ4. To what extent would the functionality of a system be affected after the hardening of its security?

The goal of the proposed methodology is to leave intact the target system’s functionality even at the cost of its security. The whole system was designed based on that, as mentioned in the 3rd Requirement (see Chapter 3). Essentially, the cL-Monitoring package logs what system configurations are vital for functionality, that way indirectly notifying cL-Hardening to not adjust these configurations even if they are not secure.

The system’s functionality is not to be affected in any case. This statement is backed not only by the design of the proposed mechanism but also from the functionality and assessment results obtained from the case study.

Functionality Assessment of OSBiz X (Subsection 7.3.3) proves that OSBiz X's functionality was not affected by the configuration hardening, which was the desired outcome. Furthermore, the serviceability of OSBiz X also remained the same as before hardening (Subsection 7.3.4).

SQ5. How can we design and develop a prototype of configuration hardening for systems using custom Linux Distributions?

The essence of this thesis was to explore the feasibility of designing and developing a mechanism that can monitor a device, create a profile based on the monitoring procedure, and, based on that profile, generate a hardening shell that can encircle the device without disturbing its functionality.

To effectively create a prototype that will assist in the defense of already deployed embedded/IoT devices, a Literature review was conducted in order to identify the ways in which hardening configuration has been used as such in the past. After reviewing the related work, I identified a gap regarding the ways hardening configuration is utilized in defending those systems.

The design of the proposed mechanism is described in depth in Chapter 3. First, based on the goals of the research, I form the requirements that the proposed mechanism must fulfill. Then, considering the requirements, the proposed system is designed. Initially, a schematic depicting of the concept of the proposed mechanism is formed. Then, the schematic is broken-down in distinct components. These components are: the monitoring process of the target system, the hardening profile generation and finally, the security-level assessment process.

The next step is to select the appropriate tools that will assist the development of the proposed mechanism. Since the goal is to harden a custom Linux based system's configuration, a way to clarify which configuration policies are considered secure must be defined. There are multiple checklists/benchmarks issued by 3rd party organizations, software vendors and governmental authorities related to secure configuration policies of Operating Systems. The selected checklist/benchmark must satisfy two criteria. First, it should include security-related configurations for custom Linux distributions. Second, it must be updated and issued from a well-respected authority, which will guarantee its quality. After carefully examining the available options, CIS's benchmark for independent Linux distributions was selected as the baseline of determining which system configurations need to be hardened to increase the overall system's security by minimizing its attack surface, enable it to cope with unauthorized access and log various events at the same time. Additionally, Yocto is utilized to enable the transformation of the proposed components into actual packages ready for deployment to the target system. Yocto is selected as it is highly customizable and can create packages targeting custom Linux distros independently of their hardware.

Next is the implementation procedure that is based on the design of the proposed mechanism's components and the tools selected. For the implementation phase, several restrictions apply since the target systems are embedded/IoT devices meaning that they function under power and performance limitations while their resources are limited. Considering the above, tools that can effectively be found in most Linux operating systems were selected for the implementation of every component.

First, a monitoring process is utilized to identify whether the security policies proposed by CIS are needed for the system's functionality. After monitoring the system for a time period, in which as many unique use-cases of the system are replicated, the monitoring process generates a system-map. The system map contains every configuration policy that is necessary for the system to function properly. Based on the system's map and the configuration hardening policies suggested by CIS, the proposed mechanism hardens every system configuration except the ones that are necessary for the system's function. Finally, a process to assess the security conformation of the system against CIS's benchmark is implemented. Every process is transformed into a deployable package through Yocto. The details of the proposed mechanism's development are included in Chapter 5.

SQ6. How does the prototype apply at an actual embedded system, and to what extent does the hardened system comply with its security, functionality, and accessibility specifications?

The prototype was deployed upon an OpenScape Business X, an embedded system running on a custom Linux distribution and functioning as a telephony center. Thus, it includes all the characteristics which the proposed prototype was designed and developed for.

After deploying cL-Hardening at OpenScape Business X, the following results were obtained. Security compliance against CIS's Benchmark for custom Linux distributions climbed from 47,1% to 70.8% Subsection 7.3.1). Functionality was tested using both automated and manual tests from the Quality Assurance department of Atos that works on the OpenScape Business X product. All tests gave the result before and after the hardening took place, indicating that its functionality remained the same (Subsection 7.3.3). Finally, its accessibility was tested from the developers who are usually called to remotely service customers'devices. After executing a test list of all the actions needed in order to service a product correctly, the results that came back revealed that its accessibility remained unchanged (Subsection 7.3.4).

Thus, it can be concluded that the prototype did provide a hardening shell to an actual embedded system, increasing its security compliance while not affecting its functionality nor its accessibility. Additionally, the results are so promising that Atos is convinced to embed the hardening-shell formed by the prototype in the next Version of OSBiz X in order to harden it further. More importantly, Atos, by assessing the potential of the mechanism and its results, also considers the creation of a service that will provide such hardening to all embedded/IoT devices, independently of their OS.

8.2 Research Quality

8.2.1 Limitations

There are, however, several limitations that could negatively influence the benefits of the proposed mechanism for embedded devices that use custom Linux distributions.

1. Initially, the proposed mechanism was deployed in each target device separately, ", and it was able to provide fruitful results, but there was a severe issue. The system was unable to know whether the device was already compromised or malfunctioning. Thus, if the target device was hacked, the configuration hardening shell would be generated encircling the vulnerability as well and not taking measures against it. A lot of thought has been put into resolving this issue. To overcome that, a hazard-free environment, represented by the isolated environment of the Quality Assurance, is selected as the place where the hardening shell will be generated and then deployed at each target device of the same type. Nevertheless, while the QA's environment is safe, almost every use-case of the system is replicated. Thus, the hardening shell becomes less restrictive since the mechanism prefers functionality over security.

2. The mechanism is designed to take into consideration as many systems' use-cases as possible and create a hardening shell that does not affect these scenarios. While this ensures that the mechanism will not affect the system's functionality nor accessibility at the same time, the generated hardening shell is more lose than it could be. This is because not every device will

perform every use-case possible; thus, the hardening shell could be more protective if it was created for each device separately.

3. The configuration hardening policies used in the proposed mechanism follow the Security Benchmark for independent Linux distributions issued by CIS. When this benchmark gets updates, the mechanism should also get updated accordingly. For now, a methodology to update the proposed mechanism automatically has not been implemented, meaning that until then, every Benchmark's update should be followed by a manual adjustment of the proposed mechanism.

4. To encircle every custom-Linux distribution, there are four specific key-elements that need to be considered.
   - The bootloader.
   - System architecture. (32bit vs. 64bit architectures)
   - System initialization. (SysV vs. systemd)
   - Package manager

The mechanism should be further improved to recognize the above-mentioned system's elements and adjust the monitoring and the hardening mechanisms based on them since they can affect the mechanism's performance.

5. The proposed mechanism has been designed to generate the less-overhead possible for embedded/IoT devices that use Linux. Nevertheless, it has not been tested against low-performing CPUs.

6. Currently, only custom-Linux distributions are supported while there is an overall rise in the usage of both Linux custom/commercially created distributions for embedded systems.

8.2.2 Internal Validity

The device, used as a case study, is a well-accepted product from the market with more than 150.000 products deployed around the world. The proposed mechanism does affect all OpenScape Business X machines. For the study of the developed mechanism, random OSBiz X machines were selected. To validate that the results obtained were due to the proposed mechanism, the systems that participated in the experiment part of the case-study were not affected by any new patches or did not undergo any sort of update during the monitoring/hardening/assessment period.

8.2.3 External Validity/ Generalizability

The external validity (generalizability) of this research project is debatable. On the one hand, the study is conducted using a case-study, a device that is functioning in the real-world. On the other hand, the implementation of the proposed mechanism was tried out on a single embedded device.

OpenScape Business X is an ideal case-study since it uses the Operating System targeted from the proposed methodology, but since it is just one case, it does present some unique characteristics that prevent the generalization of the obtained results. These characteristics are:

1: CPU - the proposed mechanism might have significant performance differences if deployed in a system with a lower/higher CPU performance

2: System’s Functionality – The results of the proposed mechanism highly depend on the target system’s functionality as well as the system’s initial configuration settings

3: Device’s Specific Characteristics: Device’s system architecture, bootloader, system initialization method and package manager do play an important role on how the prototype functions.

Thus, the generalizability of my results is a matter for further studies.

8.3 Contributions

The research project has made contributions of both scientific and practical value. Previous studies that utilized OS configuration to harden devices did require human intervention in order to generate the required hardening before applying it to the devices. This project investigated a way of automating the procedure and generating an OS hardening configuration solely based on the device’s functionality and capabilities. The collaboration with Atos made it possible to deploy the prototype mechanism into a real-world embedded device. The results obtained were more than promising since the overall device’s security was increased with no observed drawbacks. Based on these results Atos decided to incorporate the proposed hardening into the actual product (OSBiz X). For Atos, this is more than just a successful project since enhancing it and making a service out of this research is perceived as an appealing option due to the state of the deployed embedded systems’ security in combination with the potential observed in this thesis project.

8.4 Future Work

This research can be extended in several directions to further improve the proposed mechanism in order to enlarge the pool of the affected devices and provide enhanced and more tailored hardening for each one of them.

8.4.1 Generate a configuration hardening shell tailored for each target end-device

As mentioned in the Limitations section, since there is no general way to guarantee that a device deployed in the field is gracefully functioning and it has not tampered with, it was decided to generate profile considering every possible use-case of the target device and deploy this configuration to each and every device. While this approach does guarantee that the system will be fully functional, there is a possibility that the system could be further hardened since not all users use every system’s functionality. Thus, there could be cases where if the monitoring-hardening packages were deployed directly to each device, they could further improve its security. The drawback in this scenario is that if the target device is already compromised, then the hardening shell will not affect the mechanism that has compromised the system. A combination with an IDS package could probably assist in the further utilization of the proposed mechanism. Generally, combining the proposed mechanism with other techniques to provide more security to deployed devices is an exciting field for future work.

8.4.2 Automate the generation of cL-Monitoring, Hardening & Assessment packages

Since CIS provides its security Benchmarks in various formats such as Word, Excel, and XML automatically parsing these files and adjusting the cL-Monitoring, cL-Hardening, and cL-Assessment packages to conform with the updated Benchmarks should be considered. That way, the packages will always remain updated, and the human-error aspect of adjusting the packages will be eliminated.

8.4.3 Encircle more Linux-based devices

CIS creates security Benchmarks for custom as well as for commercial Linux distributions. Extending the proposed prototype to encircle the most commonly used Linux distros that are utilized for embedded/IoT devices if a core part of the future work for this research.

8.4.4 Create a reverse-hardening mechanism

A system might start using a new functionality that was not taken into consideration when creating the hardening shell, and it is possible that the hardening shell disrupts this functionality. To prevent these kinds of scenarios from happening, a reverse-hardening mechanism able to detect graceful actions from the system that are blocked due to the hardening shell should be investigated towards adjusting the hardening-shell and making it more lose in such cases.

8.4.5 Combine the proposed mechanism with other technologies

The motivation of this research is ways to defend already deployed devices effectively. Utilizing configuration management is a promising approach. Combining it with other technologies such as intrusion detection systems and malware cleaners could provide even better results.

References

[1] Cisco, "Iot At-a-Glance," 2016. [Online]. Available: https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf.

[2] Population, "Un.org," 2019. [Online]. Available: https://www.un.org/en/sections/issues-depth/population/.

[3] S. Ravi, S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in Embedded Systems: Design Challenges," *ACM Trans. Embed. Comput. Syst.,* vol. 3, pp. 461-491, 8, 2004.

[4] E. A. Security, "Owasp.org," 2019. [Online]. Available: https://www.owasp.org/index.php/OWASP_Embedded_Application_Security

[5] P. Kocher, R. Lee, G. McGraw, A. Raghunathan and S. Ravi, "Security as a new dimension in embedded system design," Proceedings. 41st Design Automation Conference, 2004., San Diego, CA, USA, pp. 753-760, 2004

[6] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer,* vol. 50, pp. 80-84, 1, 2017.

[7] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *Proceedings of the 26th USENIX Conference on Security Symposium*, Berkeley, 2017.

[8] H. Sinanović and S. Mrdovic, "Analysis of Mirai malicious software," in 2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2017.

[9] E. -. I. Porat, "Eugdpr.org," 2019. [Online]. Available: https://eugdpr.org/.

[10] R. Román-Castro, J. López and S. Gritzalis, "Evolution and Trends in IoT Security," *Computer,* vol. 51, pp. 16-25, 7, 2018.

[11] eclipse, "IOT DEVELOPER SURVEY RESULTS," 2018. [Online]. Available: https://iot.eclipse.org/resources/iot-developer-survey/iot-developer-survey-2018.pdf.

[12] M. Vai, B. Nahill, J. Kramer, M. Geis, D. Utin, D. Whelihan, and R. Khazan, "Secure architecture for embedded systems," in *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, 2015.

[13] O. Qingyu, L. Fang and H. Kai, "High-Security System Primitive for Embedded Systems," in *2009 International Conference on Multimedia Information Networking and Security*, 2009.

[14] M. Ahmad Jan, F. Khan, M. Alam, and M. Usman, "A payload-based mutual authentication scheme for Internet of Things," *Future Generation Computer Systems,* 9 2017.

[15] M. Usman and S. Khan, "SIT: A Lightweight Encryption Algorithm for Secure Internet of Things," *International Journal of Advanced Computer Science and Applications,* vol. 8, 2, 2017.

[16] M. Tausif, J. Ferzund, S. Jabbar and R. Shahzadi, "Towards Designing Efficient Lightweight Ciphers for Internet of Things," *TIIS,* vol. 11, pp. 4006-4024, 2017.

[17] B. Bogaz Zarpelão, R. Miani, C. Kawakani, and S. Alvarenga, "A Survey of Intrusion Detection in Internet of Things," *Journal of Network and Computer Applications,* 2 2017.

[18] H. Tsunoda and G. Keeni, "Societal Model for Securing Internet of Things," 2016.

[19] F. S. Report, "IoT Privacy & Security in a Connected World," 2015 . [Online]. Available: https://www.ftc.gov/system/files/documents/reports/federal-trade-commission-staff-report-november-2013-workshop-entitled-internet-things-privacy/150127iotrpt.pdf.

[20] H. Kwon, S. Lee, J. Choi, and B. Chung, "Mitigation mechanism against in-vehicle network intrusion by reconfiguring ECU and disabling attack packet," in *2018 International Conference on Information Technology (InCIT)*, 2018.

[21] S. Gürgens, C. Rudolph, A. Maña, and S. Nadjm-Tehrani, "Security engineering for embedded systems - The SecFutur vision," *ACM International Conference Proceeding Series,* 9 2010.

[22] M. Kowtko, "Securing our nation and protecting privacy," in 2011 IEEE Long Island Systems, Applications, and Technology Conference, 2011.

[23] A. Oram and J. Viega, Beautiful Security - Leading Security Experts Explain How They Think, 2009, pp. 50-61.

[24] T. a. Touhill, Cybersecurity For Executives, Wiley-AIChE, 2014.

[25] D. P. H. R. E. Möller, Guide to automotive connectivity and cybersecurity, Springer International Publishing, 2019.

[26] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-scale IoT Exploitations," *IEEE Communications Surveys & Tutorials,* 4 2019.

[27] J. Wurm, K. Hoang, O. Arias, A. Sadeghi, and Y. Jin, "Security analysis on consumer and industrial IoT devices," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016.

[28] C. Konstantinou and M. Maniatakos, "Impact of firmware modification attacks on power systems field devices," 2015.

[29] L. Markowsky and G. Markowsky, "Scanning for vulnerable devices in the Internet of Things," in 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2015.

[30] V. Sachidananda, S. Siboni, A. Shabtai, J. Toh, S. Bhairav, and Y. Elovici, "Let the Cat Out of the Bag: A Holistic Approach Towards Security Analysis of the Internet of Things," in *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security*, New York, NY, USA, 2017.

[31] A. Tekeoglu, and A. S. Tosun, "Investigating Security and Privacy of a Cloud-Based Wireless IP Camera: NetCam," in *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, 2015.

[32] M. Qabulio, Y. A. Malkani and A. Keerio, "A framework for securing mobile wireless sensor networks against physical attacks," in *2016 International Conference on Emerging Technologies (ICET)*, 2016.

[33] C. Schuett, J. Butts, and S. Dunlap, "An evaluation of modification attacks on programmable logic controllers," *International Journal of Critical Infrastructure Protection,* vol. 7, pp. 61-68, 2014.

[34] A. Biryukov, D. Dinu, and Y. Le Corre, "Side-Channel Attacks Meet Secure Network Protocols," in *Applied Cryptography and Network Security*, Cham, 2017.

[35] Y. Li and Y. Wang, "False Data Injection Attacks with Incomplete Network Topology Information in Smart Grid," *IEEE Access,* vol. PP, pp. 1-1, 12, 2018.

[36] T. Bonaci, L. Bushnell, and R. Poovendran, "Node capture attacks in wireless sensor networks: A system theoretic approach," in *49th IEEE Conference on Decision and Control (CDC)*, 2010.

[37] M. A. Jan, P. Nanda, X. He, Z. Tan and R. P. Liu, "A Robust Authentication Scheme for Observing Resources in the Internet of Things Environment," in *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, 2014.

[38] M. S. Hossain, G. Muhammad, S. M. M. Rahman, W. Abdul, A. Alelaiwi, and A. Alamri, "Toward end-to-end biomet rics-based security for IoT infrastructure," *IEEE Wireless Communications,* vol. 23, pp. 44-51, 10, 2016.

[39] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "FlowFence: Practical Data Protection for Emerging IoT Application Frameworks," in *25th USENIX Security Symposium (USENIX Security 16)*, Austin, 2016.

[40] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, "Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares," 2014.

[41] B. Wei, G. Liao, W. Li, and Z. Gong, "A Practical One-Time File Encryption Protocol for IoT Devices," in 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), 2017.

[42] A. Furfaro, L. Argento, A. Parise, and A. Piccolo, "Using virtual environments for the assessment of cybersecurity issues in IoT scenarios," *Simulation Modelling Practice and Theory,* 10 2016.

[43] Y. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "IoTPOT: A novel honeypot for revealing current IoT threats," *Journal of Information Processing,* vol. 24, pp. 522-533, 5, 2016.

[44] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "ProfilIoT: A Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis," in *Proceedings of the Symposium on Applied Computing*, New York, NY, USA, 2017.

[45] S. Raza, L. Wallgren, and T. Voigt, "SVELTE: Real-time Intrusion Detection in the Internet of Things," *Ad Hoc Netw.,* vol. 11, pp. 2661-2674, 11, 2013.

[46] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino, "Kalis — A System for Knowledge-Driven Adaptable Intrusion Detection for the Internet of Things," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017.

[47] V. H. Meshram and A. B. Sasankar, "Security in Embedded Systems : Vulnerabilities , Pigeonholing of Attacks and Countermeasures," 2016.

[48] K. M. A. Alheeti, S. Ehsan, and K. D. McDonald-Maier, "An Assessment of Recent Attacks on Specific Embedded Systems," in *2014 Fifth International Conference on Emerging Security Technologies*, 2014.

[49] D. Papp, Z. Ma and L. Buttyan, "Embedded systems security: Threats, vulnerabilities, and attack taxonomy," 2015.

[50] J. Holcomb, "Auditing cyber security configuration for control system applications," 2009 IEEE Conference on Technologies for Homeland Security, Boston, MA, 2009.

[51] C. Kuo and C. Yang, "Security Design for Configuration Management of Android Devices," in *2015 IEEE 39th Annual Computer Software and Applications Conference*, 2015.

[52] B. Chung, J. Kim, and Y. Jeon, "On-demand security configuration for IoT devices," in *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, 2016.

[53] YoctoProject, "Mega Manual," 2019. [Online]. Available: https://www.yoctoproject.org/docs/current/mega-manual/mega-manual.html.

[54] Security Configuration Checklists for Commercial IT Products", NIST, 2020. [Online]. Available: https://www.nist.gov/programs-projects/security-configuration-checklists-commercial-it-products.

[55] "CIS", CIS, 2020. [Online]. Available: https://www.cisecurity.org/.

[56] C. f. I. Security, "Securing Distribution Independent Linux," 2019. [Online]. Available: https://www.cisecurity.org/benchmark/distribution_independent_linux/.

[57] "Master Thesis Project - Creating a Software Security Layer for Embedded Devices", github, 2019. [Online]. Available: https://github.com/illarios/MT-TU_A

[58] OpenScape Business V2, OpenScape Business X3/X5/X8 Service Documentation. Unify Software and Solutions GmbH & Co. KG 20/09/2019.

[59] OpenScape Business V2, Administrator Documentation. Unify Software and Solutions GmbH & Co. KG 20/09/2019.