University of Thessaly

Faculty of Engineering

Department of Electrical & Computer Engineering

Auditing and extending security features of IoT platforms

Diploma Thesis

SOTIRIOS EVANGELOU

Supervisor

Michael Vasilakopoulos

Associate Professor

Volos, July 2020

University of Thessaly
Faculty of Engineering
Department of Electrical & Computer Engineering

Auditing and extending security features of IoT platforms

Diploma Thesis

SOTIRIOS EVANGELOU

Supervising committee

Supervisor
Michael Vasilakopoulos
Associate Professor

Co-supervisor
Christos Sotiriou
Associate Professor

Co-supervisor
George Thanos
Laboratory Teaching Staff
member

Volos, July 2020

Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Έλεγχος και Επέκταση Χαρακτηριστικών Ασφάλειας Πλατφορμών Διαδικτύου των Πραγμάτων (IoT)

Διπλωματική Εργασία

ΣΩΤΗΡΙΟΥ ΕΥΑΓΓΕΛΟΥ

Επιβλέπων
Μιχαήλ Βασιλακόπουλος
Αναπληρωτής Καθηγητής

Βόλος, Ιούλιος 2020

Abstract

The Internet of Things and "Smart Everything" trend is a reality that is progressively becoming a part of our daily lives. Consequently, there is a gradual increase in the development of IoT applications and platforms that attempt to make use of the virtually infinite possibilities and benefits the Internet of Things can offer. However, the connection of billions of - usually inherently insecure - devices in a network, paired with the lack of a clear security framework for the development of Internet of Things devices and platforms has widened the attack surface of these systems leading to them being targeted by malicious actors. In this thesis, we first introduce the reader to the various aspects of the problem, explore related research and give real examples of impactful IoT cyberattacks that have taken place in the past. Then, we compile a security auditing methodology for building and maintaining a secure Internet of Things Platform, including asset taxonomy and definition of the security requirements and countermeasures for every asset. Lastly, we focus on SYNAISTHISI, a research IoT and application enabler platform, developed by NCSR 'Demokritos'. We apply the proposed framework in this case, explore vulnerabilities, gaps and potential insecure factors, perform proof of concept attacks, propose mitigations and good practices and in some of the cases incorporate them in the platform.

Keywords

Internet of Things, IoT, Platform, Security, Vulnerabilities

Περίληψη

Το διαδίκτυο των πραγμάτων (Internet of Things - IoT) και η εμφάνιση των έξυπνων συσκευών είναι μια τάση που ολοένα γίνεται μέρος της καθημερινής μας ζωής. Συνεπώς, υπάρχει μια σταδιακή αύξηση στην ανάπτυξη IoT συσκευών και εφαρμογών που επιχειρούν να επωφεληθούν από τις θεωρητικά άπειρες ευκαιριες και πλεονεκτήματα που μπορεί να προσφέρει το Διαδίκτυο των Πραγμάτων. Ωστόσο, η σύνδεση δισεκατομμυρίων - συχνά από κατασκευής ανασφαλών - συσκευών στο διαδίκτυο, παράλληλα με την έλλειψη κάποιας διακριτής μεθοδολογίας για την ασφαλή ανάπτυξη εφαρμογών και πλατφορμών για IoT έχει οδηγήσει στη διεύρυνση της επιφάνειας επιθέσεων αυτών των συστημάτων, κάνοντας τα στόχους κακόβουλων πρακτόρων. Σε αυτή την εργασία, αρχικά εισάγουμε τον αναγνώστη στις διάφορες πλευρές του προβλήματος, διερευνούμε σχετική υπάρχουσα βιβλιογραφία και παρουσιάζουμε σημάντικες IoT κυβερνοεπιθέσεις που είχαν μεγάλο αντίκτυπο στο παρελθόν. Στη συνέχεια, ορίζουμε μια διακριτή μεθοδολογία για τον έλεγχο ασφαλείας πλατφορμών διαδικτύου των πραγμάτων, περιλαμβάνοντας μια ταξινομία των μελών ενός τυπικού IoT οικοσυστήματος και ορίζουμε τις προϋποθέσεις ασφαλείας που θα πρέπει να τηρούνται όπως και μεθόδους για την επίτευξη τους. Τέλος, επικεντρώνουμε το ενδιαφέρον μας στην ερευνητική πλατφόρμα "ΣΥΝΑΙΣΘΗΣΗ", την οποία αναπτύσσει το ΕΚΕΦΕ 'Δημόκριτος'. Εφαρμόζουμε την προτεινόμενη μεθοδολογία, ερευνούμε αδυναμίες, κενά ασφαλείας και πιθανούς ανασφαλείς παράγοντες, πραγματοποιούμε πειραματικές επιθέσεις και προτείνουμε λύσεις και καλές πρακτικές, ενσωματώνοντας κάποιες από αυτές προγραμματιστικά στην πλατφόρμα.

Λέξεις Κλειδιά

Διαδικτυο των πραγμάτων, Κυβερνοασφάλεια, Πλατφόρμα, Αδυναμίες

Στους γονείς μου, Κωνσταντίνο και Σοφία.

Acknowledgements

First and foremost, I would like to thank Prof. Michael Vasilakopoulos for the supervision of this diploma thesis and collaboration throughout the whole procedure, as well as the co-supervisors Prof. Christos Sotiriou and Prof. George Thanos. I would also like to deeply thank Dr. Charilaos Akasiadis and Dr. Constantine D. Spyropoulos for trusting me to collaborate and contribute in the SYNAISTHISI project of IIT, NCSR 'Demokritos', and continuously providing aid whenever it was needed. I would like to express my deep gratitude to my parents, Konstantinos and Sofia, who have always been providing me with unconditional help, love, support and motivation to be the best I can be. Finally, I would like to thank all of my professors, staff and fellow students of University of Thessaly for the skills, knowledge and great memories that I will carry forever with me after graduating.

Preface

The primary purpose of this thesis is to introduce a security auditing methodology-framework for Internet of Things platforms. The motivation for this was initially the exploration of security solutions for the SYNAISTHISI IoT platform, but for the purposes of a diploma thesis a higher level approach was adopted in order to generalise IoT ecosystems and enhance their security aspect with a focus on IoT platforms. The SYNAISTHISI platform belongs to the Institute of Informatics and Telecommunications, part of the National Centre for Scientific Research "Demokritos". The supervisors for the project of SYNAISTHISI are Drs. Charilaos Akasiadis, Constantine D. Spyropoulos. The research was conducted in the University of Thessaly in Volos, and the building of IIT, NCSR 'Demokritos' in Agia Paraskevi, Athens, during the months between January and July, 2020.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

In the first chapter of this thesis, the reader is introduced to the Internet of Things world. We discuss what the Internet of Things is, we explore the IoT impact through interesting statistics and numbers and we take a first step into the Cybersecurity aspect. Finally, we provide a brief description of the content organization of this thesis before moving to the main body chapters.

1.1 Introduction to the Internet of Things

The "birth" of the Internet of Things can be assumed to be the years 2008-2009. In that period, as Cisco IBSG [22] estimates, the number of Internet connected devices worldwide surpassed the world human population for the first time, initiating the so-called Fourth Industrial Revolution. Rephrasing IEEE's definition [17] in simpler terms, the Internet of Things is a complex network leveraging communication protocols to interconnect "things" , uniquely identifiable programmable devices with physical sensing and/or actuation capabilities that offer services anywhere and anytime using intelligent interfaces. They combine the physical world with the cyber world, usually receiving data or taking actions in the physical world, processing these data with programs running in cloud infrastructures, and exchanging data globally leveraging network protocols.

The benefits that such solutions can offer are virtually infinite, as they are being used in a wide range of use-cases including including healthcare [23, 36], fitness [56], manufacturing [76, 75], and agriculture industry [65, 44] among others, and ideas that would be previously deemed far-fetched and futuristic, such as autonomous self driving cars [19] and smart cities [46], are now realized.

1.2 Numbers regarding the Internet of Things

The numbers and statistics surrounding the Internet of Things are impressive, and can serve as a metric to how much of an impact they have in the modern life. According to IoT Analytics [42], in the end of 2019 the number of active IoT devices was estimated to be 9.5 billion without counting mobile devices like smartphones and laptops or IoT devices bought in the past, a number already bigger than the world population and continuously rising. During the period from 2017 to 2025 the Compound Annual Growth Rate (CAGR) for the IoT connections is is estimated to be at 17%

reaching the 25 billion point in 2025 and a 1.1 Trillion USD global market revenue according to FICCI/EY [62] (1.6 trillion USD according to Statista). The total data volume by 2025 is projected to be 79.4 ZettaBytes (ZB) [52].

PwC [61] also reports in a 2019 survey that 93% of the executives asked, believe that the Internet of Things is much more beneficial than risky, with 70% of them reporting that their company has IoT projects live or in development. Thus, the number of IoT products is also gradually rising, with publicly known Internet of Things platforms in particular counting 620 in 2019 [41].

These are just some of the vast amount of statistics and numbers about the Internet of Things, that can give us a view of their significance to the modern world. The adoption and usage rates are continuously rising and these devices and platforms are more and more used by companies to reinforce important aspects of the production including automation, monitoring, data analytics and many more, towards more sustainable and cost-effective processes.

1.3   Security Challenges in the Internet of Things Ecosystem

As with every disruptive technology, there are some challenges regarding the global adoption of the Internet of Things. Being low-cost with limited resources might remove the cost factor, that is most of the times the deal breaker for companies and individuals, but it comes with a security compromise cost in most of the situations. The PwC research mentioned earlier [61] reports a 48% cybersecurity concern and a 46% privacy concern among the 1000 executives asked, while 61% of the executives reporting active IoT projects are working to solve security-related issues with their platforms.

Throughout the years, multiple vulnerabilities and security incidents have affected the Internet of Things ecosystem, and it is evident that the companies, developers and researchers are now trying to remediate these weaknesses and improve the security posture of these products. Research on IoT Cybersecurity covers a wide range of aspects, including the incorporation of security in the development life cycle, auditing methodologies, surveys on attacks and common vulnerabilities, studies on good practices, as well as physical, hardware, software, and network security [24, 49].

In this work, we examine past research in the domain of security for IoT, and present a high-level taxonomy of the assets that compose a typical IoT ecosystem. Then, we describe methods for evaluating the security posture by focusing in each of the identified assets, exploring insecure factors to assess desired requirements and highlight the aspects that could be reinforced. We cover a variety of ecosystem entities, since each of the assets performs differently in the ecosystem, and provides diverse security propositions due to the different nature of the various technology stacks incorporated. We establish a security baseline for each asset and collectively for the ecosystem itself, with a focus on the security features of the IoT platform components.

1.4   Content organization

The thesis is organised into four main chapters excluding the introduction (Chapter 1) and conclusion (Chapter 6).

The second chapter explores previous research that has contributed in this area of study. The papers and articles that will be referenced are ones that contain ideas, techniques and facts that will be used throughout the thesis, or are worth mentioning so that the reader develops context awareness. The chapter is split into sub-chapters indicating the particular subarea that each paper in question focuses into.

The third chapter stresses the reality of the IoT security problem and the real world impact it has in the economic, social and physical factor. Significant attacks on IoT systems that have happened in the past are described, including the techniques with which they were performed and their significant aftermath.

The fourth chapter considers a typical Internet of Things ecosystem and is organised into two sub-chapters. The first presents an asset taxonomy of Internet of Things ecosystems that helps generalise the IoT ecosystems and approach solutions that can potentially be applied to a variety of different platforms thus providing an auditing methodology. The second sub-chapter focuses on each of the assets, and attempts to define the security requirements that need to be in place, proposing some controls, countermeasures and good practices that the appropriate stakeholders can apply to make their product more secure.

The fifth chapter uses the defined methodology of the previous chapter in order to explore a real world case, that of the research IoT platform SYNAISTHISI [3, 55]. By applying the framework, gaps, vulnerabilities and insecure factors are located, solutions and good practices are proposed, and some of them are programmatically incorporated in the project. In that way, the defined framework is validated presenting a real use-case and benefiting a fully functional IoT platform.

Lastly, in the sixth chapter we conclude our ideas, the benefits and impact that a defined security auditing methodology has at the development of secure products, and potential future work steps.

Chapter 2

Related Research

The Cybersecurity aspect of the Internet of Things is a well-researched subject. In this chapter, we collect and discuss interesting research in the scope of the IoT security domain in order to gain useful insights, knowledge as well as identify less researched sub-areas where this thesis could contribute knowledge. Finally, we explore research addressing the use of disruptive technologies such as Blockchain and Machine Learning in the IoT security domain.

2.1   Security Guidelines in Internet of Things

A good starting point for researching the security of an IoT ecosystem is establishing a baseline set of requirements and good practices. A product should not be deemed ready to enter the market unless it has incorporated this baseline. In particular, there are a lot of articles, guides and whitepapers describing such sets of requirements and recommendations, and we proceed to describe some of the most important among them.

Starting off, ENISA, the European Union Agency for Cybersecurity offers two significant reports providing baseline recommendations and good practices, attempting, in that way, to provide guidelines for IoT product development, maintenance and EOL (end of life) management. In both of the reports, a detailed asset and threat taxonomy is presented, with a special emphasis in the most critical of the assets and threats, along with the impact and the stakeholders that they affect. The first report[24] also performs a gap analysis, and offers good practices and recommendations. The recommendations are mostly non-technical and serve a development, maintenance and management strategy purpose. The good practices report[25] of 2019 emphasises on the incorporation of security in the Software Development Life Cycle of Internet of Things products, firstly analysing each cycle phase. Then, it presents good practices that a company can incorporate in the development lifecycle, categorising them into three types:

- the **People** which affect all stakeholders and phases.

- the **Processes** that affect the mechanisms surrounding the software project's environment.

- the **Technologies** that consist of technical countermeasures and development good practices.

Another whitepaper summarizing security guidelines that affect the Internet of things product development, is Infocomm Media Development Authority's (IMDA) IoT Cybersecurity Guide[13]. The guide offers suggestions for the implementation and operational phase of the product and two checklists, a threat modelling checklist and a vendor disclosure checklist. Using these checklists potential vendors and developers can perform a self-assessment of the security posture of the product in development, thus providing a baseline of applied controls which suggest that the product is secure and market-ready.

Kamalrudin et al.[35] present a library for eliciting security requirements. The methodology is a way for requirements engineers to break an IoT application into components from the domain of the application (Industrial, Smart City etc.) to business applications (monitoring, access control etc.) and lastly IoT attributes and technologies that will be used in order to derive security requirements for its development. The paper proves the concept of the library by applying it to a smart parking application scenario.

These works offer to vendors a defined set of requirements and guidelines that should be applied to a newly developed IoT product, so that it operates securely, adhering to privacy and safety needs. This thesis combines the identification of assets, the security requirement assessment, and insecurity exploration, as well as a proposition of measures to address such insecurities. We approach the IoT ecosystem from a higher-level cyberphysical system's viewpoint and address all the types of co-existing stakeholders, including developers, system administrators, deployment infrastructure and end-users. We also focus on standard and AI-related tools and programming techniques that can be applied by the responsible teams to improve the 'marginal' security of each asset.

2.2   Surveys on Vulnerabilities and IoT Attack Surface

In the cybersecurity domain, the "attack surface" is, in simple terms, the sum of insecure entry points or vectors that a malicious actor could use in order to enter or attack an environment. There is existing research exploring the attack surface on the Internet of Things ecosystem, surveying vulnerabilities and common exploitations that lead to cyberattacks.

Rizvi et al. [63], identified the attack surface of IoT networks. Initially, they present a decomposition of the network into trust zones, and a categorization of existing devices into IoT domains (Finance, Home, Wellness etc.). Then, 14 common vulnerabilities are mapped to common attacks such as Denial of Service (DOS), Ransomware and SCADA Trojan horses. Finally, the paper proposes security controls to mitigate these weak spots and concludes with a detailed map of IoT domains with the vulnerabilities, attacks and security controls associated with these domains.

Dabbagh et al.[18] firstly analyse the Internet of Things security challenges and present abstract security requirements such as the CIA triad - *Confidentiality, Integrity, Availability*. Then, they proceed to decompose the IoT architecture into three basic domains, the cloud domain containing the IoT applications and services, the sensing domain containing the devices and their communication means, and the fog domain including everything that stands between the sensing and cloud domains. The chapter delves into security vulnerabilities and common attacks regarding these three

domains, and proposes countermeasures for each of the cases explored. The particular research is not generalised and does not cover all possible attacks, but takes a detailed approach in virtual machine related attacks in the cloud domain, and interference-tampering of the communication of sensors in the sensing domain, both rarely documented in similar IoT security research.

Serpanos and Wolf[67] examine security in layers. First, they are assessing the security of IoT Systems, exploring software and hardware security controls applied to them as well as anti-tampering physical security techniques. Then, as far as network security is concerned, the chapter emphasizes on network encryption, authentication and secure routing, as well as key management for the encryption mechanisms. Denial of service (DoS) and distributed denial of service (DDoS) techniques are also referenced as they have been a popular malicious scenario for IoT devices. Lastly, the chapter stresses the application layer security and proposes security-by-design and runtime monitoring (model-based and profile-based) of the new Internet of Things products. The next chapter of the same book[68] focuses on the security testing aspect, where fuzzing tools and techniques are explored in different situations (White-box, Black-box). The authors present a proof of concept, fuzzing the Modbus application protocol, used in Industrial IOT communications.

Finally, Neshenko et al.[49], perform an exhaustive - as the title presumes - survey on IoT vulnerability research. The paper successfully sums up a variety of research focused into the attack surface of IoT architectures, and presents a taxonomy of the collected results depending on different aspects - *Layers, Impact, Attacks, Countermeasures, Situational Awareness Capabilities* - and maps the corresponding research to these classifications. An empirical overview of the vulnerabilities is, then, presented and the survey concludes with a presentation of the most important security challenges, paired with possible future initiatives to fight against each security challenge.

These threat analyses cover a wide range of the possible threats concerning IoT infrastructures and products. Taking them into consideration, in Section 3 we create a picture of the baseline security requirements, and consequently the security measures that need to be in place in order to protect IoT deployments from such threats.

2.3   Security Evaluation on Existing IoT Solutions

The Internet of Things is always progressing but until now there has been a huge variety of products and solutions already in market. Therefore, it is useful to approach the security evaluation of existing IoT platforms, frameworks, devices, products and protocols. By reviewing the security controls in products currently used or sold across the world, we can both see the common trends in security and their impact, as well as state more loudly the need for better and more organised security in this technology domain.

Ammar et al.[7] present a survey on the architecture, hardware and software specifications and security features regarding authentication, authorization and secure communications in multiple IoT frameworks developed by popular vendors. These products are AWS IoT (Amazon), ARMbed (ARM), AzureIoT (Microsoft), Brillo/Weave (Google), Calvin (Ericsson), Homekit (Apple), Kura (Eclipse) and Smart Things (Samsung). Looking at the conclusions, the authors compare the chosen security controls and discover trends such as the universal use of TLS/SSL and the popularity of

AES cryptography and X.509 certificates.

Alrawi et al.[5] examine the security of IoT products created for the Home domain, including smart televisions, smart bulbs and smart doorbels. After decomposing the home IoT deployment into 4 parts, namely the device, the mobile application, the cloud endpoint and the communications, they proceed to explore existing research to identify the attack vectors of each part, and lastly cross-reference them to a wide variety of home IoT products. Using the CVSS (Common Vulnerability Scoring System) standard [45] scoring system and associating every product with known CVEs (Common Vulnerabilities and Exposures), which are basically publicly stated vulnerabilities, they evaluate each product's security posture.

As with the home domain, there is also research regarding other domains such as the Healthcare domain[57], and the Industrial IoT domain [26][1].

2.4   Artificial Intelligence-based Approaches

Artificial Intelligence and and Learning techniques are employed in a variety of use cases and security is one of them. Although the manual labour and configuration is not yet fully replaceable in the aspect of Cybersecurity, there is a number of security aspects where AI techniques shine and provide great results contributing to the security of an IoT system.

Ahanger [2] evaluates the use of artificial neural networks (ANNs) as an intrusion detection system to combat distributed denial of service (DDoS) attacks. A multilayer perceptron neural network is used (MLP) to create an anomaly based network Intrusion detection system. The experimentation in a custom IoT deployment shows a great detection rate of 99% while also providing a low false positive rate (malicious traffic falsely identified as normal) which is one of the most significant goals of an IDS. Other than neural networks, techniques that perform well into network intrusion detection are kNN, Random Forest and Support Vector Machines [79][47].

Local malware scanning is also an aspect where artificial intelligence techniques can be a great supplement to existing signature based techniques. While signature based approaches can protect against known malware, the behavioral approach of A.I. techniqes can often protect against new or unknown malware. Xiao et al. [74] initially propose the use of kNN and Random Forest classification algorithms on collected traffic data to identify malware with a high accuracy on standard datasets. The lack of resources in some devices is also addressed where the solution is to collect application traces locally and appoint the model training and predicting to a capable and trusted external server.

Xiao et al. [74] also report two unique use-cases where learning techniques can play a significant role in IoT devices. The first is for secure IoT offloading as a method to combat jamming and MITM attacks. Using reinforcement learning, and specifically the Q-learning technique, the model takes into consideration the task priority, channel bandwidth, gain and jamming power in order to decide on offloading policies according to a Q-value that indicates the long term reward from choosing this policy. Using this approach, the device can choose optimal offloading channels and subbands in order to avoid interferences and jamming as well as spoofing attacks. Convolutional neural networks can also be used for the same purpose but they require more computational resources than

Q-learning. The second use case addresses authentication using learning methods in order to avoid spoofing attacks. Using physical layer indicators such as the received signal strength (RSS) or the channel state information (CSI), learning techniques are able to exploit the indicators' connection to spatial characteristics in order to lure out connections that are initiated from outside a threshold proximity reducing spoofing rates. Other than Q-learning, both supervised (distributed Frank Wolfe - dFW and incremental aggregated gradient - IAG) and unsupervised algorithms (Infinite Gaussian Mixture Model) are used. For more resourceful devices, deep neural networks can also be applied to further improve the accuracy rates. IGMM is also reported as a useful algorithm for authentication by fingerprinting in Hammed et al's work [33], where IGMM is used to create fingerprints in IoT devices and validate the credibility of the device by comparing the IGMM result with a value expected depending on the device's nature and shape. On environment changes the model is able to distinguish a normal change from a malicious change.

Machine Learning and essentially supervised learning methods require significant quantities of information, hence the term "Big Data". The information, apart from being the input that will affect a model's decision, also play significant roles into training the model, improving its prediction accuracy and validating its efficiency. In order to leverage the advantages A.I. has to offer we need to establish a way to collect, clean and format these data to use them. Roukounaki et al. [64] present an infrastructure for collecting, storing and analysing big data in IoT systems. They introduce a data collection and actuation layer in the IoT system, with the collection aspect comprised of system and application level probes, and IoT probe registry listing all the used probes, a data routing middleware to route the data to their respective recipients. The actuation aspect includes a Security Policy Enforcement Point where the data collected drive security decisions (e.g. disabling a service or closing a port) and visualization where collected data and analyses are being displayed. The infrastructure also has management agents and management and configuration tools in order to control and manage successfully the infrastructure. The authors finally discuss the security use case of this infrastructure including risk assessments and asset security monitoring.

2.5 Blockchain-oriented approaches

In the final subsection of the chapter we will see some proposed solutions for architecture and security in the Internet of Things that leverage a relatively new technology, Blockchain. We will not use such techniques throughout the thesis, but it is worth to present these solution aspects.
The blockchain is a decentralised peer to peer architecture where each node possesses a shared ledger (chain of blocks) that contains all the transactions that take place in the network. The ledger is essentially a chain of smaller groups of transactions called blocks. This chain is immutable as a single change in a block invalidates the whole chain. The decisions of the system are reached through the consensus validation of a block and the addition of the block to everyone's chain, and that block is proposed by a node that solves a "difficult" computational cryptographic problem called proof. The blockchain by default is trustless as no third parties need to be trusted (e.g. banks in fiat money transactions) and anonymous since each node is associated to only a signature. These astounding benefits come at the expense of memory storage to save the whole ledger, and power

in order to solve the problem (Proof of Work algorithm). These expenses drove a lot of researchers to the argument that the blockchain could not be suitable for Internet of Things deployments as the devices themselves tend to be low on resources, although the decentralised architecture paired with the privacy mindset of Blockchain are indubitable reasons to pursue the unity of the two technologies, especially with the incorporation of smart contracts, software programmable contracts between two or more nodes.

Khan and Salah[37] summarize blockchain features that could be useful in IoT like the huge address space (bigger than IPV6), the Identity of Things and Authentication, Authorization, Privacy and Secure communication factors to emphasize on the blockchain feasibility for this usecase. Also various different blockchain implementation propositions exist in published research. Angin et al. [8] propose a blockchain based architecture for the Internet of Things to raise data security. In this proposition, IoT devices are nodes of a device-level blockchain, and have a hierarchical relationship to "data collectors", capable cloud servers, where the heavy work of the mining and storage expenses of the ledger are offloaded. Wang et al.[73] propose Chainsplitter, a decentralised overlay network between the local networks were devices reside and the cloud servers that holds services and applications. The devices connect to "Blockchain connectors" and the Cloud entry points to "Cloud connectors", and the connectors engage as nodes in a permissioned blockchain network using Byzantine Fault tolerance[15] as a consensus algorithm. The blockchain connectors are the ones creating blocks for the blockchain based on transactions received from the devices. The paper concludes with a test case and an emphasis on the benefits of Chainsplitter for the security of the deployment, along with specifications for the connector implementation.

Both of those propositions are well-thought approaches to the incorporation of blockchain in IoT deployments, but nevertheless are outside the scope of this thesis, which is to develop a security feature auditing and extending methodology that can be applied to typical IoT platforms. As a final note, blockchain approaches, although theoretically alluring, demand to be applied to the platform from the design phase and cannot be added later as a security feature.

Table 2.1: Summary of Related Work Contributions

Research	High Level IoT Model	Asset Taxon.	Threat Taxon.	Management Strategy	Physical Security	Hardware Security	Software Security	AuthN AuthZ	Network Security	Admin Security	A.I. M.L.	Blockchain
[25]	✓	✓	✓	✓	✓		✓	✓	✓	✓		
[24]	✓	✓	✓	✓		✓		✓	✓	✓		
[13]						✓	✓	✓	✓	✓		
[35]				✓			✓	✓	✓			
[63]	✓	✓						✓	✓			
[18]	✓	✓	✓			✓	✓		✓	✓		
[67, 68]	✓		✓		✓				✓			
[49]	✓		✓	✓	✓		✓	✓	✓	✓		
[7]	✓		✓	✓		✓	✓	✓	✓			
[5]	✓	✓		✓	✓		✓	✓	✓			
[57]	✓					✓	✓	✓	✓	✓		
[26]	✓											
[1]	✓	✓						✓	✓	✓	✓	
[2, 79, 47]									✓		✓	
[74]							✓	✓	✓		✓	
[64]	✓					✓	✓	✓	✓	✓	✓	✓
[37]	✓		✓		✓	✓	✓	✓	✓	✓		✓
[8, 73]	✓							✓	✓			✓

Chapter 3

IoT Cyberattacks and Real World Impact

As the number of IoT deployments is rising, so do the vulnerabilities and security incidents worldwide. In this chapter, we will analyse some of the most significant attacks and public vulnerabilities regarding the Internet of Things domain. These attacks affected multiple domains including the economy and privacy, motivated researchers to analyse them and raised the public awareness about the insecurity of the IoT systems.

3.1 Stuxnet

Stuxnet has been labeled as one of the most complex threats created in history, and that is not an overstatement in any way [39]. The worm malware [1] targeted air gapped (physically isolated from the internet) systems and particularly industrial SCADA deployments in Iran's nuclear facilities in Natanz, and it was an active and serious threat in the years between 2009 and 2011.

3.1.1 Outline of the Attack Methodology

Stuxnet was approximately 500 kB in size and targeted Microsoft Windows Operating Systems connected to programmable PLCs specifically running Siemens Step-7 software. The malware had capabilities of propagating to other computer systems in a local area network (LAN) until it found one of the targeted computers controlling uranium centrifuges where it downloaded malicious software that performed two distinct attack routines based on the software used. The first stuxnet version (on S7-417 controllers) performed an overpressure attack in the centrifuges by closing outflow valves, and the second and more advanced one (on S7-315 controllers) changed spinning frequencies in order to progressively cause mechanical errors. Since the Iranian nuclear facilities were air gapped, the virus is suspected to have infected the first victim via an infected USB drive, by a victim engineer or a double agent that had physical access to the facilities. It spread to other computers using more than six methods, mainly shared filesystems, in some cases leveraging

---

[1] self-replicating computer malware that spreads to other computers

zero day vulnerabilities. The propagated malware versions engaged into peer-to-peer updates in case one of the infected computers had access to the internet, and consequently could contact the C&C (command and control) server. The malware was designed with an emphasis on detection avoidance using stolen trusted certificates like Realtek's in order to be trusted from antivirus software. Lastly, Stuxnet infected the telemetry server to present false normal results to the engineers, and incapacitated some of the emergency controls that were in place.

3.1.2   The Impact of Stuxnet

The knowledge level of the attack, the target and the fact that the malware had little chance to work well without a testing facility (indicating that the attack part had access to a similar testing facility) are all indicative of the malicious actor's large-scale capabilities. The Stuxnet attack is assumed to have destroyed about 1000 uranium enrichment centrifuges in Iran's nuclear facilities, and delayed the nuclear program of Iran enough to force a diplomatic agreement on nuclear usage. This is the first known case of a military cyberweapon and widely changed the way that cyberspace is weaponized. Stuxnet also emphasized the magnitude of impact a malware can have and was one of the first and most important cyber-physical attacks in industrial systems in history.

3.2   Mirai Botnet

The Mirai botnet[9] was a botnet of compromised Internet of Things embedded devices that performed significat Distributed Denial Of Service attacks on multiple high-profile targets in 2016. Some of these targets were Dyn, a major DNS provider with clients such as Amazon and Twitter, Krebs on Security, one of the most esteemed cybersecurity blogs, and Deutsche Telecom.

3.2.1   Outline of the Attack Methodology

Interestingly, the source code of the Mirai worm was leaked by its author in an early attempt to disguise himself behind the variants that would be developed, thus we know exactly how Mirai operated. When the malware was installed into a device, it would go into a rapid scanning phase using pseudo-random IPv4 addresses targeting the Telnet service ports on ports 23 and 2323. If it detected a device with an open telnet port it would bruteforce default vendor passwords to attempt to log in and, interestingly, succeeded in a lot of cases. When a new device was infiltrated, the malware would contact a hardcoded Command And Control (C&C) server, making it a bot on the Mirai botnet. A separate loader program would identify the architecture executing architecture-specific malware due to the heterogeinity of internet of things devices that were infected. The malware was not persistent because the binary was deleted after the process initiated, but it would terminate other SSH and Telnet services, as well as other botnet malware such as Qbot.

3.2.2   The Impact of Mirai

The Mirai Botnet was one of the biggest botnets ever with a peak of about 600 thousand infected devices at November 2016. It was also the perpetrator of some of the biggest DDOS attacks in

history, including the attack against Krebs on security blog, which peaked at 620 Gbit/s, and 1.2 Tbit/s against OVHcloud web hosting. What is more serious was the simplicity of the attack which gained access to millions of devices simply by using default credentials, a vulnerability introduced mainly by careless users and administrators. The Mirai botnet attack had also significant economic impact on its victims. For example, Dyn lost about 8% of its customer base following the event of the attack, which was around 14 thousand domains. Finally, it was also the reason for the research for DDOS resistance techniques and real time mitigation, leading to only 5 minutes downtime in a 1.35Tbit/s related DDOS attack (not botnet-based but amplification attack) against Github two years later, that was completely mitigated by Akamai Content Deliver provider.

3.3 CloudPets and other smart Toys

Mirai and Stuxnet were both turned against high-profile targets, but the insecurity of the Internet of Things can severely affect ordinary households as well. An example could be smart toys, toys for children that provide fun features such as voice interaction and communication with parents. A lot of these products have been found to be vulnerable, compromising the privacy of the families that own them and, in some occasions, leaking private sensitive information to the outer world. Cloud Pets were adorable plush toys that were Internet connected and allowed voice message exchange between parents and children via cloud. The cloud voice recordings were found to be accessible without authentication if someone knew the URL address to them, and strong password policies were non-existent, allowing users (parents) to use even 1 digit passwords, making them vulnerable to bruteforcing in a few seconds. This led to a leak of half a million accounts online, and consequently more than two million private voice messages of children and parents. The company was forced to shut down in face of these leaks, but cloudpets are surprisingly still being sold in online stores such as Amazon.

Another significant research[51] was performed by the security firm Mnemonic and the Norwegian Consumer Council (NCC) on the security of four children smart watches. These watches had mobile phone - like capabilities, including GPS locators, SIM cards to enable making or receiving calls, SMS messages and voicemailing. Three of the four assessed smart watches did not disclose user rights and terms to the users and without consent collected personal data. Also, none of them notified the users in a potential change of Privacy Policy. Lastly none of them provided methods for users to delete their accounts and personal data, and did not specify how they would use the personal data collected, nor made clear how and where data are stored. All of them had serious privacy issues, in some cases breaching laws regarding privacy, consumer rights and the General Data Protection Regulation (GDPR) of Europe.

3.4 Security Awareness Study

Mirai, as well as other botnets that have amassed millions of slave machines in the last years leveraging malware (worms) that is automatically spreading to vulnerable targets using simple approaches such as default passwords or attacking publicly known vulnerabilities with known

exploits, that have been possibly patched. Simply having an awareness for cybersecurity and keeping systems updated with the latest security patches goes a long way into being safe from cyberthreats.

In this subsection we aim to use real data in order to address the importance of simple actions towards the avoidance of becoming a target and maintaining secure systems. The data that is used to compute our metric can be collected using Shodan, a global crawler for Internet-connected devices. It scans global IPs, collects information for each IP such as the organization name, location, domain name, open ports, running services, and also attempts to grab the banner [2] of the audited services to learn more specific information, e.g. version, and then map it with specific CVE vulnerabilities.

In this Thesis, by using the Shodan API we were able to collect information about the number of internet connected devices all around the world categorised by country, for the top 200 country results. For keeping the results relevant, countries with population of less than 300,000 people were excluded. The results are presented in figure 3.1 and table 3.1.
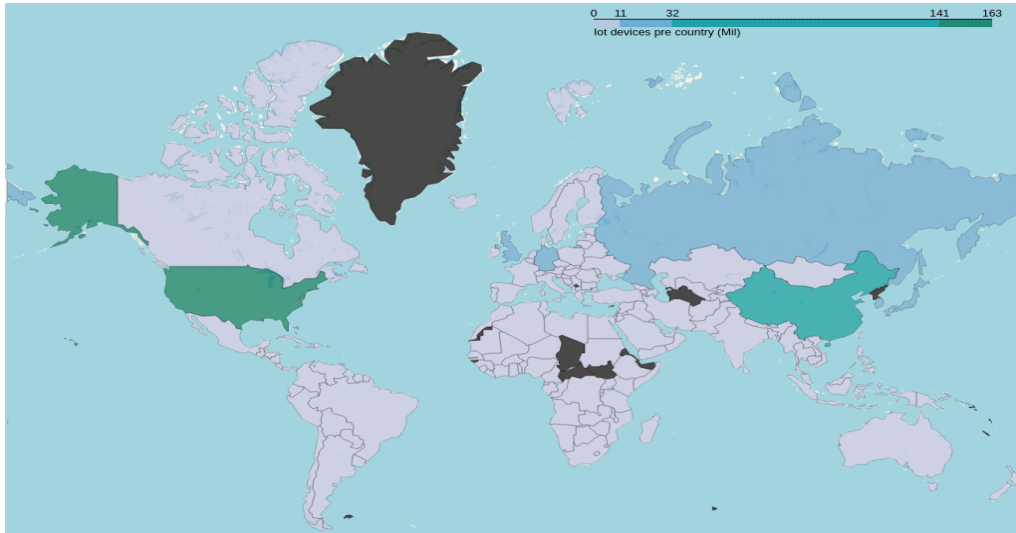
Figure 3.1: Internet Connected Devices per country (Millions)

Since the point of the study is evaluating the security awareness, the vulnerability scanning that Shodan performs was leveraged in order to get the number of devices found vulnerable with specific vulnerabilities with CVE (Common Vulnerabilities and Exposures) identification numbers. After finding the number of vulnerabilities per country, the weighted sum of them was computed for each country using as weights their CVSS score [45], and their exploitability score. From the calculations vulnerabilities with less than 6.0 CVSS score or Local/Physical attack vector were excluded in order to keep only severe and relatively easily remotely exploitable vulnerabilities. In that regard, it was assumed that devices with vulnerabilities in that category would most likely become a cyberattack target because of the ease of exploit and impact that a malicious actor can deliver. The results are presented in figure 3.2 and table 3.2.

Dividing the weighted sum of vulnerabilities per country with the number of internet connected

---

[2]Technique used to gain information about a computer system on a network and the services running on its open ports.

Table 3.1: Top 40 countries with most Internet Connected Devices

0	USA	163965019	10	HKG	7382864	20	SGP	3944996	30	UKR	1806292
1	CHN	35068171	11	ARG	7187785	21	ESP	3819222	31	COL	1701085
2	DEU	27939736	12	ITA	7114985	22	VNM	2917478	32	SAU	1681472
3	GBR	18440278	13	NLD	6527878	23	ZAF	2907405	33	CHE	1599801
4	KOR	11994966	14	AUS	5960419	24	THA	2840353	34	CZE	1351948
5	JPN	11126474	15	IND	5752842	25	IRL	2443539	35	BGR	1219356
6	RUS	11035911	16	POL	5419278	26	ROU	2362859	36	MYS	1141602
7	FRA	9537196	17	TWN	5058123	27	IRN	2041691	37	PRT	1125341
8	BRA	9277937	18	MEX	4734761	28	IDN	1942863	38	AUT	1043767
9	CAN	9200318	19	TUR	4108397	29	SWE	1826714	39	DOM	1036609
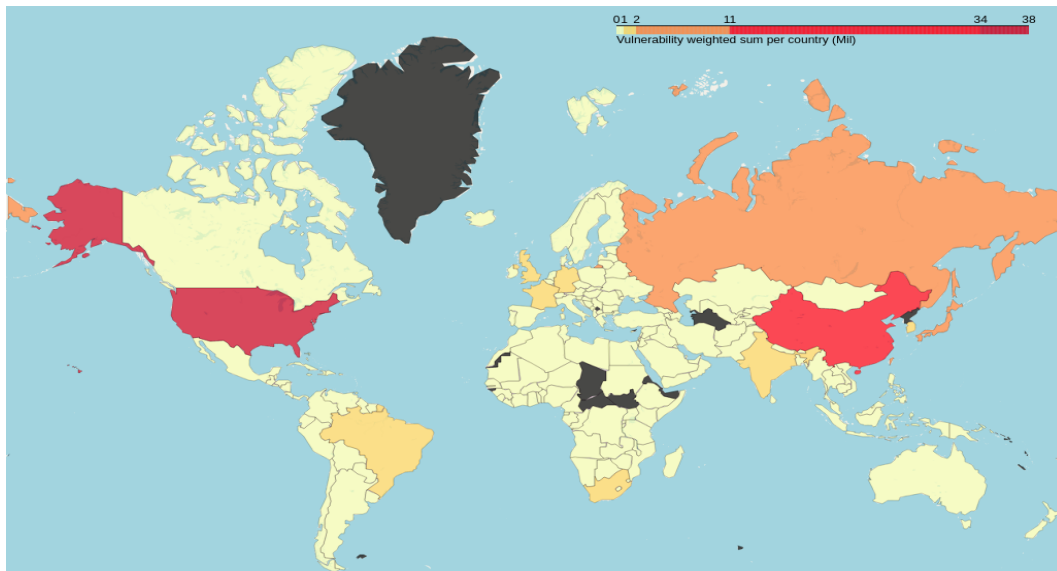
Figure 3.2: Vulnerability weighted sum per country (Millions)

Table 3.2: Top 40 countries by weighted sum of vulnerabilities

0	USA	38228573	10	SGP	999907	20	ESP	506129	30	CHL	186548
1	CHN	11620249	11	GBR	952048	21	MYS	461568	31	IRL	164000
2	HKG	5329115	12	IND	933418	22	NLD	456413	32	SWE	154903
3	JPN	2318747	13	FRA	751357	23	TUR	451199	33	BGR	141578
4	TWN	2015268	14	AUS	697862	24	VNM	407178	34	CZE	141369
5	RUS	1883171	15	ITA	686286	25	ARG	344803	35	EGY	140957
6	KOR	1746187	16	MEX	638291	26	IDN	316612	36	BEL	135994
7	BRA	1277227	17	IRN	613248	27	UKR	281631	37	AUT	124366
8	DEU	1229492	18	THA	537529	28	POL	262077	38	BGD	118454
9	ZAF	1104430	19	CAN	517218	29	COL	190741	39	PER	116814

devices in each of these countries provides us with a metric that can statistically show how updated and secure against harmful remote cyberattacks a country's systems are, and collectively give us and indication of each country's security awareness. For this reason and for the sake of having an intuitive metric name we call the metric LSAR, for Lack of Security Awareness Ratio. High values of LSAR indicate greater density of vulnerable and exploitable devices in a group of devices, deeming that group as a more possible target of malicious actors than one with a smaller LSAR.

$$LSAR = \frac{\sum_i (\#Occurences_i \times CVSSscore_i \times Exploitabilityscore_i)}{\#Devices}$$

where

$$i \in \{\text{CVE-X} | CVSSscore_i > 6.0 \cap Vector_i \notin Local, Physical\}$$

The results for the LSAR metric in the countries above are presented in figure 3.3 and table 3.3, where, surprisingly a lot of countries appear that were not included in the previous findings.
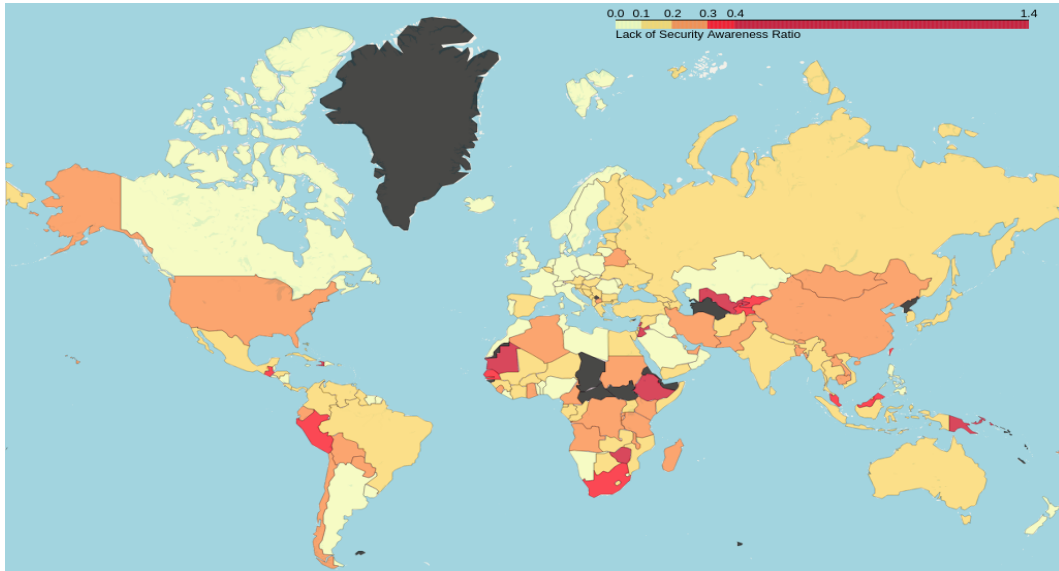
Figure 3.3: LSAR metric per country

Our findigs are including the countries with the biggest LSAR metrics as we defined it, meaning the countries with the least security preparedness against known exploits and remote cyberattacks, hence least security awareness. The point of these findings, though, was to indicate the importance of keeping systems updated and secure against publicly known vulnerabilities, so we need real data of cyberattacks in countries to correlate with and validate the above findings. To validate LSAR, we compare it with results from a survey [14] for the best and worst security in countries. The survey includes data up to March 2020, which is adequately close to data collection from Shodan for the LSAR computation (late April, 2020). In this survey countries are ranked for:

- The percentage mobile devices infected with malware

- The percentage of computers infected with malware

- The number of financial malware attacks

Table 3.3: Top 40 countries by LSAR

0	HTI	1.422832	10	MYS	0.404316	20	MDG	0.310316	30	MKD	0.272767
1	UZB	1.164537	11	TWN	0.398422	21	MNG	0.304405	31	PRY	0.258938
2	ZWE	0.782047	12	PER	0.397818	22	CPV	0.302953	32	BGD	0.258386
3	HKG	0.721822	13	TJK	0.392794	23	IRN	0.300363	33	PAK	0.255805
4	ETH	0.636363	14	ZAF	0.379868	24	BLR	0.298036	34	RWA	0.253722
5	JOR	0.522041	15	SEN	0.372707	25	AGO	0.295866	35	SGP	0.253462
6	PNG	0.455162	16	GTM	0.348438	26	LAO	0.295831	36	BDI	0.253116
7	LBN	0.451086	17	CHN	0.331362	27	KWT	0.283800	37	CHL	0.252933
8	MRT	0.441238	18	SLE	0.323695	28	BOL	0.282629	38	DZA	0.250114
9	KGZ	0.405626	19	BTN	0.321850	29	MWI	0.278312	39	CMR	0.245949

- The percentage of all telnet attacks by originating country

- The percentage of users attacked by cryptominers

- The best-prepared countries for cyber attacks

Combining this survey's results with the shodan findings, 65 countries belong in both of the datasets and thus can be compared. For that reason, we explore the correlation between the LSAR feature and the features introduced by the comparitech survey. The results are the following:
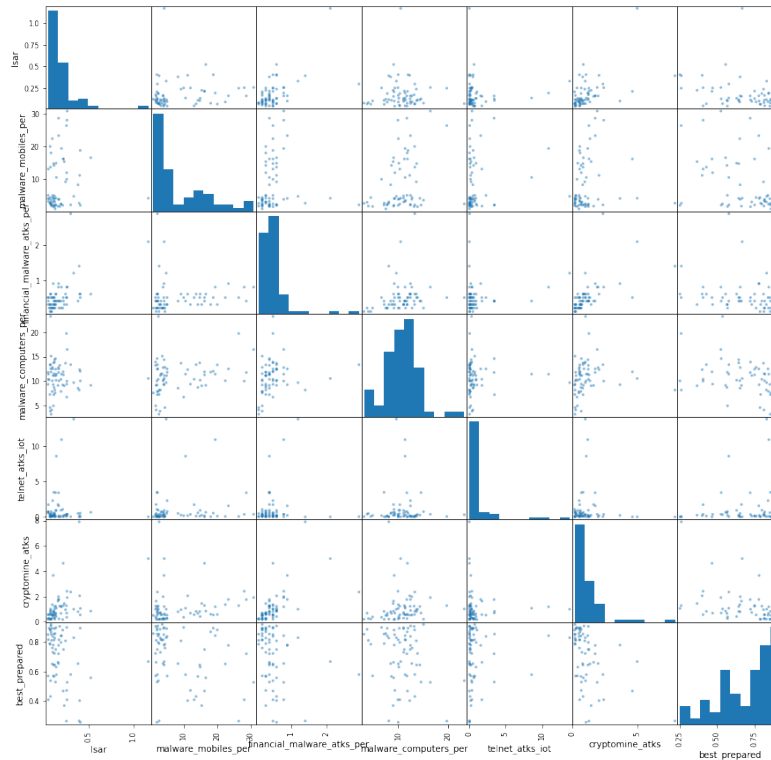

Figure 3.4: Scatter plot matrix - Correlation

(α′) Pearson correlation coefficient   (β′) Spearman correlation coefficient
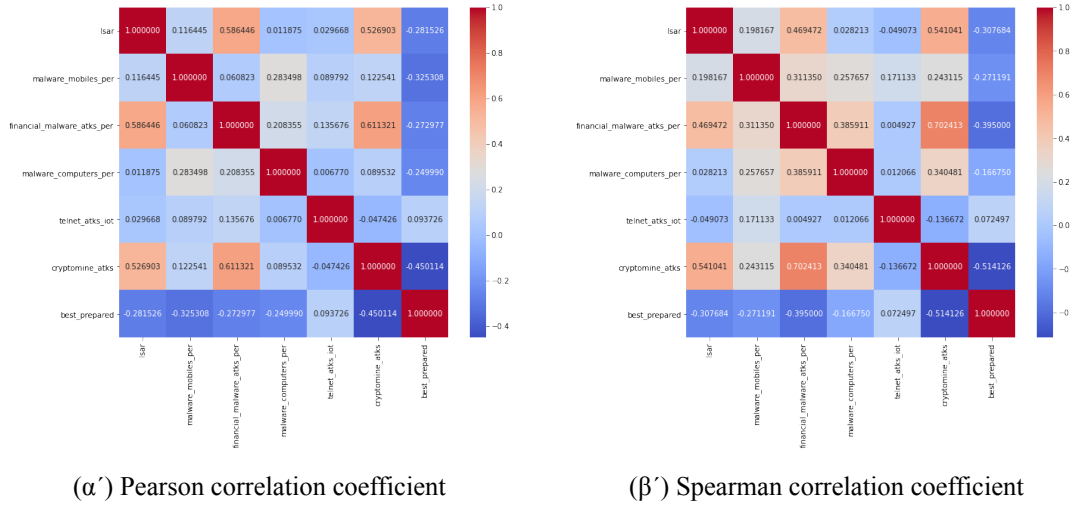
Figure 3.5: Correlation coefficient matrices

The graphs can give us information on the correlation of the LSAR metric with the rest of the data from the comparitech survey, that we need in order to validate our findings:

- LSAR has a moderate uphill relationship with cryptomining attacks (+0.52,+0.54 correlation coefficients). This means that a high LSAR is correlated with a high percentage of crypto-mining attacks. Cryptomining attacks tend to target remotely exploitable devices, even low power IoT devices, in order to amass computing power for mining operations in blockchain cryptocurrencies. Cryptomining is one of the most popular use-cases of Botnets, and this correlation validates our argument.

- LSAR has a moderate uphill relationship with financial malware attacks, malware targeting bank accounts to steal money from victims (+0.58,+0.46 correlation coefficients). While this correlation validates the relationship of high LSAR with high percentage of malware targeting the victim, we miss information to explore whether the use-case is relevant.

- LSAR and the best-prepared metric of the comparitech survey have a moderate downhill relationship (-0..28,-0.30 correlation coefficients), which is expected. This further validates our findings rendering LSAR as a metric to check the security posture of a sum of devices, in this case a country. The coefficients are not very high, which could be explained from the specificity of the usecase of the shodan findings (external attacks) compared to the best-prepared feature which is derived from the Global Cybersecurity Index scores[34]. The GCI score attempts to perform a general security evaluation on a country's cybersecurity including factors such as cyber crime legislation and information extracted from question-naires, hence the index is not fully consistent with our use-case.

- LSAR has a weak uphill relationship with mobiles infected with malware. Additionally, there is a non-significant relationship of LSAR with computer malware which could be explained from the fact that most of it deviates from attacks like phishing, downloaded

malware disguised as a useful program or infected drives. The Shodan findings address the vulnerability to external cyberattacks so a huge proportion of the variability that could be explained is missed, thus the insignificant correlation with mobile and computer malware. Telnet attacks and LSAR also have insignificant correlation which is explained from the fact that they are bruteforcing attacks, not CVE-specific exploits.

Summarizing, we can see that even the omission of a simple activity such as consistent updating of software to secure versions can compromise the security of a device, and collectively widen the attack surface of the device's environment. The LSAR is a metric that can be used to assess the security posture of a large group of internet connected devices, owned and handled by different individuals or organizations by checking the exposure to potential common vulnerabilities (CVEs). Apart from countries, large groups of machines/devices could also be considered to be Wide Area Networks (WAN), geographical regions such as cities, or even large data centers were the VM could take the place of devices, and in those cases LSAR can provide a general view of the awareness of security as well as the density of vulnerable points inside the group.

Chapter 4

Defining an auditing methodology for Internet of Things

As we explained earlier, the Internet of Things is a complex ecosystem that consists of multiple significant components that one should be aware of. And, as far as cyber security is concerned, each of these components presents its own security concerns and widens the attack surface. This chapter presents a coordinated methodology for decomposing complex Internet of Things ecosystems into simple components, and identifying essential security controls and practices that should be applied in these components in order to collectively improve the security of the ecosystem.

4.1 Asset taxonomy on a typical IoT Ecosystem

Various Internet of Things ecosystems already exist in the world, backed by different companies and organizations, using different network protocols, handling data differently in the cloud. However, there are some components that are essentially the same in whichever ecosystem one decides to research, and by identifying these components we should be able to create an image of a typical IoT ecosystem, and generalise its various assets and functionality.

Figure 4.2 presents a typical IoT ecosystem and enumerates its basic assets, providing a useful asset taxonomy. We can identify nine different basic assets, each with a separate role in the ecosystem, associated with different stakeholders and presenting its own security risks. The asset taxonomy consists of the following assets:

1. Users: The entities that use the ecosystem, and benefit from its outcomes. The users can be individuals, organizations, companies or even countries. Apart from entities that use the IoT products for profit, the users can also refer to the people behind the development or marketing of IoT products.

2. Devices: The devices entities refer to the original IoT devices. Smart watches and smart home equipment, sensors, surveillance cameras and generally low-resource internet connected devices all fall under this asset. The devices can have both sensing and actuating capabilities
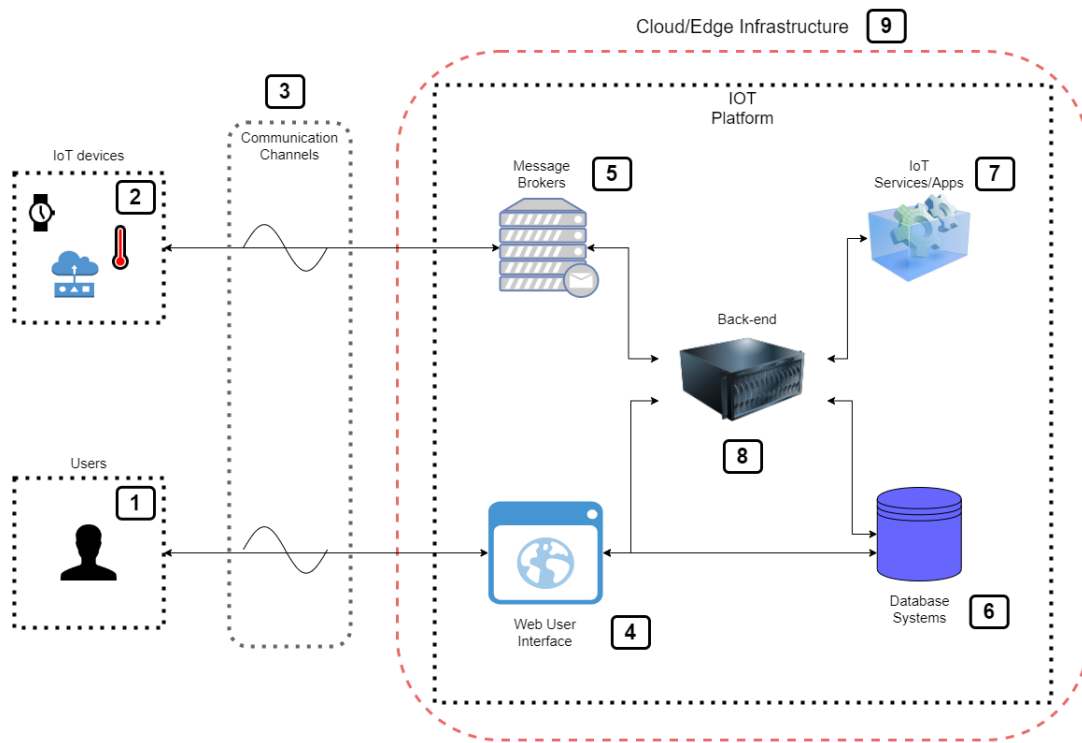
Figure 4.1: Asset Taxonomy of a typical IoT ecosystem

which means that they can be used both as sensing mechanisms, monitoring data from the real world and as actuation mechanisms transferring the results into real world actions.

3. Communication Channels: Communication channels are intangible entities and present the mechanisms that allow the interconnection and communication of users and devices to remote storage and computation spaces called the cloud.

4. Message Brokers: The message brokers are entities that reside in the cloud part of the ecosystem and specifically in platforms and are the main entrypoints for data coming from IoT devices. The message brokers support many application protocols in order to provide interoperability and allow communication with different internet of things devices.

5. Web Applications (UIs): The web applications are the cloud platform's entry points for Users. They are typical web interfaces that allow users to log in with their accounts and perform the various actions that each platforms provides to its users. Some typical actions are adding and removing devices, handling data and applying logic to them or exporting the data from the platform.

6. Database Systems: The database systems are responsible for handling data. They can be relational or non-relational databases, depending on the platform, and they participate into various mechanisms such as authentication and authorization, data management and provision of data to IoT applications.

7. Internet of Things services/applications: This asset refers to applications that perform logical operations on data provided by IoT devices or databases and output results that benefit the Users that use them. Most platforms provide a set of predefined services to their users, and some of them also allow users to create their own applications which are usually executed into sandbox environments such as virtual machines or containers.

8. Back-end Servers: The back end is the application that orchestrates the functionality of IoT platforms. It handles the interconnection and logical operation of the different assets of the IoT platforms, specifically the databases, the message brokers, the web applications and the various back end IoT services.

9. Cloud/Edge Infrastructure: The cloud/edge infrastructure is the invisible layer surrounding the IoT platforms. It refers to the physical servers where the IoT platforms run and the network deployment (routing, DNS etc.) that allows the communication between local users and devices and the remote platform services. It also provides availability to the platforms and bandwidth management.

4.2 Asset security Requirements and Countermeasures

The assets of this taxonomy are present in the majority of Internet of Things ecosystems and we can decompose a specific ecosystem into them regardless of the actual implementation and technical specifications. This divide-and-conquer approach makes the problem of security assessment simpler, as it reduces the surface a researcher has to audit, and makes it more specific and focused. Now, each asset should be assessed and the methodology should inform the researcher of requirements than need to be satisfied, and provide recommendations and practices that will make each asset, and collectively the whole ecosystem, more secure. Of course, the requirements and propositions can not, in any way, cover the whole spectrum of vulnerabilities and threats, but they should provide a secure and trusted baseline that the stakeholders of each asset can build upon. At the end of the chapter, a table will summarize the various security controls discussed for each asset for the readers convenience. Also, since the thesis focuses on Internet of Things platforms, the assets 4 - 8 will be analysed in more depth than the rest of the assets.

There are some security measures and practices that are applicable to almost any asset and for that reason we will discuss them here so that they are not continuously mentioned throughout the next sub chapters.

1. **Keep everything up to date.** Commercial software systems and programs are associated with versions and patches. Often, a specific version of a software is found to be vulnerable and is publicly declared as vulnerable to one or more Common Vulnerabilities (CVEs). Usually a patch or new version is released to fix these security holes and keep the product viable for production and secure. Thus, it is important to update the software versions regularly, because an internet search is enough for a malicious actor to find out whether the version of a system is vulnerable to a known exposure, and more often than not how to exploit it as well.

2. **Backups.** Frequent backups are one of the essential good practices that need to be followed. Source code, Database information, and other data that might be deleted by mistake or malice should be kept in separate storage as backup frequently. This allows for quick recovery of production and even identification of an attack in the event of a security incident by comparing old backups with present status.

3. **Monitoring and Logging.** Keeping information about the change of the state in various assets is important. Logging is the action of keeping records of events happening that could be useful for debugging, or in our case inspecting security incidents. Events that could be logged can belong in a huge variety of information including log ins, log in failures, service creations, errors, database queries, backup timestamps etc.. Monitoring is close to logging, except that the values monitored are usually imaged in graphs, and raise alerts when their behaviour deviates. Monitored information can include machine temperatures, network traffic, network packets, database storage and more. Sometimes Machine Learning techniques are applied to these monitored values to predict and avoid unwanted situations such as Denial of Service attacks.

4.2.1   Users

In the information technology domain, the human factor, and the user asset in our model, is widely assumed to be the most weak and exploitable factor. The occasions in which the human factor becomes the reason for a security breach, mainly fall under three categories, insider threats, careless and unaware personnel or users and lack of business security culture and strategy. The insider threat refers to company workers abusing their internal access deliberately in order to harm the company, most of the times for personal profit. The second category consists of staff that does not have security awareness and employs insecure practices that could compromise a company's security, or an account's and individual's privacy in the case of a regular product user instead of personnel, should they become the target of a malicious individual. The last category includes managerial and technical strategies that a company can apply in order to boost the security posture by minimizing the threat of the human factor compromise.

Undoubtedly, out of the three categories, the insider threat the most difficult to protect against. The malicious actor is always undercover and it is not an issue of knowledge, rather a matter of ethical and psychological factors of the individual. However, there are some strategies that a company could employ in order to minimize the threat of insiders, and basically the aim is to discourage such actors by making it difficult to harm the company severely, even as internal personnel. One measure that companies apply at scale are the so-called NDAs (Non Disclosure Agreements). These are legal contracts, signed by the employees, that forbid them from disclosing information regarding the company to thirds, therefore protecting the intellectual property of the company or organization. Another significant measure, more on the technical side, is strict control over the company's assets and the trust that employees have over accessing them. This can be achieved by monitoring the access on these assets, logging and producing analytics. The monitoring of access

and data processing activities could discourage an actor from misusing this trust for malicious purposes. One more measure in the same fashion are company-wide policies for least priviledge and segregation of duties, so that access is given to only the assets that are absolutely needed for the completion of each employee's function.In this regard, AI technology can also contribute through User and Entity Behavioral Analytics (UEBA), that monitor user behaviour to identify anomalies and potentially prevent malicious actions. Tools like IBM QRadar UBA are able to monitor human factor behaviour, assign roles and identify role behaviour deviations to alert on occasions like tool misconfigurations, sharing of credentials, or admins changing user attributes [53].

Security awareness training is applied from companies worldwide in order to cultivate the personnel's security culture, awareness of good practices and sense of responsibility. Occasional briefing on security and simulation of attacks such as phishing from the security teams of companies to the rest of the teams, can help create good practice habits on the employees that can collectively improve the security posture of the company. Good practices could include strong passwords, scepticism against the trust of emails, locked screens when away from a computer, avoidance of using third party devices such as USBs and personal laptops at work, destruction of documents that are no longer useful and many more.The power of machine learning can also be leveraged here, as algorithms such as kNN, SVM, Random Forest, Neural Networks as well as unsupervised and similarity learning techniques perform well into detecting social engineering attacks such as phishing [4][58] and malicious URL links[69][66].

Lastly, there are management strategies that are not purely technical and could improve the security of the company significantly. Responsible vulnerability disclosure programs, are methods in which external researchers or regular users that manage to find a bug in a product can disclose it to the engineering teams of the product so that it is quickly patched before malicious activities exploit this vulnerability/bug. Usually, the individual is rewarded by the company financially, or with exposure, depending on the severity of the found bug. This gives an incentive for researchers to disclose the bugs responsibly and not personally profit from them with malicious activities. Bug bounty programs are the next step in this mindset where scopes and potential rewards are defined by the company for security researchers. Another significant measure is periodic company-wide risk and threat assessment, by either the company's internal security team or employment of external red teams and penetration testers. These teams perform simulations of real attack scenarios in order assess the effectiveness of applied security controls and countermeasures. Finally, security incident scenario strategies should be in place in order to define the actions that will take place in the worst case of a security breach so that the company can identify a potential security hole, patch it and recover from the breach as soon as possible.

4.2.2 Devices

The Devices asset consists of every Internet of Things device that can be used inside the IoT ecosystem. From a security perspective, the asset is concerned with the aspects of hardware, software architecture and physical security of the device, since the network security aspect is mostly discussed in the Communication Channels chapter.

Table 4.1: Users Asset Security Measure Checklist

Users asset Security Measure Checklist	
Insider threat	Non Disclosure Agreement Signing
	Strict Asset Access Control/Monitoring (UEBA,logging)
	Least Priviledge policies
	Segregation of duties
Lack of security awareness	Security Awareness Training, Briefing
	Social Engineering Attack Simulations
	AI On Social Engineering (Phishing, Malicious URL)
Management Strategy	Vulnerability Disclosure Programs
	Company-Wide Risk Asessments
	Security Incident Scenario Strategies In Place

Physical security is defined by the controls that exist in place to protect against malicious activities from actors with physical access to the device. There are a lot of techniques that users and device vendors can apply in order to improve the physical security of their product. As far as users are concerned, it is important that the device is not accessible, at least easily, by thirds. In the case that this is not avoidable (e.g. in the case of street cameras, or IoT devices in crops) the devices should not have physical ports such as USB ports exposed. AI biometric access control to the IoT devices is encouraged when combined with rule-based access such as passwords, as AI substantially improves the accuracy of fingerprint, facial or iris scans [77].

In the case of vendors, the physical security aspect is a more technical issue, and much more difficult to apply. Starting off, there must be tampering prevention mechanisms [21] in place that should make it difficult for someone to tamper physically with the device. A common technique is security bits hidden under rubber feet or labels with complex screw heads that need specific screwdrivers to open. Also, strong adhesives and in some cases ultrasonic welding can make opening the shell of the device difficult to do without damaging the device. Integrated circuits and boards can be encapsulated or coated with specific materials such as epoxy or silicone, that secure the devices from physical factors such as heat or dust, but can also protect from malicious acts such as cloning or reverse engineering. Security fuses are also widely used, and they are mechanisms of access control to the on-chip memory. These mechanisms are usually built in a way that they destroy stored data in the case that someone attempts to erase or reprogram them (usually using UV lights). Scrambling of memory, data buses and even topology layout is a method that is commonly used in order to prevent reverse engineering of the device and forensics. Memory scrambling [48], for example, turns data into pseudo random patterns that make it difficult for someone to reverse engineer and read them.

In many cases, tampering detectors are also installed into the device. Detectors include temperature sensors, voltage and radiation sensors. Therefore, physical attacks like cold boot[32], X-rays (used for reverse engineering-looking under coatings) and fault injection attacks that use sharp electrical signals to cause circuit errors can be detected and be handled accordingly. Side

channel attacks [6] are also a major threat for embedded devices. Passive Side channel attacks resort to analysing times, power consumption and temperature during cryptographic operations in order to identify properties, algorithms used or even keys. Countermeasures insert randomness in order to render the analysis useless, by time skewing, random heating, cache flushing, disabling or bypassing and many more methods.

Internet of Things devices can also be severely susceptible to Denial of Service attacks. Vampire attacks[72] are attacks that attempt to drain the battery of ad-hoc wireless devices in order to lead to a DoS situation where the nodes shut down and do not communicate with the rest of the deployment. This can be achieved through various methods and two of them are the adversary introducing routing with loops (Carousel attack) or introducing very long routes (Stretch attack) in the node. In the first case, the node is forced to sleep deprivation as it has to process incoming packets each time, and in the second case multiple nodes are affected if the route is constructed to contain them. Mitigation controls include the ability to reroute at each node if a shorter route is known or introducing a no-backtracking metric that ensures the gradual progress of the packet and avoids loops. Denial of Service can also be achieved through Jamming attacks, but they can easily be avoided through frequency hopping, the existence of directional antennas, or by spectrum spreading [18].

Trusted computing [71] is another important aspect of the embedded IoT devices' security posture. Trusted Execution Environments (TEEs) are processing units that ensure the protection of code and data inside them. Usually this is achieved with dedicated coprocessors with secure mode where security tasks are being offloaded from the main processor, and secure memory (dedicated on-chip RAM). Also, since outside the TEE the data are not secure there should be integrity checks in order to detect modifications while outside the TEE. Secure booting is a significant feature of a TEE, as it verifies an image before it is executed, and in order to be successful secure storage of signatures and secure code for verification must be ensured. Therefore, the keys and signatures are written into protected read-only memory called hardware root of trust, that usually is on-SoC (System on chip) OTP (one-time-programmable) hardware that acts as anchor for the chain of trust.

Firmware updates is another issue that should be addressed. It is suggested that firmware updates should be encrypted and authenticated as well as be installed over the air (OTA) via secure protocol channels.

Finally, application whitelisting is a popular method for avoiding malware installed inside the device. Gopal et al. [28] use a store of binary checksums collected at a clean device state to block untrusted software execution and prevent its spreading, specifically of the Mirai malware in that work. Malware detection in IoT devices can also be performed by static analysis of high level features using multiple classifiers like RIPPER, SVM, neural networks and more [50].

4.2.3 Communication Channels

Inside an Internet of Things ecosystem devices communicate with the cloud and the same is true about the individual users. The data exchanged within the communication channels can be sensitive and private, thus eavesdropping and tampering must be avoided. Cryptography is the method that

Table 4.2: Devices Asset Security Measure Checklist

Devices asset Security Measure Checklist	
Physical Security	Inaccessibility To Devices, No Exposed Ports
	Biometrics Access Control
	Board Encapsulation/Coating
	Secure Casing (Hidden/Complex Security Bits, Strong adhesives or ultrasonic welding on assembly)
	Layout Scrambling
Hardware Security	Security Fuse Usage
	Tampering Detectors, Randomness Against Side Channel Attacks
	Jamming and DoS Avoidance through Secure Routing, Frequency Hopping, Spectrum Spreading
	TEE - Trusted Execution Environment
	Secure Booting, HRoT (Hardware Root of Trust)
Software Security	Application Whitelisting
	Static Analysis Malware Detection (AI Classifiers)
	OTA Authenticated/Encrypted Firmware Updates

is widely used in order to avoid Man In The Middle attacks (MITM), where a malicious actor uses network packet sniffers to inspect the data exchanged through the network between a victim node and its communication target, in order to read sensitive data.

Transport Layer Security (TLS) is the standardized and globally accepted solution for secure encrypted communication, descendant of the now deprecated Secure Sockets Layer (SSL). Specifically, TLSv1.2 and TLSv1.3 are the standardized (defined in RFC5246 [59] and RFC8446 [60] respectively) non-deprecated protocol versions used at the moment. The protocol consists of the TLS Handshake Protocol which initiates the communication and the TLS Record Protocol. During the TLS Handshake, the client initiates the communication and communicates the symmetric keys and algorithms that will be used throughout the communication with the server. This handshake part is encrypted asymmetrically, using public and private keys before communicating sensitive information such as the symmetric key. After the symmetric cryptography parameters have been deciced, the TLS Record Protocol takes over, and the secure client-server communication is now achieved over symmetric cryptographic algorithms. Using TLS provides the client and the server with privacy (confidentiality), authentication and integrity. TLSv1.3 provides faster and more secure communication that 1.2, with more features such as Forward Secrecy, which ensures that even if the symmetric key is found by a malicious actor, no past messages can be decrypted.

The use of cryptography, however, presents the engineers with a significant tradeoff in the case of Internet of Things devices. Overheads in time and processing power happen during the calculations for encrypting, decrypting and key generating and exchanging. Since IoT devices tend to use fewer resources than the rest of Internet connected devices such as personal computers, a lot of research has been conducted towards the finding of a TLS cipher suite that minimises that

overhead and makes the encryption of IoT communication feasible and fast. A TLS cipher suite is comprised of an asymmetric algorithm used for key exchange, a symmetric bulk encryption algorithm, a hashing algorithm and a MAC algorithm. Algorithms of the Authenticated Encryption with Associated Data (AEAD) category combine the Bulk encryption with the MAC part in order to provide a more robust and fast security solution, and are proposed heavily by the 1.3 version of TLS.

Starting off with assymetric cryptographic algorithms, the main options have been the Diffie-Hellman (DH) and RSA, with DH being used more and more nowadays as its ephemeral version, the Ephemeral Diffie-Hellman (DHE) which provides Forward Secrecy. Elliptic curve cryptography (ECC) is also preferred as it provides analogous security with smaller key sizes [27] (meaning less resource usage) to non-ECC public key cryptography, and an indicative analogy is that an ECC system with a 160-bit key is equivalent to a 1024-bit RSA key system. Therefore, ECDHE and ECDSA should be the proposed key exchange and digital signature algorithms respectively. ECC security and usefulness stems from pure algebra and elliptic curves such as:

$$y^2 = x^3 - 4x + 5$$

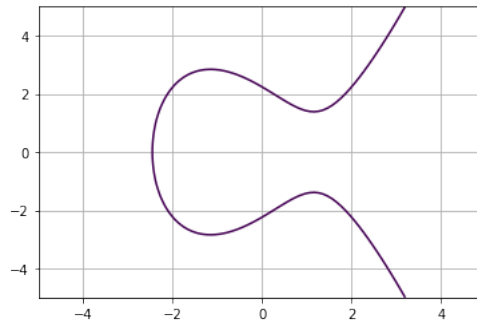Plotting this curve gives us the following plot:

Figure 4.2: Plot of the $y^2 = x^3 - 4x + 5$ ellliptic curve

The reason that such curves are used for cryptographic reasons is the property, that if a line connects two points of the curve, the line will pass through a third point. If we suggest that two of these three points are the same point P, infinite lines can pass through, but the tangent line is chosen and the third point is where the tangent line intercepts the curve. We are using a second property of the elliptic curves which is the symmetry about the x-axis, in order to reflect that third point and get its symmetrical across the x-axis. This is the result of the addition of the two first points which happen to be both P, so the third point is 2P. Finding the line that passes through P and 2P will give us another point, which if we reflect gives us 3P and so on. Using this approach for N repetitions gives us a point Q = N*P which is used as a public key, and N is a 256-bit integer which is used as a private key. The base point P and some other constraints are dependent on the ECC algorithm used. Even if an adversary knows the curve used and the starting point additionally to the public key, calculating the private key would take continuous additions which would be any number between 0 and 2^{256}, making it not practical in any way to bruteforce.

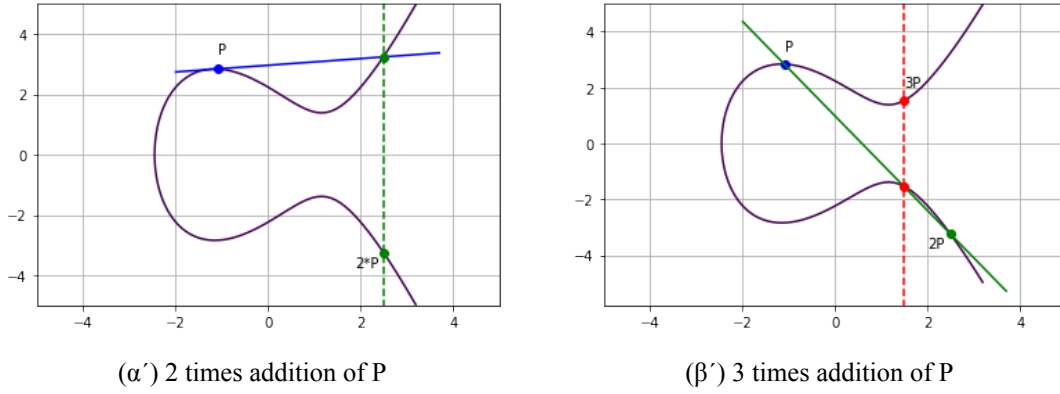(α´) 2 times addition of P   (β´) 3 times addition of P

Figure 4.3: Point addition operations

Regarding block ciphers, algorithms of the AES (Advanced Encryption Standard) family are proposed and specifically AEAD algorithms are beneficial to use as they are highly secure because they encrypt and authenticate in one internal pass and difficult to implement wrongly, as opposed to combining two algorithms (e.g. AES-CBC and HMAC). The two options that are generally preferred are firstly AES-GCM and secondly a stream-based cipher combination of ChaCha20 and Poly1305-AES. AES-CBC used to be a widely used cipher but its lack of speed (cannot be written in parallel) and a vulnerability to padding oracle attacks [78] makes it a less favorable option. 128-bit keys are generally used in Internet of Things for their better performance comparing to 256-bit keys. Arunkumar et al. [10] propose ChaCha20-Poly1305 as the favorable option for smart devices in TLSv1.3, but in TLSv1.2 AES-GCM is proposed, especially with the performance spike in devices with specific instructions for hardware acceleration in AES rounds.

As far as hashing algorithms are concerned, the performance evaluation generally seems to have minimal significance compared to the latencies and energy consumptions of asymmetric algorithms for example. Nevertheless, Pereira et al. [54] on their performance evaluation of hashing algorithms in IoT platforms and embedded devices, find Blake2 [12] to be more lightweight, energy efficient and fast. Other lightweight hashing families of algorithms are Photon [31] and Quark [11].

Lastly, TLS provides the capability for two-way authentication. Servers carry certificates, most of the time signed by a trusted third party organization called certificate authority (CA), in order to be identifiable and trusted by the clients. The reverse can also happen, where a server can be configured to accept only clients that are authenticated with their own certificates, signed also by a trusted CA. This can be sometimes useful in the ecosystem of Internet of Things in order to authenticate devices that send data to the IoT platform, and the standard format being used is X.509. When TLS client certificates are not preferred and an Edge deployment is used instead of a cloud one, the devices can be authenticated through the use of AI algorithms for proximity-based or fingerprint-based authentication, where IGMM, Q-learning and neural networks are found to produce highly accurate results [74, 33].

So far, we have assumed that the devices have the capabilities of establishing a TLS connection with a remote server. In some low-resourced devices though this is not the case, and the minimum threshold for TLS-based solutions is 10KBs of RAM and 100KBs of ROM. In these situations a

middleware is needed to provide the TLS-based communication for the constrained IoT devices [38].

Table 4.3: Communication Channels Asset Security Measure Checklist

Communication Channels asset Security Measure Checklist	
TLS Cryptography Usecase	Independent TCP over TLS, or Offloading to TLS gateway
TLS Characteristics	TLSv1.3 for Forward Secrecy, Efficient Cryptography, Else v.1.2
	Assymetric: Elliptic-Curve Diffie Hellman Ephemeral (ECDHE)
	Symmetric: AES-GCM, ChaCha20-Poly1305
	Hashing: Blake2, Photon, Quark
Authentication	2-Way TLS Authentication with X.509 Certificates
	AI-based Authentication (Proximity/Fingerprint Based)

4.2.4 Message Brokers

The message brokers are the entry points of IoT device data to the IoT platform, and they usually work with multiple application layer protocols such as HTTP (REST), MQTT and CoAP. TLS and X.509 certificates are the way to secure communication between devices and message brokers, as discussed in the previous chapter, and that is the way to avoid Man In The middle attacks. If mutual authentication is configured, this is the asset where devices are authenticated and it is decided whether they can send information to the platform. If client authentication with TLS certificates is not used, authentication with passwords or tokens can also be implemented in this asset, where the broker might consult the backend for authentication purposes.

Another security measure that can be typically implemented in the asset of message brokers is authorization and access control. Let's take for example the Message Queuing Telemetry Transport (MQTT) protocol. MQTT works in a publish-subscribe fashion where topics are the main entity, and data sent (published) to a certain topic are received by clients subscribed to that topic. By defining policies, we can authorize certain IoT devices to publish in a certain topic, so that their data are used by the intended subscribers, and vice versa so that subscribers ensure that the data they use deviate from specific trusted publishers. Each Message Queue/Broker server usually provides a certain way of defining access control and authorization policies, but the two most common approaches are:

- Access Control Lists (ACL): In this case, every entity sending or receiving information has an ACL entry associated with them, where permissions to services are defined, as well as actions that can be performed within these services.

- Role-Based Access Control (RBAC): Role-based control works with roles. Every entity is associated with one or more roles, and each role has defined permissions and allowed actions. Genetic Algorithms can be used for role-mining in order to automatically create roles and define RBAC policies [20].

Lee et al. [40] also present some other authorization trends such as UCON (Usage control)

which is used for continuously mutating authorization factors such as pay-per-view or metered payment situations and CapBAC which uses tokens to associate users with specific capabilities.

Table 4.4: Message Brokers Asset Security Measure Checklist

Message Brokers asset Security Measure Checklist	
Authentication	Client TLS Certificates
	Authentication Tokens
Authorization	Access Control Lists (ACL)
	Role-Based Access (RBAC)
	Usage Control (UCON)
	Capability Based Access Control (CapBAC)

4.2.5   Web Application Interfaces

Web applications usually are the asset that provides the biggest attack surface to potential malicious actors. This is because they provide a wide range of functionalities triggered by user actions, and they are fully visible to the outer world. There are numerous ways in which a hacker can approach harming a web application, and there are also various well-functioning tools that help them to do so.

The first obvious step, as discussed previously, is encrypting the communication of users and frontend. Users perform actions with sensitive data including logging in using their credentials, thus the web application should not be vulnerable to eavesdropping. This is achieved through TLS certificates (of version 1.2 or 1.3) and enforcement of the HTTPS protocol to protect the communication. The certificate's validity should also be ensured, and it should be renewed before its expiration dates, when someone else could buy the domain name and perform a domain takeover. In order to build a general security mindset about developing web applications, a developer should be aware of at least the most popular web application risks, and know how to protect systems against them. OWASP Top 10 is a widely known report listing the top vulnerabilities in web applications in order to provide guidance for developers, and it is important to inspect these risks, most of whom have been persistent threats for a lot of years, and recommend ways of tackling them and making the web application more secure.

Injection attacks can take many forms and they essentially refer to commands being passed to an interpreter or another program, where part of the commands is derived from user input. SQL, NoSQL, LDAP and OS injections all belong to this category. In order to create an image for it, let's assume a simple login form containing a username and a password field, with these data being stored in a MySQL database. Assuming that the input value for the username was used as input into a back end query to the database, for example:

```
1     "SELECT * FROM user_table WHERE username = " + username + ";"
```

a malicious user could enter the following input as username:

```
1     user; DROP TABLE user_table
```

Using the input username in the query would make MySQL perform a select operation and then drop the whole users table. The same problem can appear in back end functions that perform local operating system commands (in a local shell). User input validation (making sure it conforms to the available options and is consistent with the expectations) and sanitation (modifying the input in order to be valid) is needed to constrain the choices the user has in the data he enters. White-listing of characters/length/format is needed to be enforced in the input. For example, a username should not contain symbols such as ';' or spaces. Another measure, specific in the case of SQL injections, is the use of parameters instead of pasting the input inside the query. Parameters are handled internally as literal values instead of executable code. Also, stored procedures are preferred to dynamic SQL whenever there is the choice between the two, so that business logic is not mixed with the database logic. Lastly, the database user used for the queries should be created with a least privilege logic, to avoid causing damage to other assets in the case of a successful SQL injection. A popular tool to check for SQL injection in a particular form is SQLMap.

Broken authentication is another common issue that deserves a more focused approach. Starting from login and register procedures, a strong password policy should be enforced by the software, and error messages on login failure should not provide information to adversaries that would allow user enumeration. Brute-forcing username-password combinations in these forms should also be taken into consideration. Some measures to avoid brute-force automation include temporary account lockouts after consecutive login failures (usually 5), captchas (simple puzzles for humans that cannot be automated from a machine) or 2-factor authentication methods. Also, robust password change or reset procedures should be in place, so that a malicious actor cannot change a victim's password and take over. Another major factor in authentication is robust session and authentication token management. Tokens and session ids are 'secret' strings with random characters that identify a user and a session and enable authenticated access to resources and functionalities. Such strings should be created with true random number generators, not pseudo random ones, as with pseudo random generators one could be able to predict future values, and therefore access information using another user's authentication token. They should also have adequate length, an expiration date, and should be invalidated after the end of a session so that they are not reusable by someone else. Browser caching of these fields should also be avoided using Cache-Control headers in the HTTP replies and associated fields such as no-cache, no-store, max-age or private.

In the same fashion, access control should be implemented correctly and carefully so that users only have access to authorized content. The authorization is mostly implemented with middleware software between function calls that acknowledges whether the user is authorised to access the functionality after the middleware. This decision is based on whitelisting of users or user roles for specific content.

Cross-site Scripting (XSS) is one of the most common exploitations happening in web applications, and it occurs when malicious actors enter scripts in user input which are not correctly validated and then take part into the generation of an output. When another user renders the page containing that output, the user's browser will execute these scripts and cause malicious behaviour. Two kinds of XSS exist depending on the persistence of the data where the malicious script is inserted, reflected XSS for non persistent data like links and stored XSS (or persistent) for data

that are stored in databases or source code. The identification of such vulnerabilities is based into identifying these entrypoints of user input and applying validation and escaping when this input is going to be used into HTML, CSS, Javascript and generally any interpretable content. URL queries can also be used as an entry point to enter malicious scripts so they should be escaped as well. Content-type headers should also be setup appropriately for the expected input type so that the browser does not misunderstand the content and execute malicious input. For example a JSON input should have set application/json as the Content-type Header and not text/html. Content Security Policy (CSP) is also a good but relatively complex measure to avoid XSS vulerabilities, which is a special Header instructing the browser to execute or render resources from whitelisted sources. Lastly, It is suggested to use the HttpOnly flag on cookies, which instructs the browser to not let client-side scripts read the value of cookies.

Moving on to XXEs, eXternal XML Entities is a vulnerability found mainly to old or misconfigured XML processors, where they evaluate XML entities from external sources which can be used maliciously to perfrom internal actions such as file reading, internal port scanning or remote code execution.

Errors are also a major factor that should be handled appropriately by developers. A malicious actor trying to break the application should not be able to see error messages that could potentially reveal information about the implementation of the app. Thus, it is important that debugging and error information are not logged in production web applications, and generic error messages like "404 Not Found" are shown to the users in the case of a failed state.

Common updates and inspection of the potential existence of known vulnerabilities in these components are a way to evaluate the secure state of the product. Frequent scanning for vulnerabilities is also a good measure. Web applications are usually massive, making it hard to track security issues in its entirety. Thankfully security tools for web application security testing are in abundance, a lot of them being Open Source and having the support of the developer and research community. Arachni, Iron Wasp, OWASP ZAP, W3af, Wapiti and Wfuzz are well known vulnerability scanners and fuzzers that can be used to assess the security of a web application, testing for a huge variety of potential vulnerabilities including XSSes, XXEs, Privilege Escalations, SQL injections, Information disclosure, error disclosure and several more insecure factors. Finally, web application firewalls can help mitigate lots of attacks through a mixture of the traditional signature-based approach and supervised or unsupervised Machine Learning techniques to handle unknown injection attacks [29, 43].

4.2.6   Database Systems

Databases are the asset that holds the majority of the information of the IoT platform as well as the functionality to access them. The information stored inside should have their confidentiality and integrity protected. Starting with the SQL injection vulnerability that was mentioned in the web application asset chapter, stored procedures was proposed as a way to limit outer effect to internal queries to the database. Access control is another way to limit the capabilities of queries that could potentially be malicious. The user that makes the queries should not be a root user, but should only

Table 4.5: Web Application Asset Security Measure Checklist

Web Application asset Security Measure Checklist	
Injection Attacks (SQLi, OSi, XSS etc,)	Input Validation, Sanitization, Character Escaping
	Trusted third-party Libraries or frameworks
	Adequate Content-Type Headers
Authentication	Strong Password Policies, Robust password change procedures
	Bruteforcing protection
	Truly random auth tokens, adequately lengthy with expiration dates
	No caching of sensitive information e.g. tokens
Access Control	Authorized access to content
Information-Disclosure	Non-Information Exposing Error Messages
	Non-Exposure of Sensitive assets
Vulnerability Assessment	Web-Application Vulnerability testing tools
Intrusion Detection/Prevention	Hybrid Web Application Firewalls (WAFs)

have authorization for specific information and procedures. Furthermore, the databases should not be directly accessible from the Internet. In that case, remote malicious actors would be able to gain information such as the database's version or port, as well as send payloads to test penetrating them.

To avoid losing data from accidental or malicious behaviour, often backups of the database should be taken, and be stored in separate storage space. In addition, data, and essentially sensitive data should be protected even in the incident of an information leakage. For that reason, credential information such as passwords should be hashed, and the authentication should be performed by comparing the hash of the password given by the login form with the password hash located in the database, so that even if the case that this hash is leaked, the malicious actor cannot discover the original password and access the victim's account without first having to bruteforce for the original password. Additionally, the whole database could be encrypted though that does come with a trade-off in the latency (and potential insecurity) that the middleware application that encrypts and decrypts the data introduces, especially when the data are not encrypted uniformly with the same key, but with a different key for each row for example.

Cryptographic key management is also an issue that should be tackled in the database asset level. Private, Symmetric and Hash keys that are used to encrypt, decrypt or digitally sign data need to be kept on a secure storage where they are accessed only by authenticated users, mostly developers. A leak of these keys could be catastrophic as the adversary would potentially be able to read sensitive information in plain text. First and foremost, these keys should not be kept in the database with the data they protect, and if possible not even in the same server. In the case they belong in the same server, they should be given appropriate read-write-execute permissions. A solution heavily proposed, although expensive, are Hardware Security Modules (HSMs) which

are hardware solutions for keeping keys and performing cryptographic functionalities for the server and sending back to it encrypted or decrypted data.

Table 4.6: Database Systems Asset Security Measure Checklist

Databases asset Security Measure Checklist	
SQL Injection Protection	Stored Procedures
	Parameterized Queries
	Non-Root Database User for Queries
Database System Protection	Non-Exposure to the Internet
Data Protection	Hashed Sensitive Data Fields
	Encrypted Database (Optional - Tradeoff)
	Regular Backups
Key Management	No Key Storage where Data are stored
	Access Control where Keys are stored
	Hardware Security Modules (Optional - Tradeoff)

4.2.7   Internet of Things Services

Every Internet of Things platforms provides applications and services that process the data incoming from the IoT devices and forward the results to users, IoT devices or other applications in the case of a pipeline. A range of preset applications is usually provided by the platform to the users, but most of the commercial platforms also provide a capability for users to create their own applications, deploy them in the cloud platform, use them or share them with the platform's community. As with any user input and especially executable content in this case, this poses a security risk for the platform and should be carefully handled and implemented as a service.

Whenever the execution of a process needs to be controlled, there is a need for isolated environments, and the solution is usually through virtualized environments. These types of environments are capable of running untrusted programs or opening untrusted files that could potentially be malicious inside a controlled environment without directly affecting the server in which they reside. The main forms of virtualization are virtual machines and containers, and their differences lie in the implementation level.

Virtual Machines implement full virtualization to create different machine instances using a special handler software called Hypervisor. The hypervisor stands above the OS privilege level and has the ability to provide safe virtual hardware resources for the different operating systems that it controls, isolating them from the main host. Virtual Machines are great for creating isolated production level servers on the same host but they can be costly in storage and time for temporary spawning of untrusted user-created programs.

Containers stem from Operating System level virtualization where multiple isolated user-spaces are allowed to work by the OS. Containers produce significantly less overhead than virtual machines
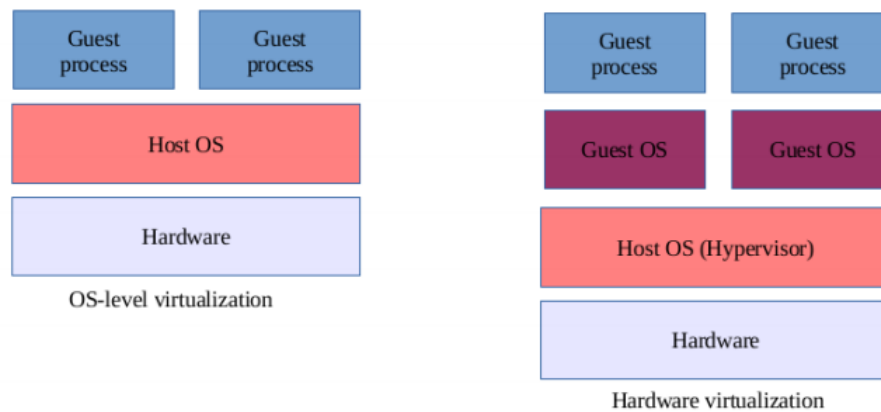
Figure 4.4: Difference between Containers and Virtual Machines, picture from 'Understanding and Hardening Linux Containers' [30]

in time and storage, they are fast to deploy and kill, and they provide a customizable isolated environment and that is the main reason why they are popular for application deployment, with the main container platforms being Docker, LinuxContainer (LXC).Containers usually make use of Linux kernel's features to become more secure. Namespaces allow the container to have a limited view on the resources providing isolation in different aspects such as Filesystem (MNT Namespace), networking (NET Namespace) and processes (PID Namespace). Furthermore, the cgroups utility is used to hide system limits from containers in order to provide limited CPU, memory and other resources to the containers. Seccomp (Secure computing) is another feature of the Linux kernel being used in order to limit the system calls that the container can perform, and the user can define seccomp profile configurations in order to restrict these capabilities of the container.

The most popular of these environments are developed with security in mind in order to protect the host system, but the code that runs inside the containers can attempt a plethora of malicious activities like internal network scanning or privilege escalation inside the container. That is why it is important to go a couple steps further in order to secure the untrusted code execution. The first and always relevant countermeasure is keeping the host and the containers up-to-date with relevant security features and options enabled. Also, since containers have direct access to the kernel compared to VMs that have a virtual kernel between, Upadhya et al. [70] propose to run containers inside a VM in order to add the virtual kernel layer of security in the case of a container escape. Other measures applied to make this process more secure are running the programs created by the user as non-root and with least privileges, and in secure minimal container images containing just the binaries that the program needs in order to function. Common unix tools like shells, package managers, compilers or network tools like curl, netcat and nmap should not be included inside the container that will run the untrusted code. Also, restricted versions of programming languages are usually employed in these situations in order to avoid language-specific capabilities such as execution of shell commands, and on compilation of the untrusted code relevant security

features should be enabled. The network is also a major security issue. If the program does not use internet resources, or programs cannot use the internet as a design choice of the platform, then the networking capabilities can easily be removed when initiating the container. Otherwise, some measures should be taken so that the container cannot communicate with other containers in the network unless explicitly stated. Lastly, in some cases the spawned services might attempt to starve the host of resources by either looping or starving for prolonged timeframes. Thus there should be a time and resource (CPU, Memory, Storage) quota on the spawned containers in order to avoid this kind of Denial Of Service situation.

Table 4.7: IoT Cloud Services Asset Security Measure Checklist

IoT Cloud Services asset Security Measure Checklist	
Container Configuration	Containers spawned in VM for Virtual Kernel
	Seccomp profiling if applicable
	Strict Resource Quotas
	Only essential internal networking (No networking if not needed)
Untrusted Code Execution	Non-Root, Least Priviledge Container User
	Minimal Container Images
	Installed only needed Libraries/Binaries/Tools
	Restricted Programming Language Support

4.2.8   Backend Servers

Backend is the asset where the functionality of the different parts of the IoT platform is orchestrated. The backend is the asset that receives the data from the brokers and stores them into databases, provides and receives data to the IoT services and communicates with the Frontend Web Applications to serve the users' needs. The backend might also provide functionality directly to the Internet, mostly through Application Programming Interfaces (APIs). Web application and Database Security have already been discussed in their respective sub chapters, so here the focus is to the APIs used either by other assets or external users.

First, the public APIs that are exposed to the Internet should be protected with encryption (HTTPS if we are talking for the popular REST API paradigm) in order to avoid eavesdropping. Also, authentication should be enforced in order to use the API. For the authentication the most common choice is API keys or tokens that are created by the user inside the web application. Sometimes the user-password pair of the webapp are sent inside the request but this should be avoided as a leak of them would compromise the whole account and not only the API functionality, while in the case of a leak of the API key, the victim could just invalidate the old key and create a new one. Authorization should also be kept in mind in this level. As with functionalities that users use from a Web frontend application, the APIs must also ensure that the user only accesses and uses the content he is authorized for.

The rate of the requests is also a factor that needs to be accounted for in order to avoid DOS

situations and make the API scalable. Rate limiting can be implemented in many ways, with the most popular being putting the request in message queues and process each one in a specific rate, or throttling of the user's connection (bandwidth limiting or drop) upon detection of surpassing the request rate.

Input parameter validation should be performed in the API requests as with any entry point, using validation rules to enforce that the input is consistent with the API expectations. In the event of an invalid request, the request should be denied and a reply explaining the invalidity should be returned, without involving technical characteristics of the validation mechanism. The validation could be implemented as a middleware receiving the requests at an API gateway which could be used for other reasons as well, such as monitoring API traffic and applying machine learning and AI to find deviations from normal behaviour and flag possible attack attempts.

Finally, the backend code is typically the platform's most lengthy and important part of software. Consequently, the quality of the code and the use of secure practices should be enforced. Static Analysis tools are essentially tools that parse code and search for insecure functions or practices in order to find vulnerabilities and security holes in the software. It is proposed that such tools are run, often in conjuction to each other, in order to ensure code quality, and even better they should be added into the SDLC as an extra step in testing before the deployment of a new version.

Table 4.8: Backend Asset Security Measure Checklist

Backend Server asset Security Measure Checklist	
API security	Encryption on API requests (HTTPS)
	Authentication with API tokens
	Authorization
	Rate Limiting in API requests
	API request Validation and Sanitization
	Intrusion Detection on API requests (Heuristic and AI based)
	Regular Static Analysis of Code

4.2.9   Cloud/Edge Infrastructure

The last asset that is worthy of concern security-wise is the cloud/edge Infrastructure. The Internet of Things platforms are deployed to cloud infrastructure in order to become available globally to IoT devices and users and enjoy the extra layer of security surrounding them. The stake holders for this asset are the cloud (or edge) space providers (CSPs) which are typically different than the platform creators. This asset is mostly concerned with measures that raise the IoT platform's safety, availability and resiliency against cyber attacks.

Starting from physical security, the cloud/edge infrastructure is expected to have strict access control with multi factor authentication to the machines and other assets, and a great resiliency to physical disasters like fire, lightning or blackouts. Camera surveillance is also a significant factor to physical security in order to protect the assets and control the access to them. Device and network

monitoring is a must for cloud providers. Everything, from machine temperature, CPU, storage and memory usage to network traffic should be monitored and alerts should be implemented in the case that the values reported exceed expected behaviour. In case of a problem in the provider's side, the problem should be resolved as quickly as possible since availability should be kept to a high standard. However, if the problem is from the client's side, the support should communicate and collaborate with the platform's support to resolve the issue. Strict control should also happen in the application level. Since the deployments of cloud platforms as well as services like e-mails and DNS tend to be controlled from management applications used by the cloud provider stuff, the access control to the IoT platform's source code and settings is only as secure as the cloud provider's access control. Additionally, the CSP should be integral and confidential with the customer's data, credentials and communication.

On the technical side, in the case of production level platforms, deployment to dedicated machines should be chosen above virtual machines. That measure is significant in order to avoid co-existence with other VMs in the same machine, that could potentially attempt to harm the platform's VM or extract information through side-channel attacks. In the case of VMs on the same host, the CSP should avoid interconnection between the VMs and provide isolation of the processes, as well as well-defined resource limits so that one VM cannot starve some of the host's other VMs.

Cyberthreat detection is also a responsibility of the CSP in order to provide appropriate security of the customers but also its own assets. Multi-technology systems are deployed in strategic network locations for this purpose, such as Network Intrusion Detection systems (NIDS) and Network Intrusion Prevention Systems (NIPS) that essentially combine the NIDS real-time threat detection with linkage to firewall rules in order to block those threats. These systems are based on anomaly detection techniques to detect anomalies and deviations from normal behaviour and block the untrusted data packets before they reach the hosts. This approach allows not only protection against known attacks, which could very well be avoided by the firewall rules, but also against unknown attacks in some cases. A lot of machine learning techniques have been proved to perform well in intrusion detection including Neural Networks (CNNs or MLPs), SVMs, Naive Bayes, Decision Trees and Logistic Regression [16].

Finally, inside the machine itself, where the customer is responsible for the system's security there are some more measures that could be taken into consideration. Using only the services needed, keeping them up to date and having a strict control over the services that are exposed to network ports is essential in order to reduce the attack surface. Often security scanning for viruses, malware or rootkits inside the system is also encouraged through Antivirus systems. However, the customer should be careful and monitor the overhead added from these host or application based protection mechanisms, so that the platform's availability is not compromised.

Table 4.9: Cloud/Edge Infrastructure Asset Security Measure Checklist

Cloud Infrastructure Asset Security Measure Checklist	
Physical Security	Multi Factor Authentication Access Control (Biometrics)
	Camera Surveillance
	Well equiped server rooms
	Natural Disaster Resilience
Client Relationships	Integrity, Confidentiality with client data
	Secure client management platforms
	Client asset monitoring and communicated problem resolution
Hosting Management	Vm isolation and resource limits
	Suggestion for dedicated machines in production
Cyberattack resilience	Intrusion prevention systems (IDS, Firewalls)
	Web Application Firewalls (WAFs)
	Antivirus systems inside the machine

Chapter 5

Case of the SYNAISTHISI platform

The fourth chapter was dedicated to defining a structured methodology of decomposing an Internet of Things ecosystem, identifying the assets that assemble it and define baseline security requirements and mesaures for each of these assets. An emphasis is given on the assets that compose the IoT platform of the ecosystem, and in this chapter the methodology is put on the test in order to secure a real internet of things platform called SYNAISTHISI [3, 55].

5.1 Methodology application on SYNAISTHISI Platform

Firstly, it is important to describe SYNAISTHISI from a higher level perspective before attempting to assess its security posture. SYNAISTHISI is a research IoT platform developed by NCSR 'Demokritos', and is a project in an alpha release stage at the time of writing this thesis. The platform's strong points are two key features, its support of multiple application layer communication protocols that make it suitable for a plethora of different internet of things devices, and its capability of the users creating their own standalone or pipelined services to handle and apply logic to the IoT devices' data. The platform is docker-based making it easy and fast to deploy and it combines various technologies in order to achieve its purposes. In this chapter, we will initially identify the assets of SYNAISTHISI that correspond to the asset taxonomy of our defined methodology. Then, we will focus on each of the chapter assess security gaps and security solutions that are already implemented in the platform. Lastly, we will propose countermeasures to solve any identified insecure factor of the platform, and wherever possible implement the solution programmatically.

5.1.1 Identifying the Assets

Starting off with our methodology, we have to decompose SYNAISTHISI into assets that correspond to our asset taxonomy. Since SYNAISTHISI is an IoT platform we will not address the assets regarding Users (and human factor in general), Devices and Deployment Infrastructure (Cloud/Edge). Thus, we have to decompose the platform into the message brokers, the web interfaces, the backend, the databases, the IoT services/applications and the communication channels. Before we delve into it, however, we have to present some technical characteristics first. The

evaluated version of the platform is not a finished product, and is currently on Alpha version. In the version we evaluate, the platform has its different components organized into docker containers, and deployed using the docker-compose command.

Firstly, we must identify the web user interfaces asset, meaning the web applications that are exposed to the internet, viewed by and offer multiple functionalities to the users of the platform. The platform's main web portal and graphical user interface is an Angular frontend application connected to a backend Python Flask application, exposed at port 80 of the server where the platform is deployed. Through this portal, the user can login and register for an account, as well as perform a variety of actions. These actions include service creation and management of any type, including S-type (Sensing) services, P-type (Processing) services and A-type (Actuating) services, depending on the service's functionality. An Nginx web server is the forefront of this app, acting as a reverse proxy, receiving the HTTP requests for the GUI or authentication/authorization requests and forwarding them to internal routes. The second web interface exposed to users is a Node-RED interface, a flow editor that allows users to pipeline flows (services) in order to create complex processing logic for their device's data.

Moving on to the Message Brokers asset, which is the main entry point for device data, there are a variety of solutions used by the platform for this purpose. One of the platform's main goals is interoperability, meaning retaining the ability to connect and use data from a variety of different IoT devices or other third party platforms using different application layer communication protocols (ALPs). Consequently, three different open source applications are used, specifically the popular RabbitMQ message broker, Eclipse Ponte and Eclipse OM2M. RabbitMQ exposes port 5672 for the AMQP protocol, OM2M exposes port 8080 for the OneM2M protocol and Ponte exposes three ports for protocols, 3000 for HTTP REST and Websockets, 5683 for CoAP and 1883 for the MQTT protocol. The Ponte external MQTT endpoint is internally bridged with the MQTT endpoint of RabbitMQ, as well as with OM2M in order to translate the messages to other protocols.

Regarding databases, the platform uses two different systems. The first and most useful one is a PostgreSQL instance, where most information reside, organized into three major components referring to User data, Topic data and Service data. This means that this instance holds all the information about the users, the data coming from or being delivered to devices or other forms of output (topics), and logic that the users either implement by themselves or are authorized to use from other users. The second data storage system incorporated by the platform is RDF4J, a Java-based framework (runs using an Apache Tomcat web-server in the SYNAISTHISI case) used for storing and querying semantics data using SPARQL endpoints. The ontologies stored in the RDF repository enable the back end to perform a variety of tasks including semantically annotating internet of things resources.

The back-end asset that orchestrates the functionality and provides the API functionality of the platform is a Python Flask application. Apart from the interconnection with the two data-stores (PostgreSQL, RDF4J), the back end is interconnected with the rest of the assets and the Internet through the Nginx gateway mentioned before. Since the back end has a publicly exposed API, the Nginx web server acts as an API gateway which can be useful in regard to security configurations in order to differentiate the business logic (back end) with the security logic (Nginx) as better as

possible.

The IoT services/applications asset that SYNAISTHISI offers to the platform users is topic-based. Each service is associated with at least one topic either input topic or output topic, which typically means that a service must at least produce functionality on some input, or provide output that another topic or an actuation subscriber will use, depending on the authorization fields in the Database. The platform provides the opportunity for using any type of language for the source code of the service, as the user uploads along with the source code, a Dockerfile containing information regarding the environment that the application will run on. Using this Dockerfile, a container is spawned and put in the topic pipeline depending on the configurations made.

Finally, in the communication channels asset in our methodology (asset number 3), we emphasized heavily into cryptography and TLS for secure communication. The platform, still in development, has no cryptographic techniques implemented for the security of communications at the time of writing this work. This includes the two GUI endpoints as well as the Back-end API and the message broker endpoints.
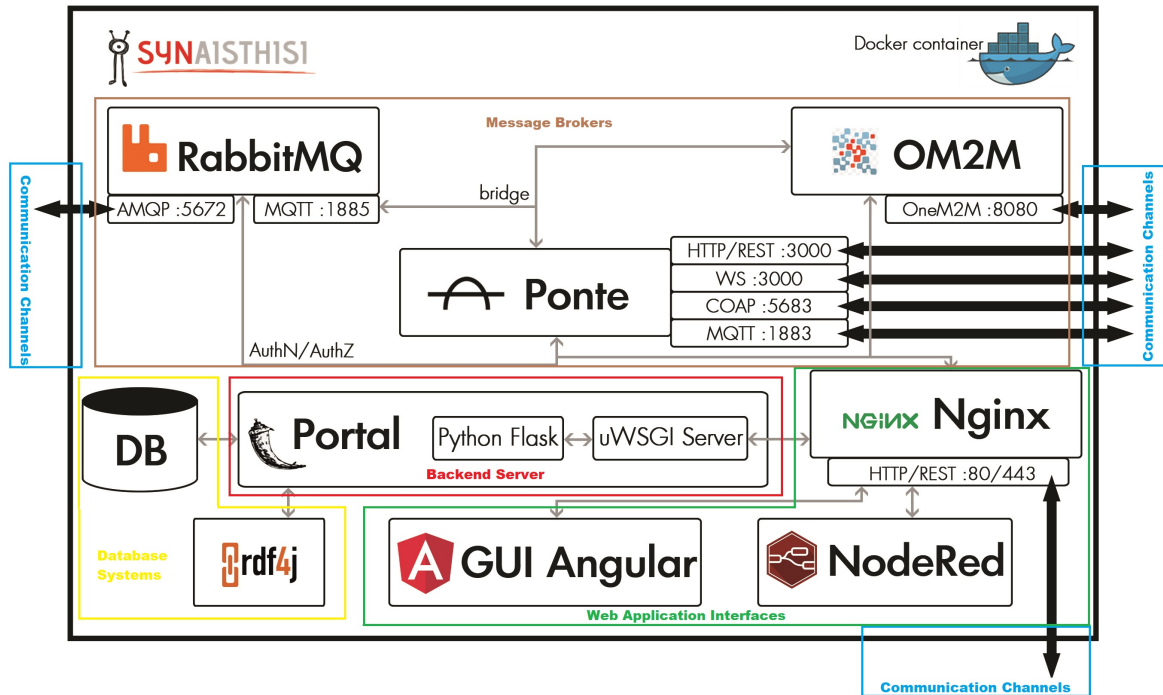

Figure 5.1: Asset Taxonomy of SYNAISTHISI based on the diagram in [3]

5.2 Security Feature Assessment and Extensions

After having decomposed the SYNAISTHISI platform into the parts from our methodology we are ready to assess their security posture, find security gaps and applied security controls. Using the asset security requirements defined in the fourth chapter, we focus on each asset and evaluate the security features that the chosen software solutions and custom software implementations offer,

as well as the requirements that are not satisfied yet. For the identified insecure factors, this thesis provides propositions and code/configuration contributions.

5.2.1   Security posture of the Communication Channels

Starting off with the communication channels (outlined with blue in the figure 5.2), we must address the communication between users or devices and the endpoints of the platforms that are exposed to the internet. These endpoints include the Nginx endpoint in port 80 which is the gateway for the main web GUI and API, the Node-RED instance at port 1880 and the message brokers at ports 1883 (MQTT), 3000 (HTTP REST, Websockets), 5672 (AMQP), 5683 (CoAP) and 8080 (OneM2M). As mentioned in the respective subsection in our methodology in chapter 4, these channels of communication should be resilient against eavesdropping and injection through cryptography and specifically TLS. At the current version of the platform none of these endpoints is secured using TLS or another type of cryptography, consequently a malicious actor can sniff sensitive information like usernames and passwords from these particular channels. In order to prove the vulnerability more clearly, we will try authenticate ourselves by sending a POST request to /mqtt/auth to use the API provided by the platform.

Figure 5.2: API not protected with TLS : Wireshark sniffing

In this situation, using the open source network protocol analysis and sniffing tool Wireshark (www.wireshark.org), we are able to read sensitive credentials in plain text by sniffing on the communication between the victim and the web portal. The communications exposed to the internet are in dire need of encryption using TLS.

For this work we configured the use of TLS in Nginx, to protect the web portal and the API provided by the Back end, to the Node-RED GUI as credentials are also used there, and one of the brokers - Ponte (MQTT). Needless to say, all of the endpoints need appropriate encryption measures to be considered resilient to eavesdroppers. Also, for experimentation purposes, the TLS certificates are self signed which means that we are the certificate authorities that verify the certificates. Computer systems have hardcoded trusted CAs like Sectigo or DigiCert which means that self-signed certificates are deemed insecure from browsers or command line tools like curl.

For the purposes of this work, self-signed certificates are a quick and easy way to implement and test TLS security but in a production environment trusted certificates must be used.

For Nginx configurations, firstly port 443 needs to be exposed for HTTPS communications. Thus in *docker-compose.yml*, the line — ''443:443'' was added in order to expose port 443 of the Nginx container and bind it to the 443 port of the host server. In the Dockerfile specifying information about the deployment of the Nginx container, we add the following lines that use openSSL to create a private key and a certificate which are put in a specific directory where Nginx can locate them and then OpenSSL utility is removed from the container. We use a 4096 bit RSA key to create a certificate with the information of NCSR 'Demokritos' and an expiration date of ten years from the issuing of the certificate.

```
1    RUN mkdir /etc/ssl
2    RUN apt-get update
3    RUN apt-get -y install openssl
4    RUN openssl req -subj '/C=GR/ST=Athens/L=AgiaParaskevi/O=
         NCSR Demokritos/OU=IIT/CN=localhost' -x509 -nodes -days
         3650 -newkey rsa:4096 -keyout /etc/ssl/key.key -out /etc/
         ssl/cert.crt
5    RUN yes | apt-get remove openssl
```

Finally, configurations were added for the use of TLS into the configuration file of the Nginx server. Port 80 and HTTP redirects with status 301 to HTTPS in port 443. For HTTPS the certificate and key we created before are used and the accepted TLS versions are the non-deprecated versions 1.2 and 1.3.

```
1    server {
2      listen 80;
3      return 301 https://$host$request_uri;
4      }
5
6      server {
7        listen 443 ssl;
8        ssl_certificate     /etc/ssl/cert.crt;
9        ssl_certificate_key /etc/ssl/key.key;
10       ssl_protocols       TLSv1.2 TLSv1.3;
11       server_name localhost;
12       ...
```

For the Node-RED endpoint in port 1880, the same procedure was followed regarding the Dockerfile where the key and certificate are being created and put into a certain directory where Node-Red can access and use them. The configurations for the Node-Red server in order to use TLS were put inside the *settings.js* file.

```
1    ...
```

```
2      https: {
3            key: fs.readFileSync('/etc/ssl/key.key'),
4            cert: fs.readFileSync('/etc/ssl/cert.crt')
5        },
6
7      // The following property can be used to cause insecure HTTP
              connections to
8      // be redirected to HTTPS.
9      requireHttps: true,
10       ...
```

Last but not least, brokers must also be secured since every message carries user credentials as well as potentially sensitive messages. The Ponte platform has the endpoints for MQTT, WebSockets and CoAP, and internally MQTT is relayed to the RabbitMQ broker via the port 1885 of the broker which is not exposed to the public. For this work we secure MQTT and enable the use of MQTTS, which is the version of MQTT that makes use of TLS encryption. We first create a certificate and a private key, which are copied inside the container during the container build, and we add the following configurations so that Ponte is listening to MQTTS connections at its default port, 8883.

```
1      ...
2      http: {...},
3      coap: {...},
4      mqtt: {
5       secure : {
6         keyPath:"/key.key",
7         certPath:"/cert.crt"
8       },
9       ...
10     }
11     ...
```

This enables the connection with the endpoint through TLS, as well as the authentication and authorization of the client sending data, thus sensitive information are secured during the communication. It is important to also note that since components are interconnected, changing an endpoint to use encryption, means that each code requesting content from these endpoints is also changed to use the HTTPS - MQTTS protocol for this communication. For a proof of concept that validates the security applied by TLS in the communication channels, we can use Wireshark like before. An API POST request is sent to the route /mqtt/auth in order to authenticate ourselves. This time, however, the request is encrypted since the Nginx web server that acts as a Back end API gateway uses TLS, thus we cannot read the sensitive credentials as seen in figure 5.3.

```
29 7.898327270   192.168.1.159      192.168.1.101     TLSv1.3   285 Application Data
30 7.914386932   192.168.1.101      192.168.1.159     TCP        66 443 → 56582 [ACK] Seq=2309 Ack=598 Win=64768 Len=0 TSval=2683…
31 7.917600326   192.168.1.101      192.168.1.159     TCP        66 443 → 56582 [ACK] Seq=2309 Ack=817 Win=64640 Len=0 TSval=2683…
Encrypted Application Data: d7693fc06f94c61425d1f07d8ef0ba8ebbd21fe476efec01…

00 0c 29 f8 0e 04 d8 12   65 5e 19 c7 08 00 45 00    ··)·····  e^····E·
01 0f 86 cc 40 00 40 06   2e c8 c0 a8 01 9f c0 a8    ····@·@·  ········
01 65 dd 06 01 bb 46 eb   7e 02 7f 24 60 06 80 18    ·e····F·  ~··$`···
01 f5 93 ac 00 00 01 01   08 0a 0a 65 9f 8c 9f f5    ········  ···e····
94 e8 17 03 03 00 d6 d7   69 3f c0 6f 94 c6 14 25    ········  i?·o···%
d1 f0 7d 8e f0 ba 8e bb   d2 1f e4 76 ef ec 01 af    ··}·····  ···v····
c7 bd 40 d5 56 35 f3 37   63 b3 fe 8c 8f 0c 65 5b    ·@·V5·7   c·····e[
00 f5 94 aa 00 e0 1d 98   75 b3 58 49 88 4d 17 b1    ········  u·XI·M··
56 f8 5a 51 92 38 21 a5   34 d2 2b d3 c0 3b 7d 9a    V·ZQ·8!·  4·+··;}·
2e 16 00 a4 34 f9 c9 77   d2 98 e5 ec 33 8e 50 1e    ····4··w  ····3·P·
0c a5 43 dd c6 f7 7b c7   93 a8 ce 91 27 1f 10 1f    ··C···{·  ····'···
21 bd 74 5e 1e 50 2f ee   ec 96 24 64 40 aa 70 f3    !·t^·P/·  ··$d@·p·
d0 45 98 3f 09 b7 11 bf   44 5f cb c0 95 cf 6e da    ·E·?····  D_···n·
fc 05 b7 b7 e2 c9 15 4b   48 0c 57 7b 5b 73 45 eb    ·······K  H·W{[sE·
52 65 bb 2b e7 55 73 fc   0c 26 d5 d2 e5 a5 af 8d    Re·+·Us·  ·&······
c4 cf ec 2e a8 37 3b ce   97 2e 9e 03 3c 3e b3 fe    ·····7;·  ·.··<>··
ce 88 5b 54 d6 37 d9 89   5c cd a6 73 f1 c8 f5 48    ··[T·7··  \··s···H
2e 5f 27 7e d8 d0 9f b7   1a 14 cf c4 18             ._'~····  ·····
```

Figure 5.3: API protected with TLS : Wireshark sniffing

5.2.2 Security posture of the Message Brokers

The communication channels asset's proposition and implementation of TLS solves the major issue of confidentiality and protection from tampering. The communication between devices and brokers is now encrypted and secret from third parties eavesdropping.

Authentication, however, is a factor not implemented with a security mindset in this version of the platform. The PUB/SUB mechanism that the platform uses authenticates the devices by using the username-password credential pair inside the request that they send. A simple example of a PUB message to an existing topic is the following, using mosquitto client targeting the MQTT endpoint.

```
mosquitto_pub -h <hostIP> -p 1883 -t <topic> -u <username> -P <
    password> -i <id> -m "sample message"
```

It is easily noticeable that the user account credentials are included in every pub/sub request to the brokers for every device sending to or receiving data from the platform. For a high level user such as an organization using a plethora of IoT devices, this is a great vulnerability, as the compromise of a single device compromises the whole user account of the organization. As we proposed in the respective subchapter of our methodology, the solution to that can vary, but the two solutions we provide are either tokens, different for every device that will be included into every request and will identify the device, or two-way identification within TLS where every device should possess a TLS client certificate either issued by the platform or trusted by the platform. The latter is the most robust solution if implemented correctly, but it is also associated with a number of caveats like the mechanism through which the user registers a new device, and CA management if the design choice renders the platform as the issuer of special certificates for device authentication. The first solution is more easily implementable and usable in the case of e.g. weak IoT devices such as sensors where the configuration and use of a client TLS certificate introduces yet another level of complexity to achieve.

Firstly, we should perform a simple example to see how a device's pub/sub request works with the platform. For this reason we execute an example provided by the platform developers, which includes two topics and one processing service. The input topic is called *camera_stream*, and is

an image coming from a webcam, and the output topic called *faces_count* is an integer describing
the number of people that the processing service recognises in the picture using computer vision.
Both of the python scripts that interact with the topics use user credentials:

```
1 python3 sType.py --username sevangelou --p 123456 --
     output_topics /camera_stream --delay 5 --camera_index 0
2 python3 aType.py --username sevangelou --p 123456 --input_topics
     /faces_count
```

This example runs well, and produces the following logs and results:

(α´) Ponte and Nginx Logs

(β´) aType script logs, results from faces_count topic

Figure 5.4: Face Recognition SPA example

1. Client does the PUB/SUB request containing credentials.

2. Ponte receives the request.

3. Ponte asks Flask API for authentication of the user.

4. On successful authentication, Ponte asks Flask API for authorization of the user against the
   topic.

5. On successful authorization, Ponte forwards the request to RabbitMQ, and moves on.

We need to (a) provide a way for the user to create new device tokens and (b) authorize his devices using them. So, first, an HTTP REST endpoint was implemented , where a user can POST with his credentials in order to create a new device token to authenticate his devices. These tokens are created when the endpoint receives the request for the token generation and after firstly authenticating the user, by using the SHA1 hashing algorithm into the combined string of a time-stamp using *time()* from the time module and a truly random float number using *SystemRandom()* from the random module. Then, the script appends the token inside the database's *users* table, so that the user can authenticate his devices using these tokens. Specifically, the code that generates and appends the token in the database is the following:

```
1  class MosquittoDevToken(Resource):
2      '''path : /mqtt/devtoken'''
3      def post(self):
4          '''
5              @param1: 'username' (user to authenticate)
6              @param2: 'password' (user password or token)
7          '''
8          auth_parser = reqparse.RequestParser()
9          auth_parser.add_argument('username', type=str)
10         auth_parser.add_argument('password', type=str)
11         auth_data = auth_parser.parse_args()
12         mqtt_user = UserModel.find_by_username(auth_data.
               username)
13         if mqtt_user:
14             if mqtt_user.authenticate(auth_data.password):
15                 time_now = str(time.time())
16                 truly_random = str(random.SystemRandom().random
                       ())
17                 hash_object = hashlib.sha1((time_now+
                       truly_random).encode())
18                 unique_dev_code = hash_object.hexdigest()
19                 mqtt_user.append_token(unique_dev_code)
20                 return unique_dev_code, 200
21         return None, 403
```

Internally, the API endpoint */mqtt/devtoken* is used to create the code, append it in the token list inside the database and return it to the user, and the API endpoint */mqtt/devtoken/auth* is used to authenticate the device using this token, which is only usable from Ponte, and cannot be used for logging into the portal or the Node-RED utility. The latter endpoint makes use of the following authentication function that checks both if the password given is the user password and if it is a token included to the user's list of tokens:

```
1  def authenticate_dev(self, token):
```

```
2          token_arr = self.device_tokens.split(',')
3          return (token in token_arr) or bcrypt.
              check_password_hash(self.password, token)
```

After testing the same example with the use of the new device tokens, we make sure that it works:

(α´) Token generation example

(β´) face recognition example using tokens

Figure 5.5: Face Recognition SPA example with tokens

Another aspect that demands close attention at the asset of message brokers, according to our methodology, is authorization. The platform has a robust custom solution for authorization, based on the PostgreSQL database. There are three tables, namely *services_topics_map*, *users_topics_map* and *users_services_map* where each row associates two different entities showing that the first entity has access and is permitted to use the second entity. Consequently, whenever a message broker gets a PUB message towards a topic, or a SUB message to get updates on a topic, the server requests the back-end for an authorization check on the user (from the credentials included in the message or the token) and the associated topic. This way the platform enforces access control successfully and securely.

5.2.3   Security posture of the Web Application Interfaces

The main web portal of the platform is a mixture of an Angular v. 7.0.1 front-end web application and a Python Flask v. 1.0.2 back-end application exposed through an Nginx web server to the internet. Having covered the encryption of the communication between user and application in the communication channels subsection, we will focus on the rest of the web application's potential insecurities.

Starting from the register screen, the user has to submit a username, an email, a password and an acceptance of the user terms in order to create an account. User enumeration cannot be performed
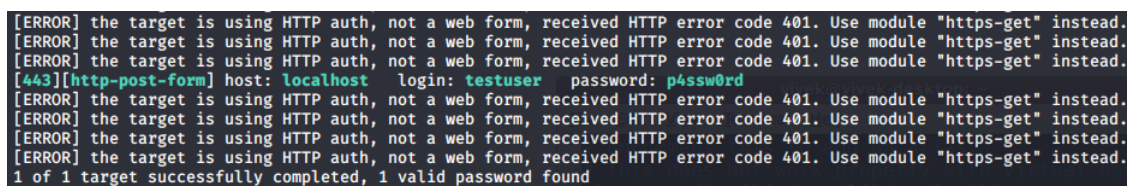
since the user needs to finalize the submission before knowing whether the username or email used already belongs to another user. The errors do not disclose information to a potential attacker, and are only giving information on whether the username or email is already taken (after submission) or the password is weak according to the application's standards. While these standards of at least 6 characters that are enforced by the application ensure that the password is adequately lengthy, the password policy allows the use of number-only passwords such as '123456'. In order to make the passwords of the platform more robust, more password policies must be enforced, like not having part of the username or email inside the password, and having at least 3 of the 4 categories in capital letters, lowercase letters, numbers and symbols.

Moving on to the login screen, there are a number of aspects that need to be assessed. The simplest is information disclosing error messages. A wrong input of username-password pair returns an error message of invalid credentials, not indicating whether the username-email or password is wrong, thus preventing user enumeration. The second aspect that needs to be tested is SQL injection. The login screen is the most straightforward POST request that results to an SQL query in the back-end, and is the most important place that needs to be resilient against SQL injections. Since the back-end is also an asset described in our taxonomy, it is better to leave the conversation for the back-end subsection (5.2.4) of this chapter.

Brute-forcing credentials is also an issue that needs to be tackled in the interface asset. Knowing a user's username or email, someone could be able to try different passwords sequentially in order to gain access to a user account that uses a weak password with dictionary attacks where a list of common passwords is utilized to find potentially used passwords. In order to assess whether the application is vulnerable to bruteforcing we are going to use THC-Hydra a parallelized login cracker - that apart from HTTP can also bruteforce FTP,SSH and many other protocols. The command run with hydra is:

```
hydra -I localhost -s 443 https-post-form "/login:username=^USER
    ^&password=^PASS^:Invalid" -l testuser -P /usr/share/
    wordlists/rockyou.txt
```

In this command, we brute-force the account of the user with username *testuser*. We are inserting into Hydra the request's format with USER and PASS keywords where the changes have to be made, and we input the *rockyou.txt* wordlist, a well-known wordlist containing over 14 million of commonly used passwords. The command spawns 12 threads (default) and sequentially tries logging in with the passwords of the wordlist.

Figure 5.6: THC-Hydra achieves to bruteforce a user's password

As we can see, hydra has found the password after around 8000 attempts, and the malicious

actor can now overtake testuser's account. Mitigation for this includes various methods, with the top options being recaptcha puzzles (which we cannot implement since we have no access to frontend code), account locking for a certain amount of time, and rate limiting/throttling. In this work we will use the third solution in order to limit the amount of login requests through rate limiting. This solution is explored in the next subchapter as an API rate limiting mitigation towards reinforcing the platform's availability.

Moving on after logging in, the user receives a JWT token. JWT or Javascript Web Tokens are a popular authentication mechanism where a json object containing information about the user used for authentication and authorization is encrypted using a secret key from the backend. Because this token is digitally signed by the server (with an HMAC algorithm), whenever the user requests content, this token authenticates him and is used as a means to check if he is authorized to receive the requested content. These tokens have expiration dates so that they are invalidated after a user logs out or some time has passed. Upon decoding the token and converting the exp field (expiration date) to human-readable we can see that the token expires 15 minutes after its creation, which is a very small timelapse. To avoid enforcing a log out on the user, the platform also makes use of Refresh Tokens, jwt tokens that enable the client to create new jwt tokens without going through the process of re-logging in. This is more secure than issuing a a long expiration date or even worse a never-expiring jwt token, and makes the interface friendlier and more scalable.

The web portal of SYNAISTHISI uses the Angular Framework. This modern Javascript framework is entrusted to protect the web application against XSS, since it offers built-in protection against some common web application attacks (angular.io/guide/security) including XSS. As discussed in the official documentation, Angular promotes the use of templates where, through interpolation, the content is automatically sanitized and escaped by the framework. This way, malicious scripts or commands that a potential malicious user had inserted do not execute as they would with DOM XSS. Angular's HttpClient, used to perform HTTP requests to the backend also provides a client-side security against Cross-Site Request Forgery (CSRF or XSRF) attacks. In CSRF, a malicious actor uses another user's cookies (how they are obtained is out of scope in this discussion) in order to perform actions as if he were the victim user. Unfortunately, in order for Angular's CSRF protection to work, the backend needs to provide the client-side implementation of the mechanism, and by code auditing there are no created XSRF-TOKENS in the JWT cookie generation.

Of course, just from code auditing and looking at specific points it is not possible to holistically discover security issues inside a web application. Dynamic Analysis tools can be a significant supplement into discovering gaps and consequently fixing them. For this purpose, we employ Owasp ZAP (owasp.org/www-project-zap), a popular open source dynamic analysis scanner and crawler, and scan the web portal once without authentication, and once using the jwt token we receive after logging in.

The first (unauthenticated) scan, does not seem to report any severe vulnerabilties of the platform as summarized in table 5.1.

1. The first informational issue is a false positive, since the timestamps reported are just numbers that have the same length as a timestamp value. To validate that, three of the found values

Table 5.1: Owasp ZAP Unauthenticated Scan Report

id	severity	issue_text
1	Inform.	A timestamp was disclosed by the application/web server - Unix
2	Inform.	The response appears to contain suspicious comments.
3	Low	The Anti-MIME-Sniffing header X-Content-Type-Options not set to 'nosniff'.
4	Low	A private IP has been found in the HTTP response body.
5	Low	The cache-control and pragma HTTP header have not been set properly or are missing allowing the browser and proxies to cache content.
6	Medium	X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.

classified as "timestamps" were checked and they correspond to years with no association e.g. 1974, 2038 etc.

2. The comments found are of no security significance.

3. The nosniff value on the X-Content-Type-Options mitigates MIME-based attacks where older browsers could sniff a response and override the Content-type set by the server in order to render content as a different type. While in SYNAISTHISI this is not a severe issue since no intricate content is served, we still add the Header in the Nginx configuration for that extra protection.

4. The private IP found inside the code does not correspond to the internal network deployment of the production platform so it is not a security issue.

5. The fifth issue of Low severity addresses the cache control of resources. Sometimes it is preferable to cache resources in order make the web application faster and provide a better user experience. However, the information cached might be sensitive ones, which could be compromised if a malicious actor had access to the browser of the victim before the expiration of the cache. To mitigate the following code was added in the Nginx configuration file, to create the $expires variable which handles the cache directives:

```
1    map $sent_http_content_type $expires {
2        default                    -5d;
3        text/css                   max;
4        ~image/                    max;
5    }
```

The configuration forces browsers to not cache resources as a default policy (because of the negative sign), whereas in the case of CSS styling files and images, the cache stores them for as long as possible as they usually pose no security risk and it is inefficient to request them every time.

6. Same as the third issue, this is not a severe security issue because of the content that the platform serves, since no iframe elements and relevant content are shown to the users. Nevertheless, we also add this header with the value "DENY" in the Nginx configuration to apply the protection needed.

After applying the configurations that mitigate the non-severe issues that ZAP reported, these are the headers of a request for the web-page of the SYNAISTHISI portal.
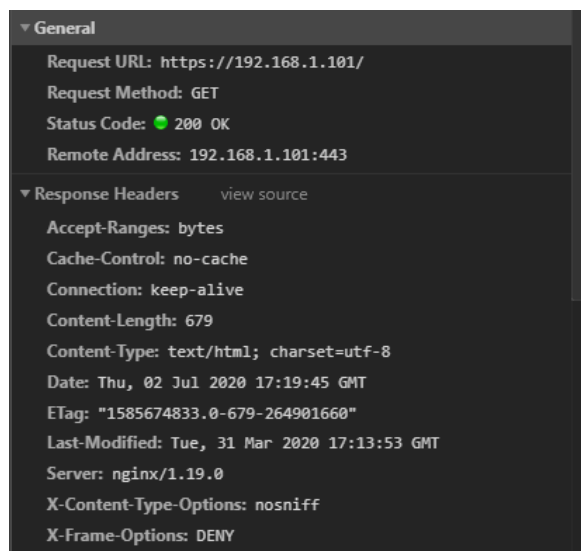
Figure 5.7: Headers of the GET Request on the Web Portal

Moving on, we are using Owasp Zap with Manual scanning, a mode where ZAP acts as a proxy for the browser, receiving and analyzing the requests and responses in order to root out vulnerabilities. This also allows for authenticated penetration testing by logging in manually and performing actions as a regular user, letting ZAP perform the dynamic analysis of the web application by intercepting and modifying requests in order to evaluate security robustness. After moving through each possible user interaction with the application, ZAP has reported the following three more issues:
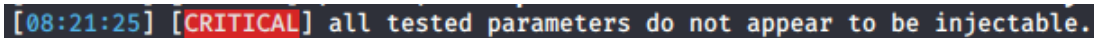
Table 5.2: Owasp ZAP Manual Scan Report

id	severity	issue_text
1	Low	A XSS attack was reflected in JSON response, this might leave content consumers vulnerable to attack if they don't appropriately handle the data.
2	Medium	A Format String Error occurs when the submitted data of an input string is evaluated as a command by the application.
3	High	SQL Injection might be possible. Page results were successfully manipulated using the boolean conditions [name AND 1=1 –] and [name AND 1=2 –]

1. The XSS reported is a false positive. ZAP's active attack created a topic with a name where embedded javascript was put. On getting the topics, the *<script>alert(1);</script>* still exists, which made ZAP regard it as a successful reflected XSS attack. However, the JSON is not rendered inside HTML where the script could be executed, thus it is not an issue. Upon searching we can find the topic with that name, where the name is rendered as a literal string value.

2. The format string error found actually leads to an error disclosure bad practice. Looking at the error we can identify the use of PostgreSQL as well as the exact query used. The error is easily reproducible by using a large name value when creating a new topic. A solution

"An Internal error occured: (psycopg2.DataError) value too long for type character varying(50)\n [SQL: 'INSERT INTO topics (name, description, topic_ontology) VALUES (%(name)s, %(description)s, %(topic_ontology)s) RETURNING topics.id'] [parameters: {'name': 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 'description': 'fmeoafme', 'topic_ontology': 'anyURI'}]"

Name	Description	Topic Ontology

Add   Clear

Search in topics...

Figure 5.8: Application Error Disclosure

needed is adequate input validation that will return customised error messages in case of similar behaviour, rather than a 500 Internal Server Error message along with the Exception description.

3. The reported SQLi issue is a false positive as well. While there is no evident explanation for the false positive, every attempt of using ZAP indications to reproduce the attack failed, and a standard expected error appeared for already existing topic names, as the boolean conditions were used to create services or topics with SQL inside the name value.

5.2.4 Security posture of the Back-end

The backend is associated with the APIs, the interconnection to the platform's components and databases. A lot of the frontend's actions, as well as the API requests exposed by the documentation for the users, result to actions taking part in the back-end, querying the databases or performing functionalities.

For the first part, it is important that these API requests are resilient against SQL injection attacks, as discussed also in the subsection for the web interfaces. Let's take for example the */login* HTTP POST request that is used to login as a user by providing credentials. To assess SQLi resiliency, the tool SQLmap was used, that takes the url and input format of the data and performs automated SQLi payloads in order to get information about the inner structure of the database, and if possible perform custom queries directly. After intercepting a simple login attempt from the

front-end, we can see that the button submit sends a POST request with json data at the route */login*, which is also available as an API call by the platform. The command run with the SQLmap tool in order to check for potential SQL injections, where *login_request.txt* was the request we intercepted at our login attempt with wildcards in the injectable parameters, is:

```
1  sqlmap -r ../login_request.txt --level 5 -a
```

Thankfully, none of the attempts performed from SQLmap resulted into information or even worse query results.

Figure 5.9: SQLmap fails to discover an SQL injection in the login form

The SYNAISTHISI back-end queries the PostgreSQL database through a middleware called SQLAlchemy. SQLAlchemy is essentially an Object Relational Mapper and by using the orm methods provided and avoiding raw SQL queries throughout the application, the web portal is deemed secure against SQL injection attacks, since parameters are automatically escaped by the SQLAlchemy middleware.

The back-end is mainly used for the APIs it exposes. The API in SYNAISTHISI is publicly exposed and described in the documentation inside the web application interface. Thus, the Same-Origin (SO) policy that browsers enforce is extended with a Cross-Origin Resource Policy (CORS) from the back-end server which is globally permissive, allowing requests from any Origin by defining the wildcard * symbol. Generally requests to the Back-end are allowed only to trusted domains, but in this case it is a design choice to allow these requests to everyone. Speaking of APIs, the platform does not apply rate limiting to the API requests that it receives, which could be a major drawback to its scalability and resiliency against DoS and DDoS attacks. To test whether such measure exists a python script was created, that spawns 5 threads that perform POST requests to the */login* endpoint for about 30 seconds.

```
1  import requests as r
2  import threading, time, sys
3
4  threadLock = threading.Lock()
5  counter = 0
6
7  def thread_function(name):
8      parameters = {"username": "sevangelou","password": "123456"}
9      while True:
10         res = r.post('https://localhost/login', params =
               parameters, headers={"accept": "application/json"},
               verify=False)
11         if res.status_code != 200:
12             continue
```

```
13        global counter
14        with threadLock:
15            counter += 1
16 threads = []
17 for i in range(5):
18     x = threading.Thread(target=thread_function, args=(i,))
19     x.setDaemon(True);
20     threads.append(x)
21 start_time = time.time()
22 for x in threads:
23     x.start()
24 time.sleep(30)
25 print("{} requests in time: {:.2f} seconds.".format(counter,
       time.time() - start_time))
```

The end result of this script is:

```
1 211 requests in time: 30.04 seconds.
```

This is calculated to be about 1 POST request each 142ms, or about 7 requests per second. It is easily noticeable that for many hosts, some of them much more powerful than the Virtual Machine used for the experiment, this could endanger the availability of the server and collectively the platform so it is wise to apply limits to the requests coming through, and NGINX as an API gateway provides an easy solution for this problem that can also reinforce protection to bruteforce attacks that were explored in the web application sub chapter.

Inside the *syndelesis.conf* file that is copied and used as the configuration file for NGINX, we add the following lines in order to apply rules for rate limiting of 1 request per second per unique IP address, for a total size of 10mB in the size of saved IPs - around 160,000 addresses according to the documentation. The rest of the requests that excess the 1 per second rule fail with 503 error. These rules are applied to the */login* path in our example, but can also be applied to other locations. Options for burst queued requests are also offered by NGINX but in order to give effective rules, the average traffic and population of the users must be computed, so that the values given neither compromise the platform's availability nor deteriorate user experience.

```
1 limit_req_zone $binary_remote_addr zone=loginlim:10m rate=1r/s;
2 server {
3     ...
4     location /login {
5         limit_req zone=loginlim;
6         try_files $uri @app;
7     }
8 }
```

Running the same script as before, we can see that the rule is successfully enforced:

```
1  30 requests in time: 30.04 seconds.
```

Languages can also hide different security issues and bad practices inside them, so it is always useful to apply static analysis to the source code in order to identify such issues. The back-end is a Python application and for this reason we use a Python static analysis tool called Bandit. The results of this tool are summarized in table 5.3, where we omit fields such as the respective faulty filenames and code sections as well as duplicate occurrences of issues:

Table 5.3: Bandit Static Analysis Results

id	severity	issue_text
1	LOW	Consider possible security implications associated with subprocess module.
2	LOW	Possible hardcoded password: "
3	LOW	Try, Except, Pass detected.
4	LOW	Use of assert detected. The enclosed code will be removed when compiling to optimised byte code.
5	MEDIUM	Use of possibly insecure function - consider using safer ast.literal_eval.
6	HIGH	A FTP-related module is being imported. FTP is considered insecure. Use SSH/SFTP/SCP or some other encrypted protocol.

1. The first issue addresses the use of the subprocess module which is used to spawn and control processes in Python. The use of this module presents many security caveats, although in the case of SYNAISTHISI it belongs to commented code of a previous version where services were run in the server as processes and not within spawned Docker containers, which was a much riskier approach.

2. The second issue is a false positive of the tool, addressing a default empty value in a string variable that later holds the created token of a user.

3. The third issue reveals a bad practice where a caught Exception is not handled, and the keyword *pass* allows the exception to be ignored. Instead of a *pass*, a logging command was added to reveal the error in the logs in case that these Exceptions occur.

```
1      ...
2      except Exception as e:
3          self.app_logger.error(e)
4          # pass
```

4. The last LOW severity issue addresses the use of *assert*, which when compiled into optimised python byte code (.pyo) is removed, thus its code is not executed. Inside the platform, assert is used to enforce username and email policies and avoid symbols like "__". After simply creating a username where the assert statement should be enforced, we receive a 400 Bad Request status code, which means that the assert statement's code was executed and the issue was a false positive.

5. The MEDIUM severity issue is regarding the use of the function *eval* that is used to parse the expressions it is given and execute them as Python code. This allows for the developer to turn strings into objects, for a example the string "{'value':'value'}" into a dict with these values. However, this capability of executing python code makes it susceptible to malicious input, as in figure 5.10.

```
>>> output = eval("__import__('subprocess').getoutput('pwd')")
>>> output
'/home/sevangelou/skel-iot-platform/docker_compose/flask_app'
>>>
```

Figure 5.10: Executing arbitrary commands through Python's eval

For this purpose we substitute its use with the more secure *literal_eval* function of the ast module as suggested from Bandit. In the case of malformed string inputs, a 400 Bad Request is returned to the user after catching the ValueError exception raised from the function.

6. Finally the HIGH severity issue addresses the use of FTP included from the *ftplib* module. Inside the platform this is used to upload created services into an FTP server that will be used as a Service Marketplace in the future of the platform. The issue is correct to suggest an encrypted protocol in order to upload these files, as FTP is vulnerable to eavesdropping because it communicated with unencrypted packets. Thus, when the marketplace is implemented it should use a more secure protocol such as the proposed SFTP and SCP.

5.2.5 Security posture of the Database Systems

Databases are the data stores of the platform, and SYNAISTHISI employs two kinds of databases, a relational SQL one, PostgreSQL, and a semantic store using SPARQL in RDF4j as we can also see in the diagram at 5.1. RDF4j is used as a storage for ontologies that the platform uses in order to create, edit, categorize and use topics and services. These ontologies, as a design choice, do not require authentication to be requested for because they do not contain sensitive data. Thus, apart from keeping the store from being exposed to the internet, there are no other security measures targeted at RDF4j, as the potential impact of a breach or leak is minimal.

PostgreSQL is the database that needs to be protected most in the platform. The database contains information about users, passwords, tokens, services, topics and many more. The first and foremost step is not exposing the database to the Internet, and a simple Nmap command to the server reveals that the Database at port 5432 is indeed exposed in Figure 5.11. Consequently, we can use the *psql* command to remotely connect to the database, and generally interact with it in a lot of ways.

The easy and most correct solution to this issue would be to avoid mapping the postgreSQL container to a host port from the *docker-compose.yml* file, so that the database is not listening to outside connections on port 5432 of the host. The other containers would have to query the database using the container's IP, given to them as an environment variable during the docker-compose.

Thus,the following lines are removed from the docker-compose YAML file:

```
1  ports:
2  - "5432:5432"
```


Figure 5.11: Nmap scan at SYNAISTHISI server

Now, outsiders cannot inspect the use of postgreSQL using Nmap. If the deployment of the platform is distributed and postgreSQL needs to be accessible from the Back-end which belongs to another server, we can make use of the *iptables* firewall utility, and apply the following firewall rules in order to make postrgeSQL inaccessible for anyone else except the domain where back-end exists (e.g. apisynaisthisi.com):

```
1  iptables -A INPUT -s apisynaisthisi.com -p tcp --dport 5432 -j
      ACCEPT
2  iptables -A INPUT -p tcp --dport 5432 -j DROP
```

Reviewing the *docker-compose.yml* file which contains information for the deployment of the containers, we can notice that the user created to handle the database is not a root user. This reduces the potential capabilities that an intruder would gain in the case of a succssful exploit, for example an SQL injection through the web application.

Moving on, it is valuable to make sure that sensitive information such as passwords are hashed, so that in the case of a leak no accounts are compromised straight away. After logging in and querying some user accounts inside postgreSQL we can see the following:

Figure 5.12: Hashed passwords in db

We can see that the password fields are encrypted using a hash. Just by looking into the hash's form we can identify a common first part between the hashes, that being the *$2b$13$* substring. This indicates that the algorithm used for hashing is the BCrypt algorithm, with 2^{13} key expansion rounds. While bcrypt is a robust algorithm with no core vulnerabilities, choosing the number of key expansion rounds is a matter that needs attention. The developers should employ a number that is the highest possible for which the server performs adequately and as expected. A very high

number would make actions like verifying a password take a lot of time, and possibly endanger the server with loss of availability, but a low number could make bruteforcing faster and possibly more successful.

Looking at global encryption of the database information, in this case it would be an inefficient choice. The brokers request data from the PostgreSQL container at every device message for authentication and authorization purposes, and it is possible that availability of the platform would be compromised if cryptographic functions were run at each request, as especially the requests to the brokers can scale both in amount of requests as well as minimal time distance between consecutive requests.

5.2.6 Security posture of the IoT Processing Services

The last asset and possibly the one most difficult to protect against, is the asset that addresses the services that users can create and upload to the platform. In this asset the developers of the platform are met with a major tradeoff in security and usability. A more flexible service creation procedure enables the user to create complex application and use a plethora of different languages and frameworks, but this freedom can lead to a massive security breach in the case of malicious activity.

So far, the platform works with Docker containers. Whenever users attempt to create a new service, they are required to provide the code of the service, any other required file, and a Dockerfile with information about the creation of the Docker container. In order to validate our security concerns, we will create a custom service as a regular user and try to execute code that should not be allowed to execute. For this experiment we create a processing service using the Dockerfile from the SPA example we explored above and a Python script receiving a string from its input topic */command* which resembles a shell command, using the *subprocess* module to execute the command on the shell, and publish the command output to an output topic called */command_output*. In the local computer we run to scripts, the first receives the command from the standard input and publishes to the /command topic, and the second subscribes to the /command_output topic and on message prints the command's output. The main functionality of the processing script is the following code:

```
1   ...
2   def run_malicious_commands(command):
3       process = subprocess.Popen(command.split(' '),stdout=
           subprocess.PIPE,stderr=subprocess.PIPE)
4       output,err = process.communicate()
5       client.publish(out_topic, output)
6
7   # Callback to handle subscription topics incoming messages
8   def on_message(client, userdata, message):
9           msg = message.payload.decode("utf-8")
10          run_malicious_commands(msg)
```

[11]   **. . .**

After deploying it, we send 3 publish messages with the first python script with the commands **ls**, **pip install nmap** and **ifconfig**. As we can see in figure 5.13, all of those three commands execute successfully, receiving their output at the output topic.


```
connecting to broker  192.168.1.101
subscribing to topics
log:  Sending SUBSCRIBE (d0, m1) [(b'/command_output', 0)]
log:  Received SUBACK
log:  Received PUBLISH (d0, q0, r0, m0), '/command_output', ...  (1118 bytes)
OUTPUT##################################################
total 80
drwxr-xr-x   1 root root 4096 Jun 30 20:08 .
drwxr-xr-x   1 root root 4096 Jun 30 20:08 ..
-rwxr-xr-x   1 root root    0 Jun 30 20:08 .dockerenv
drwxr-xr-x   1 root root 4096 Jul 24  2017 bin
drwxr-xr-x   5 root root  340 Jun 30 20:08 dev
drwxr-xr-x   1 root root 4096 Jun 30 20:08 etc
drwxr-xr-x   2 root root 4096 Jun 25  2017 home
drwxr-xr-x   1 root root 4096 Jul 24  2017 lib
drwxr-xr-x   2 root root 4096 Jun 30 19:45 log
drwxr-xr-x   5 root root 4096 Jun 25  2017 media
drwxr-xr-x   2 root root 4096 Jun 25  2017 mnt
drwxr-xr-x   4 root root 4096 Jul 24  2017 opt
-rw-r--r--   1 root root 4174 Jun 30 10:59 pType.py
dr-xr-xr-x 286 root root    0 Jun 30 20:08 proc
-rw-r--r--   1 root root   18 Jun 30 10:46 requirements.txt
drwx------   1 root root 4096 Jul 24  2017 root
drwxr-xr-x   2 root root 4096 Jun 25  2017 run
drwxr-xr-x   2 root root 4096 Jun 25  2017 sbin
drwxr-xr-x   2 root root 4096 Jun 25  2017 srv
dr-xr-xr-x  13 root root    0 Jun 30 20:08 sys
drwxrwxrwt   1 root root 4096 Jun 30 10:59 tmp
drwxr-xr-x   1 root root 4096 Jul 24  2017 usr
drwxr-xr-x   1 root root 4096 Jul 24  2017 var
OUTPUT##################################################
Collecting nmap
  Downloading https://files.pythonhosted.org/packages/f8/6f/6813025bd575ebc771189afaab7c405fdf3f1febaa197525d5aa6fd88ac5/nmap-0.0.1-py3-none-any.whl
Installing collected packages: nmap
Successfully installed nmap-0.0.1
OUTPUT##################################################
eth0      Link encap:Ethernet  HWaddr 02:42:AC:12:00:0A
          inet addr:172.18.0.10  Bcast:172.18.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:61 errors:0 dropped:0 overruns:0 frame:0
          TX packets:62 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:22804 (22.2 KiB)  TX bytes:8994 (8.7 KiB)
```

Figure 5.13: Reverse Shell as a Malicious Service (output topic PUBs)

Apart from these commands, we can also download files using curl, and execute potentially malicious python or bash scripts. The user is also a root user inside the container which gives him a full range of capabilities inside the container, which before the user namespace security feature of Docker could lead to container escape and privilege escalation in the host. Lastly, all of the other production containers including the database, back-end and brokers are visible internally from the spawned service.

Since it is a design choice, the following measures will only affect the Dockerfile and the deployment of the container that will run the untrusted code, since constraining the services in terms of accessibility to the Internet or accessibility to language modules and libraries would severely limit the capabilities of the users' services, and possibly discourage them from using the platform.

Starting from the Dockerfile, the user should only control the files copied inside the container, the language of the service and the language modules/libraries needed for the service to function successfully. By letting the user submit Dockerfiles, there is the freedom to declare whichever available docker image (e.g. image containing penetration testing tools) paired with access to root privileges inside the container. The way to tackle this is to prepare and offer specific images according to the user's choice of language, selected from a dropdown inside the web application, as well as running the CMD command with a non-root user account. An example of a Dockerfile

as described could be:

```
1    FROM python:3.6 # User chose Python v.3.6.
2
3    RUN useradd -ms /bin/sh useraccount # create non-root user
4    WORKDIR /home/useraccount
5
6    ######### USER CONTROLLED FILES
7    ADD pType.py ./
8    ADD requirements.txt ./
9    ####
10
11   RUN pip install -r requirements.txt # install modules
12
13   USER useraccount # from here on non-root privs
14   RUN mkdir /home/useraccount/log # creation of log dir
15   CMD [ "python", "./pType.py" ] # User controls the run file
```

The use of such Dockerfiles solves some problems such as the lack of Root privileges and the use of trusted container images handpicked from the developers, but it does not solve issues such as the ability to reach other containers, the ability to use host sources without limit. For this reason we need to add some options in the python commands that performs the container build. This is the code that performs the run command on the container:

```
1 obj = self.client.containers.run(image=self.IMAGE_TAG, name=self
     .CONTAINER_NAME, detach=True, environment=env_dict, network=
     app.config.get('CONTAINER_NETWORK'), volumes={str(self.
     service.path_to_service_folder()): {'bind': '/log/','mode': '
     rw'}})
```

We can apply the following parameters into the run call to further apply security controls regarding the resource usage:

1. mem_limit : Controls the limit of memory the container can use. For example if we want to constrain the memory use at 1 GB of Ram we can apply mem_limit=1g.

2. cpu_quota : Indicates the CPU microseconds per cycle threshold the container can use. The default value of the cycle is 100 microseconds so using at most 50% of the CPU per second would mean a value of cpu_quota=50

The correct values for these parameters should be chosen upon testing and carefully assessing the platform's capabilities and the services' needs.

Moving to the networking issue, the Python SDK for Docker does not provide a straightforward solution like the –icc=false parameter in docker run CLI command. As we can see from the code, the command uses *network = app.config.get('CONTAINER_NETWORK')* which means

that all containers of the deployment can be reached from the untrusted container. The most suitable solution would be to create a separate network inside the *docker-compose.yml* file to connect only the essential containers like the brokers to the untrusted container and leave out all the other. Having the ability to reach containers such as the databases which are not exposed to the Internet might compromise their security posture. Finally, it is heavily recommended that services created by users undergo a thorough review from experienced security professionals before finally getting accepted to the platform. Setting controls, and even applying automated tools can be a great supplementary protection but the human factor and individual review is not replaceable in this situation.

5.2.7   Applying the Methodology to SYNAISTHISI

Using the methodology described at the fourth chapter, we were able to assess the security controls and gaps in a real Internet of Things platform with the results summarized at Figure 5.4. The requirements that were defined were used as an anchor in order to identify potential insecurities, and the propositions led to actual solutions that will reinforce the security posture of the platform and the protection of its users. Establishing a security baseline for a real product by using this methodology, validates its usability through a real world use-case and the positive impact it can provide to similar products.

Table 5.4: Chapter 5 Findings Summary

Summary	
	Robust authorization scheme
	User Enumeration avoidance
Existing Robust Security	JWT authentication scheme
Controls	XSS Protection (Angular)
	SQL Injection Protection (SQLAlchemy)
	Hashed Sensitive Passwords inside Database
	TLS in Web Portal, NodeRED
	MQTTS for encrypted MQTT
	Device Authentication Token creation/Authn
	Bruteforce rate limiting
Identified Security Gaps	Mime-Sniffing and Clickjacking headers
with Feature Implementation	Cache control for sensitive information
	API Rate Limiting
	Back-end code review and insecure function fix
	Database Internet Exposure
	Secure Client Processing Service creation (Propositions)
Identified Security Gaps	Bad Password Policy
without Feature Implementation	Lack of CSRF controls
	Application Error Disclosure

Chapter 6

Conclusion

This work delved into the security domain of IoT ecosystems, and specifically IoT platforms. We initially set the background of the discussion, explored similar research and real incidents in order motivate the work's impact.

By devising an asset taxonomy, we were able to generalize IoT ecosystems and decompose them into assets that appear in the majority of ecosystem implementations. This allowed us to further focus and assess each asset for its security posture, requirements and mitigation controls, composing a methodology that can be used by related professions in the IoT domain.

Finally, a real IoT platform named SYNAISTHISI was put into scope in order to apply the methodology and validate its applicability in real world situations. The methodology helped identify insecurities of the platform and apply controls to these gaps in order to provide a better security baseline for the product and its users.

This work holistically addressed a variety of sub domains of Cybersecurity, including cryptography, web application security, network security and even social engineering among others. The final aim of this work is to provide a framework for establishing that secure baseline for IoT platforms, supplementing in its own way the work and security awareness of System Administrators, Software Engineers, Database Administrators or even End Users, and contributing towards a more resilient and secure future for the Internet of Things.

6.1 Future Work

The Cybersecurity domain is a fast-paced one. Malicious actors continuously search for new insecure factors to exploit, while security professionals continuously strive to protect computer systems and technology products. With new attacks, exploits, as well as mitigations and security controls, an obvious future step could be the extension of this work for new security issues that are not yet addressed in this version. In the same fashion, more non-standard methods towards securing the IoT could be added, which leverage technologies such as Blockchain or Machine Learning. The two technologies can potentially provide great supplementary tools in the Cybersecurity domain, while their compatibility with the Internet of Things security is yet to be fully realized.

Bibliography

[1] Carolina Adaros Boye, Paul Kearney, and Mark Josephs. Cyber-risks in the industrial internet of things (iiot): Towards a method for continuous assessment. In Liqun Chen, Mark Manulis, and Steve Schneider, editors, *Information Security*, pages 502–519, Cham, 2018. Springer International Publishing.

[2] T. Ahanger. Defense scheme to protect iot from cyber attacks using ai principles. *Int. Journal of Computers Communications & Control*, 13:915–926, 11 2018.

[3] Charilaos Akasiadis, Vassilis Pitsilis, and Constantine D. Spyropoulos. A multi-protocol IoT platform based on open-source frameworks. *Sensors*, 19(19):4217, September 2019.

[4] A. A. Akinyelu and A. O. Adewumi. Classification of phishing email using random forest machine learning technique. *Journal of Applied Mathematics*, 2014:425731, Apr 2014.

[5] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose. Sok: Security evaluation of home-based iot deployments. In *2019 2019 IEEE Symposium on Security and Privacy (SP)*, Los Alamitos, CA, USA, may 2019. IEEE Computer Society.

[6] Jude Ambrose, Roshan Ragel, Darshana Jayasinghe, Tuo Li, and Sri Parameswaran. Side channel attacks in embedded systems: A tale of hostilities and deterrence. 2015:452–459, 04 2015.

[7] M. Ammar, G. Russello, and B. Crispo. Internet of things: A survey on the security of iot frameworks. *Journal of Information Security and Applications*, 38:8–27, 2018. cited By 144.

[8] Pelin Angin, Melih Burak Mert, Okan Mete, Azer Ramazanli, Kaan Sarica, and Bora Gungoren. A blockchain-based decentralized security architecture for iot. In Dimitrios Georgakopoulos and Liang-Jie Zhang, editors, *Internet of Things – ICIOT 2018*, pages 3–18, Cham, 2018. Springer International Publishing.

[9] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *USENIX Security Symposium*, 2017.

[10] B. Arunkumar and Kousalya Govardhanan. Analysis of aes-gcm cipher suites in tls. pages 102–111, 01 2018.

[11] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A lightweight hash. volume 26, pages 1–15, 08 2010.

[12] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. Blake2: Simpler, smaller, fast as md5. In Michael Jacobson, Michael Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, pages 119–135, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[13] Infocomm Media Development Authority. Guidelines: Internet of things (iot) cybersecurity guide. `https://www.imda.gov.sg/-/media/imda/files/regulation-licensing-and-consultations/consultations/open-for-public-comments/consultation-for-iot-cyber-security-guide/imda-iot-cyber-security-guide.pdf`, January 2019.

[14] Paul Bischoff. Which countries have the worst (and best) cybersecurity? `https://www.comparitech.com/blog/vpn-privacy/cybersecurity-by-country/`, March 2020.

[15] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. *OSDI*, 03 1999.

[16] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki. Network intrusion detection for iot security based on learning techniques. *IEEE Communications Surveys Tutorials*, 21(3):2671–2701, 2019.

[17] Abiy Biru Chebudie, Roberto Minerva, and Domenico Rotondi. *Towards a definition of the Internet of Things (IoT)*. PhD thesis, 05 2015.

[18] Mehiar Dabbagh and Ammar Rayes. *Internet of Things Security and Privacy*, pages 195–223. 10 2017.

[19] M. Dikmen and C. Burns. Trust in autonomous vehicles: The case of tesla autopilot and summon. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1093–1098, 2017.

[20] X. Du and X. Chang. Performance of ai algorithms for mining meaningful roles. pages 2070–2076, 07 2014.

[21] Elena Dubrova. Anti-tamper techniques. `https://people.kth.se/~msmith/is2500_pdf/Anti-Tamper%20Techniques_elena.pdf`, 2018.

[22] Dave Evans. How the next evolution of the internet is changing everything. `https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf`, 2011.

[23] B. Farahani, F. Firouzi, and K. Chakrabarty. *Healthcare IoT*, pages 515–545. 01 2020.

[24] ENISA: European Union Agency for Cybersecurity. Baseline security recommendations for iot: in the context of critical infrastructures. `https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot`, November 2017.

[25] ENISA: European Union Agency for Cybersecurity. Good practices for security of iot: Secure software development lifecycle. `https://www.enisa.europa.eu/publications/good-practices-for-security-of-iot-1`, November 2019.

[26] M. Frey, C. Gündoğan, P. Kietzmann, M. Lenders, H. Petersen, T. C. Schmidt, F. Juraschek, and M. Wählisch. Security for the industrial iot: The case for information-centric networking. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 424–429, 2019.

[27] C.H. Gebotys. *Security in Embedded Devices*. Embedded Systems. Springer US, 2009.

[28] T. Gopal, M. Meerolla, G. Jyostna, L. Eswari, and E. Magesh. Mitigating mirai malware spreading in iot environment. pages 2226–2230, 09 2018.

[29] S. Goswami, N. Hoque, Dhruba K Bhattacharyya, and Jugal Kalita. An unsupervised method for detection of xss attack. *International Journal of Network Security*, 19:761–775, 09 2017.

[30] Aaron Grattafiori. Understanding and hardening linux containers. `https://www.nccgroup.trust/us/our-research/understanding-and-hardening-linux-containers/`, June 2016.

[31] Jian Guo, Thomas Peyrin, and Axel Poschmann. The photon family of lightweight hash functions. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 222–239, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[32] J. Halderman, Seth Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph Calandrino, Ariel Feldman, Jacob Appelbaum, and Edward Felten. Lest we remember: Cold boot attacks on encryption keys. pages 45–60, 01 2008.

[33] A. Hameed and A. Alomary. Security issues in iot: A survey. pages 1–5, 09 2019.

[34] International. Global cybersecurity index. `https://www.comparitech.com/blog/vpn-privacy/cybersecurity-by-country/`, March 2020.

[35] Massila Kamalrudin, Asma Asdayana Ibrahim, and Safiah Sidek. A security requirements library for the development of internet of things (iot) applications. In Massila Kamalrudin, Sabrina Ahmad, and Naveed Ikram, editors, *Requirements Engineering for Internet of Things*, pages 87–96, Singapore, 2018. Springer Singapore.

[36] G. Kaur and M. Sohal. Iot survey: The phase changer in healthcare industry. *Int. Journal of Scientific Research in Network Security and Communication*, 6:34–39, 04 2018.

[37] Minhaj Ahmad Khan and Khaled Salah. Iot security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395 – 411, 2018.

[38] J. King and A. I. Awad. A distributed security mechanism for resource-constrained iot devices. 40:133–143, 01 2016.

[39] Ralph Langner. To kill a centrifuge a technical analysis of what stuxnet ' s creators tried to achieve. 2013.

[40] Y. Lee, J. Lim, Y. Jeon, and J. Kim. Technology trends of access control in iot and requirements analysis. pages 1031–1033, 10 2015.

[41] Shanhong Liu. Internet of things - statistics facts. `https://www.statista.com/topics/2637/internet-of-things/`, March 2020.

[42] Knud Lasse Lueth. Iot 2019 in review: The 10 most relevant iot developments of the year. `https://iot-analytics.com/iot-2019-in-review/`, January 2020.

[43] A. Makiou, Y. Begriche, and A. Serhrouchni. Improving web application firewalls to detect advanced sql injection attacks. *2014 10th Int. Conf. on Inf. Assurance and Security*, 11 2014.

[44] M. S. Mekala and V. Perumal. A survey: Smart agriculture iot with cloud computing. pages 1–7, 08 2017.

[45] Romanosky S. Mell P, Scarfone K. Cvss: a complete guide to the common vulnerability scoring system version 2.0. Technical report, FIRST: forum of incident response and security teams, June 2007.

[46] Saraju Mohanty. Everything you wanted to know about smart cities. *IEEE Consumer Electronics Magazine*, 5:60–70, 07 2016.

[47] S. Mukkamala, G. Janoski, and A. Sung. Intrusion detection using neural networks and support vector machines. volume 2, pages 1702 – 1707, 02 2002.

[48] Madalin Neagu and Liviu Miclea. Data scrambling in memories: A security measure. pages 1–6, 05 2014.

[49] 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*, 21(3):2702–2733, 2019.

[50] Q.-D. Ngo, H.-T. Nguyen, V.-H. Le, and D.-H. Nguyen. A survey of iot malware and detection methods based on static features. *ICT Express*, 2020.

[51] Mnemonic Norwegian Consumer Council. # watchout: Analysis of smartwatches for children. `https://fil.forbrukerradet.no/wp-content/uploads/2017/10/watchout-rapport-october-2017.pdf`, October 2017.

[52] S. O'Dea. Data volume of iot connected devices worldwide 2018 and 2025. `https://www.statista.com/statistics/1017863/worldwide-iot-connected-devices-data-size/`, February 2020.

[53] M. Patel. Qradar uba app adds machine learning and peer group analyses to detect anomalies in users' activities. `https://securityintelligence.com/qradar-uba-app-adds-machine-learning-and-peer-group-analyses-to-detect-anomalies-in-users-activities/`, March 2017.

[54] Geovandro C. C. F. Pereira, Renan C. A. Alves, Felipe L. da Silva, Roberto M. Azevedo, Bruno C. Albertini, and Cíntia B. Margi. Performance evaluation of cryptographic algorithms over IoT platforms and operating systems. *Security and Communication Networks*, 2017:1–16, 2017.

[55] Georgios Pierris, Dimosthenis Kothris, Evaggelos Spyrou, and Costas Spyropoulos. Synaisthisi: an enabling platform for the current internet of things ecosystem. In *PCI '15*, 2015.

[56] H. Qiu, X. Wang, and F. Xie. A survey on smart wearables in the application of fitness. pages 303–307, 11 2017.

[57] Somasundaram Ragupathy and Mythili Thirugnanam. *Review on Communication Security Issues in IoT Medical Devices*, page 189. 10 2017.

[58] S. Rawal, B. Rawal, A. Shaheen, and S. Malik. Phishing detection in e-mails using machine learning. *Int. Journal of Applied Information Systems*, 12:21–24, 10 2017.

[59] E. Rescorla. The transport layer security (tls) protocol version 1.2. `https://www.rfc-editor.org/rfc/rfc5246.txt`, August 2008.

[60] E. Rescorla. The transport layer security (tls) protocol version 1.3. `https://www.rfc-editor.org/rfc/rfc8446.txt`, August 2018.

[61] PwC Research. 2019 internet of things survey. `https://www.pwc.com/us/en/services/consulting/technology/emerging-technology/iot-pov.html`, July 2019.

[62] Rahul Rishi and Rajeev Saluja. Future of iot. `http://ficci.in/spdocument/23092/Future-of-IoT.pdf`.

[63] Syed Rizvi, RJ Orr, Austin Cox, Prithvee Ashokkumar, and Mohammad R. Rizvi. Identifying the attack surface for iot network. *Internet of Things*, 9:100162, 2020.

[64] A. Roukounaki, S. Efremidis, J. Soldatos, J. Neises, T. Walloschke, and N. Kefalakis. Scalable and configurable end-to-end collection and analysis of iot security data : Towards end-to-end security in iot systems. pages 1–6, 06 2019.

[65] J. Ruan, H. Jiang, C. Zhu, X. Hu, Y. Shi, T. Liu, W. Rao, and F Chan. Agriculture iot: Emerging trends, cooperation networks, and outlook. *IEEE Wireless Communications*, 26:56–63, 12 2019.

[66] Doyen Sahoo, Chenghao Liu, and Steven C. H. Hoi. Malicious url detection using machine learning: A survey. *ArXiv*, abs/1701.07179, 2017.

[67] Dimitrios Serpanos and Marilyn Wolf. Security and safety. In *Internet-of-Things (IoT) Systems*, pages 55–76. Springer International Publishing, November 2017.

[68] Dimitrios Serpanos and Marilyn Wolf. Security testing IoT systems. In *Internet-of-Things (IoT) Systems*, pages 77–89. Springer International Publishing, November 2017.

[69] A. Sharma and A. Thakral. Malicious url classification using machine learning algorithms and comparative analysis. In K. S. Raju, A. Govardhan, B. P. Rani, R. Sridevi, and M. R. Murty, editors, *Proc. of the 3rd Int. Conf. on Computational Intelligence and Informatics*, pages 791–799, Singapore, 2020. Springer Singapore.

[70] Jyoti Shetty. A state-of-art review of docker container security issues and solutions. *American International Journal of Research in Science, Technology, Engineering Mathematics*, 01 2017.

[71] Arijit Ukil, Jaydip Sen, and Sripad Koilakonda. Embedded security for internet of things. pages 1 – 6, 04 2011.

[72] Eugene Vasserman and Nicholas Hopper. Vampire attacks: Draining life from wireless ad hoc sensor networks. *Mobile Computing, IEEE Transactions on*, 12:318–332, 02 2013.

[73] Gang Wang, Zhijie Shi, Mark Nixon, and Song Han. ChainSplitter: Towards blockchain-based industrial IoT architecture for supporting hierarchical storage. In *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, July 2019.

[74] L. Xiao, X. Wan, X. Lu, Y. Zhang, and D. Wu. Iot security techniques based on machine learning. 01 2018.

[75] H. Xu, W. Yu, D. Griffith, and N. Golmie. A survey on industrial internet of things: A cyber-physical systems perspective. *IEEE Access*, 6:78238–78259, 2018.

[76] L. Xu, W. He, and S. Li. Internet of things in industries: A survey. *IEEE Trans. on Industrial Informatics*, 10:2233–2243, 11 2014.

[77] W. Yang, S. Wang, J. Hu, Z. Guanglou, and C. Valli. Security and accuracy of fingerprint-based biometrics: A review. *Symmetry*, 11:141, 01 2019.

[78] Arnold Yau, K.G. Paterson, and Chris Mitchell. Padding oracle attacks on cbc-mode encryption with secret and random ivs. volume 3557, pages 11–43, 07 2005.

[79] M. Zamani and M. Movahedi. Machine learning techniques for intrusion detection. *arXiv preprint arXiv:1312.2177*, 2013.

Abbreviations

ΠΘ	Πανεπιστήμιο Θεσσαλίας
Dr.	Doctor
e.g.	exempli gratia, meaning "for example"
Prof.	Professor
AES	Advanced Encryption Standard
AES-GCM	Galois-Counter mode
AEAD	Authenticated Encryption with Associated Data
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CIA	Confidentiality, Integrity, Availability
CLI	Command Line Interface
CPU	Central Processing Unit
CSRF/XSRF	Cross-Site Request Forgery
CVE	Common Vulnerability and Exposure
CVSS	Common Vulnerability Scoring System
C&C	Command and Control
DOS	Denial of Service
DDOS	Distributed Denial of Service
DNS	Domain Name System
ECC	Elliptic Curve Cryptography
EOL	End of Life
FTP	File Transfer Protocol
GPS	Global Positioning System
GDPR	General Data Protection Regulation
HTTP(S)	HyperText Transfer Protocol (Secure)
IDS	Intrusion Detection System
IoT	Internet of Things
IIT	Institute of Informatics and Telecommunications
IPv4/6	Internet Protocol version 4/6
IPS	Intrusion Prevention System
JWT	JSON Web Token
MQTT(S)	Message Queuing Telemetry Transport (Secure)

NDA	Non Disclosure Agreement
OS	Operating System
OTP	One Time Programmable
PLC	Programmable Logic Controller
RAM	Random Access Memory
REST	Representational State Transfer
RFC	Request for Comments
SDK	Software Development Kit
SSL	Secure Sockets Layer
SCADA	Supervisory Control and Data Acquisition
SCP	Secure Copy Protocol
SDLC	Software Development Life Cycle
SMS	Short Message System
SQL	Structured Query Language
SSH	Secure Shell
SoC	System on Chip
TEE	Trusted Execution Environment
TLS	Transport Layer Security
UI	User Interface
UTH	University of Thessaly
USB	Universal Serial Bus
VM	Virtual Machine
WAF	Web Application Firewall
XML	eXtensible Markup Language
XSS	Cross-Site Scripting