Apparatus: A design and analysis security framework for IoT systems

University of Brighton

Orestis Mavropoulos
Department of Computing, Engineering and Mathematics
University of Brighton

Supervised by:
Prof. Haralambos Mouratidis
Dr. Andrew Fish
Dr. Emmanouil Panaousis

A thesis submitted in partial fulfilment of the requirements of the University of Brighton for the degree of Doctor of Philosophy

College of Life, Health and Physical Sciences

October 2019
Declaration

I declare that the research contained in this thesis, unless otherwise formally indicated within the text, is the original work of the author. The thesis has not been previously submitted to this or any other university for a degree, and does not incorporate any material already submitted for a degree.

Orestis Mavropoulos
October 2019
Acknowledgements

I would like to express my gratitude to Prof. Haralambos Mouratidis, Dr. Andrew Fish, and Dr. Emmanouil Panaousis, my research supervisors, for their guidance, encouragement and useful critiques throughout the development of this research project. I would also like to thank Dr. Christos Kalloniatis, and Dr. Michalis Pavlidis, for their collaboration and support. I would like to extend my thanks to my colleagues, Nikos, Shaun, Daniel, Duaa and Myrsini for their support and inspiration. Finally, I wish to thank my family and friends for their unconditional support and encouragement in this endeavor.
Abstract

Internet of Things (IoT) systems are ubiquitous, highly complex and dynamic event-based systems. These characteristics make their security analysis challenging. One of the most significant concerns about IoT is that it is not secure. Prominent attacks, such as WannaCry and the Mirai botnets, have only stoked these fears and raised questions about the security of IoT. This thesis aims to develop a framework to facilitate design and security analysis in IoT systems. The proposed framework is composed of the following components: (1) a modeling language to represent IoT systems; (2) a modeling methodology to create models; (3) processes to assess the security of the models and (4) propose countermeasures to increase the security of the models. The modeling language provides components to create IoT system models that capture the information needed by a security engineer to design and perform security analysis on an IoT system. The modeling methodology provides instructions as well as restrictions on how models are created using the modeling language. It provides a structured approach to transition models between the engineering phases. Then, automated and semi-automated processes are used to identify existing vulnerabilities and configurations that increase the attack surface in the models. Finally, a report of the countermeasures is generated based on the attributes of the models to mitigate the identified threats and vulnerabilities. To evaluate the framework, it was applied to case studies. Each case study assessed specific aspects of the framework. One case study was performed to assess the processes of eliciting security requirements from the existing hardware architecture of a system. The feedback from the case study was used to refine the modeling language. A case study was performed to evaluate the automated and semi-automated analysis processes that are part of the framework. The processes were measured regarding resources, error rate, and additional information when compared to the same task undertaken by a human engineer. This case study was performed on a real-life infrastructure of a security organization. The organization’s infrastructure was modeled and analyzed using the framework to measure the attack surface of their system and improve its security mechanisms with the use of the analysis processes of the framework. Another case study was performed to model the infrastructure of a
smart city. This was made to assess the scalability of the framework when designing and analyzing large-scale systems.
# Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of figures</td>
<td>xiii</td>
</tr>
<tr>
<td>List of tables</td>
<td>xv</td>
</tr>
<tr>
<td>Nomenclature</td>
<td>xv</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Related Scientific Fields ............................ 3
   1.1.1 Internet of Things .............................. 3
   1.1.2 Security by design ............................. 5
   1.1.3 Summary of findings ............................. 6
1.2 Research Questions and Objectives .................. 6
1.3 Research Methodology ............................... 8
1.4 Document Structure ................................ 10
1.5 Main Contributions ................................ 11
1.6 Publications ..................................... 12

## 2 Literature Review

2.1 Review Protocol .................................... 15
2.2 Security in the IoT ................................. 17
   2.2.1 IoT modeling and visualization .................. 22
2.3 Security Requirements Engineering ................ 25
2.4 Research Gaps and Challenges ...................... 27
2.5 Evaluation of Security Frameworks ................. 31
2.6 Evaluation of the Terminology of Security Applications .................. 34
   2.6.1 Summary ...................................... 36

## 3 Apparatus Framework

3.1 Modeling Language of Apparatus .................. 40
Table of contents

3.1.1 Design Phase Metamodel ........................................... 42
3.1.2 Implementation Phase Metamodel ................................. 45
3.1.3 System State in the Apparatus Framework ....................... 50
3.1.4 Notation .................................................................. 52
3.2 Modeling Procedure ..................................................... 54
  3.2.1 Design phase modeling procedure ................................. 54
  3.2.2 Implementation phase modeling procedure ...................... 55
  3.2.3 Transition rules between the different engineering phases ... 58
3.3 Model based Analysis .................................................... 60
  3.3.1 Implementation Phase Analysis ................................. 61
3.4 Discussion on the modeling language .................................. 77
3.5 ASTo: Software Application of Apparatus ......................... 80
  3.5.1 Requirements of the software application ...................... 80
  3.5.2 ASTo’s architecture .................................................. 82
3.6 Proof of Concept Applications ....................................... 90
  3.6.1 First PoC - Initial approach to IoT security modeling ....... 90
  3.6.2 Second PoC - Implementation phase analysis ................. 94
  3.6.3 Third PoC - Security analysis of a smart city ............... 96
3.7 Summary ................................................................. 99

4 Evaluation ................................................................. 101
  4.1 Case Study ................................................................ 101
    4.1.1 Case Study Process ............................................... 102
    4.1.2 Case Study Design ............................................... 102
    4.1.3 Case Study Privacy and Security considerations .............. 104
    4.1.4 Framework Application ......................................... 105
    4.1.5 Case Study Results .............................................. 127
    4.1.6 Threats to the validity of the Case Study .................... 133
  4.2 Lessons Learned ....................................................... 134
  4.3 Usage of ASTo ........................................................ 136
    4.3.1 ASTo’s Open source community contribution .............. 136
    4.3.2 Feature comparison of ASTo and relevant Security Tools ... 140
  4.4 Summary ............................................................... 142

5 Conclusion ............................................................... 143
  5.1 Research Outputs ..................................................... 144
  5.2 Future Research Directions .......................................... 146
Table of contents

References 149

Appendix A Terminology 159
List of figures

1.1 The Information Systems Research Framework[46] ........................................ 9

3.1 APPARATUS Framework components ......................................................... 40
3.2 Design phase metamodel ........................................................................... 46
3.3 Implementation phase metamodel ............................................................... 50
3.4 Example of an APPARATUS State diagram ................................................. 52
3.5 Design phase flow chart ............................................................................ 56
3.6 Implementation phase flow chart ................................................................. 59
3.7 ASTo home screen ..................................................................................... 83
3.8 ASTo design phase GUI ............................................................................. 84
3.9 ASTo Implementation phase GUI ................................................................. 85
3.10 ASTo software architecture ...................................................................... 86
3.11 First metamodel of Apparatus .................................................................. 90
3.12 JSON representation of the PoC ................................................................. 92
3.13 PoC 2 - Implementation phase metamodel ................................................ 94
3.14 PoC 2 - Smart home system ..................................................................... 95
3.15 PoC 2 - Security analysis of the camera component .................................... 96
3.16 Design phase network module of the transport system .............................. 97
3.17 Design phase model of the transport system ............................................. 98
3.18 Implementation phase model of the transport system ............................... 99

4.1 System model derived from network capture files (concepts) .................... 108
4.2 System model derived from network capture files (concept description) .... 109
4.3 System model with the network components (concept description) .......... 110
4.4 Group 1 vulnerability analysis .................................................................... 117
4.5 Group 2 system information ...................................................................... 118
4.6 Group 2 vulnerability identification ............................................................. 120
4.7 Group 3 system information ...................................................................... 121
List of figures

4.8 Group 3 vulnerability identification ................................. 123
4.9 Group 4 system information ........................................... 124
4.10 Group 4 vulnerability identification ............................... 125
4.11 Security insights of the model ....................................... 126
4.12 ASTo traffic statistics from 28/06 - 11/07 ......................... 137
4.13 ASTo referrer webpages from 28/06 - 11/07 ....................... 138
4.14 ASTo traffic statistics from 14/07 - 27/07 ......................... 138
4.15 ASTo referrer webpages from 14/07 - 27/07 ....................... 139
List of tables

2.1 Number of records by keyword search ........................................ 16
2.2 Evaluation of Frameworks .............................................................. 33
2.3 Terminology in tools and frameworks in network security and administra-
tion ........................................................................................................ 35

3.1 Notation Classes ................................................................................ 53
3.2 Threat - Constraint pairs ................................................................. 61
3.3 Security insights based on the Device concept ................................. 74
3.4 Security insights based on the Application concept ....................... 74
3.5 Security insights based on the Connection concept ...................... 75
3.6 Security insights based on the Micronet concept ......................... 75
3.7 Vulnerability Identification Attributes ........................................... 76
3.8 Security requirements elicited from PoC properties ....................... 93

4.1 Windows 7 Vulnerabilities list ......................................................... 122
4.2 Function intention ............................................................................. 128
4.3 Functions’ time resources comparison ............................................ 129
4.4 Comparison of online and local hosted vulnerability databases .... 130
4.5 Feature comparison of ASTo and relevant Security Tools ............. 141
Chapter 1

Introduction

The upcoming age of the Internet of Things (IoT) blurs the line between our physical and digital lives. Attacks targeting the digital space will put our physical security at risk. Traditionally, the attack vectors to our infrastructure required physical tampering. Before the IoT, digital access to physical infrastructures has been limited to a select number of applications. The majority of those applications included some form of automation, such as using a mobile application to control a home security system. The number of digital applications that interact with the physical realm is about to change with the disruption that will be caused by a future with billions of “things” connected to the Internet. The vision of IoT is not new but has existed since the early conception of computer networks. Weiser in 1991 [126] provides one of the most accurate yet simple visions of IoT by stating that the most profound technologies merge with the environment. He states that technology will be so evident that we will start perceiving it as a natural part of life. Our communication with technology will be similar to the interaction we are used to having from our environment and other people. IoT along with Cloud computing promises to make that statement into a reality.

IoT has drawn interest from both industry and academia. IoT incorporates radiofrequency identification (RFID), sensors, smart devices, the Internet, smart grids, cloud computing, vehicle networks, and many more information carriers. Typical “things” may be end users, data centers, processing units, smartphones, tablets, Bluetooth, ZigBee, IrDA, UWB, cellular networks, Wi-Fi networks, NFC, RFID tags, household machinery, wrist watches, and vehicles among some examples. The cheap availability of nanodevices, smartphones, 5G connectivity, microcomputers and distributed networks, along with the advancements in open source software and programming languages has attracted the interest of a variety of makers and tinkerers.
Concerning security, the interconnectivity of such a vast assortment of devices can expand the attack surface of systems. Interconnecting many “things”, also means the possibility of interconnecting many different threats and attacks. IoT will blur the line between the physical environment and the cyber environment. Attacks that originated from the cyber realm did not have a direct impact on the physical realm before IoT. For example, attackers can gain control of a home security system, so that they can physically access it without triggering an alarm. Another example is the compromise of health embedded electronics. Health-related electronics monitor and regulate the health of a patient. When such a device is compromised, it can lead to fatal health implications to the affected patient.

In this work, a security framework to design and analyze IoT systems using a model-driven approach is introduced. To design an IoT system, a modeling language is developed. The modeling language contains the elements with which a model of an IoT system can be described. Based on the information of a model, the security posture of the system can be analyzed. The analysis processes are formalized by a set of algorithms. The algorithms have been implemented in a software tool that facilitates the application of the framework. The framework borrows concepts from Computer Network Security and Security Requirements Engineering research areas. From Computer Network Security and Security Requirements Engineering the necessary concepts to define an IoT system are identified. The identified concepts are used to develop a modeling language to create modeling instances of IoT systems in the form of graphs. Graphs are used for the front-end representation of the models as well as graph-based functions. An example of a graph-based function is the identification of the malicious point of entry to visualize the threat propagation in the system.

The present chapter aims to provide the reader with background information about the main research areas of the thesis. The main research areas are the IoT and Security Requirements Engineering (SRE). The background information of IoT summarizes how the definition of IoT evolved and how different organizations envision an IoT world. Information about SRE outline the common approaches of system analysis using SRE and how they translate to an IoT system. Next, the aims, objectives and research questions which this project tackles are presented. Then, the research method that was used is explained. Finally, the publications where parts of this work have been featured are listed.
1.1 Related Scientific Fields

1.1.1 Internet of Things

Since the term 'Internet of Things' (IoT) was popularized, similar terms have made their appearance, either to define more specialized domains of IoT or express something different altogether. One such term is the “Web of Things” (WoT). The words web and Internet are often used as synonyms. Some works define “Internet of Things” and “Web of Things” as different domains [11, 18, 40], while others interchangeably treat the terms [129]. The creators of WoT define it by stating the difference between IoT and WoT. Their difference lies in the network protocols that IoT and WoT will use. IoT will use the first six OSI layers (physical, data link, network, transport, session, presentation), while WoT will only use the seventh (application) layer [130]. They argue that the application layer is much easier for a developer to work with and it exposes Things to greater functionality. Another term that is often used in the same breath as IoT is "Internet of Everything," which was first presented by Cisco. Cisco stated that the Internet of Things would be composed of billions of devices, while the Internet of Everything will be the next step that will connect "everything" together. Cisco defined IoE by first defining IoT and how it differs from IoE [30]. What both IoE and WoT have in common is that they have been defined using the IoT as a comparison, without using an established IoT reference model. The existence of similar terms used for IoT only begs the question of what IoT is in the first place?

The research community is working on standardizing the many different aspects of IoT. The Global Standards Initiative on the Internet of Things (IoT-GSI) was one institute aiming to develop technical standards for IoT systems. Since July 2015, IoT-GSI concluded its activities and established another group named Study Group 20 (SG20) with similar aims. IoT-GSI developed the standard for IoT in July 2016 in Recommendation ITU-T Y.2060. In the ITU-Y Y.2060 document, IoT is defined as “A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies.”, along with two notes for clarification [51]:

1. NOTE 1 - Through the exploitation of identification, data capture, processing and communication capabilities, the IoT makes full use of things to offer services to all kinds of applications, whilst ensuring that security and privacy requirements are fulfilled.
2. NOTE 2 - From a broader perspective, the IoT can be perceived as a vision with technological and societal implications.

The perception of what is considered IoT has changed over time. The first publications relating to IoT were focused either on its usage of secular networks or RFID chipsets. The wide availability along with low cost of RFID were significant factors in the adoption of IoT. Cheap electronics meant that they could be included in more devices, even ones with short life expectancy, to improve their functionality. The industrial sector embraced RFID tags and incorporated them into their workflow. RFID tags contain an antenna and a memory chip that stores data. RFID tags saw wide usage in retail to facilitate inventory and package tracking. Health sector uses RFID tags to monitor the health of patients and their medication intake. RFID tags are also incorporated in smart passwords, credit cards, as well as identification badges to access secure areas. The trend of adding RFID tags and readers in “things” gave inspiration to the term “Internet of Things”.

Inventors started toying with the idea of having an interactive environment. Cheap, powerful microcomputers and microcontrollers made their appearance. Notable ones were the Raspberry Pi and the Arduino. These devices are purchased en-mass by makers, hobbyists and industry alike for a wide range of applications. The prominent features of those devices were the networking capabilities, ability to execute high-level programming languages, programmable input and outputs, low cost and low energy requirements. The factors mentioned above significantly reduced the barrier to an interconnected world [79]. While embedded devices saw more extensive usage, their hardware architecture could not make full use of the computer networks protocols that were widely used at that time [19]. For example, the HTTP protocol was designed with the architecture commonly found in computers. Embedded devices required more resources and boilerplate to be able to execute it. The different architecture of embedded devices and the need for the embedded devices to have network capabilities made clear that new networking protocols need to be designed [6]. The Near Field Communication [31] (NFC) and the Message Queuing Telemetry Transport [47] (MQTT) protocols are examples of network protocols designed to be used by embedded devices.

As Cloud computing became more affordable and Web technologies became more prominent, the vision of IoT started changing. Researchers began discussing the possibility of a more interconnected world with a more extensive variety of connected devices. Cities, where traffic infrastructure, administration services, and health applications are interconnected to provide additional services to their inhabitants, is one such vision.
A key area of research of IoT is industrial applications as is being evident in the survey by Xu [131]. Industrial applications in IoT are being developed for some years, while taking advantage of certain key technologies, such as RFID, NFC, smartphones, location-based services, and social networks. Internet of Things may seem an umbrella term that seeks to have any notion of a network under it, but it offers unique opportunities and applications. For example, today we make a distinct separation of what is known as the Internet and a private home network. In reality, the difference concerning the network topology is insignificant. Both topologies represent networks, populated by devices that communicate through the use of protocols. Their only difference lays in their scope. Internet of Things is the next phase of the Internet, where everything will be interconnected. Its scope will be far greater than today’s Internet. It is argued that IoT is a pervasive deployment of smart objects, instead of a new technology, that can be achieved by making objects smart using RFID tags. The same approach can be seen with other technologies such as Bluetooth LE, where objects can be augmented with Bluetooth Tags as reported by Gomez [38], or with RESTful applications based on Web technologies to move from an IoT architecture to a WoT architecture [40].

1.1.2 Security by design

Secure by design, in software engineering, means that the software has been designed from the foundation to be secure. Malicious practices are taken for granted and care is taken to minimize impact in anticipation of security vulnerabilities, when a security vulnerability is discovered or on invalid user input [27]. A similar approach is used in Security Requirements engineering (SRE). SRE is derived from Requirements Engineering (RE), which refers to the process of defining, documenting and maintaining requirements in the engineering design process [20, 84]. An essential part of that process is the elicitation and analysis of the security requirements of the software system. Security Requirements Engineering advocates the systematic process for identifying, analyzing and specifying the security requirements for a system under development [43]. Software systems that are designed with security during the early system development stages are more robust and more cost-effective during their life cycle than systems that are not designed with security in mind but rather implement security measures when a breach has occurred [36].

During security analysis, it is essential to consider the users of the application and how they interact with the application’s various components. In requirements engineering the term “socio-technical” systems is used to refer to the social interactions between the system’s users or autonomous participants and the software application [86].
The socio-technical analysis is not limited to the technical requirements of the system, but also includes the social entities that will interact with the system, along with the entity’s goals and constraints in the given system. The aim of the security analysis of the social interactions between the entities and the system under development is to mitigate security issues by the system’s design. The literature suggests the use of goal-oriented requirement engineering approaches to perform socio-technical requirements analysis [86].

The primary use of SRE is the security analysis of software systems, their interactions with the human component and the goals of the system’s participants.

1.1.3 Summary of findings

While security considerations are not new in the context of information technology, the attributes of many IoT implementations present new and unique security challenges. Addressing these challenges and ensuring security in IoT products and services must be a fundamental priority. Users need to trust that IoT devices and related data services are secure from vulnerabilities, especially as this technology becomes more pervasive and integrated into our daily lives. Insecure IoT devices and services can serve as potential entry points for malicious actors.

The interconnected nature of IoT devices means that every device that is connected to the public-facing Internet affects the security and resilience of the Internet in whole. This challenge is amplified by other considerations like the mass-scale deployment of IoT devices, the ability of some devices of automatic provisioning and connection to other devices, and the likelihood of deployment in hostile environments.

1.2 Research Questions and Objectives

This project aims to develop a security framework to facilitate the design and analysis of IoT systems. The presented research questions are derived from the identified gaps in the literature which are presented in Sec. 2. The objectives of the project are addressing these questions with a novel and a structured approach.

Research Questions

- RQ1: What are the fundamental characteristics and requirements of an IoT system regarding security?
1.2 Research Questions and Objectives

- RQ2: What are the necessary components of a modeling language to elicit the security requirements of an IoT system?

- RQ3: How can security requirements of an IoT system be elicited from its system (hardware/software/network) architecture?

- RQ4: What types of security analysis need to be performed in an IoT system to offer baseline security?

Research Objectives

- Obj1: To answer RQ1, an extensive and in-depth literature review was pursued. The literature review was made to identify the unique features of IoT and the unique challenges concerning IoT security. Sec. 2.4 describes the findings regarding research gaps and challenges for IoT system security;

- Obj2: To address RQ2, a modeling language is developed that is able to express IoT systems based on unique characteristics of IoT that were identified in RQ1;

- Obj3: To address RQ3, a set of processes is developed to be used with the modeling language. Within the context of such processes, the hardware characteristics of a system are considered a stakeholder, regarding how certain security requirements can be elicited. In comparison, in traditional requirements engineering the security requirements are extracted from the stakeholders of the system. The requirements elicitation process usually takes the form of a discussion or an interview, between the requirements engineer and the stakeholders. This process takes place during the design of the system before any resources have been committed to its implementation. The identified security requirements aid in defining the final configuration of the implemented system. In certain use cases, IoT systems are expected to be designed on top of existing traditional systems. In such cases, resources have been spent on the existing system. The existing architecture could impose specific requirements on the system design. This is common in legacy systems, where the costs of upgrading the system are not feasible. The IoT extension to the system must take into account the existing system requirements before its design;

- Obj4: To address RQ4, the analysis processes that can be conducted on the models will be formalized. After the formalization is finished, a case study will be performed to evaluate the benefits of using the modeling language along with
analysis procedures. The modeling language should be used to facilitate certain types of security analysis. Depending on the phase of the engineering process, different types of security analysis can be performed. For example, during the engineering design phase, threat modeling can be performed but not vulnerability identification. Vulnerability identification relays on information that only exists in the implementation phase.

1.3 Research Methodology

There is an established research methodology in the field of information systems, introduced by Simon in “The Sciences of the Artificial”, published in 1969. The Design Science Research method (DSR) is a highly popular research method. It offers a clear step by step approach that can be followed in any information systems research endeavor. The paradigm of the method is to identify specific problems and provide innovative and valuable solutions using four artifacts: Constructs, models, methods, and implementations [71]. Once a problem is identified, an artifact is developed to provide a solution. The steps provided by the design science research method are the following [72]:

1. Identification and description of relevant organizational IT problem.

2. Demonstration of no existing solutions for the identified problem in the knowledge base of the area.


4. Evaluation of the utility offered by the created artifact.

5. Articulation of added value provided by the artifact to the practice and knowledge base.

6. Explanation of the practical implementations of the developed solution.

In the context of the research project, the developed artifact is a framework, aiming at the development of secure IoT systems. Steps 1 and 2, as defined above, were performed by reviewing the literature on the area of security in IoT and SRE, as presented in Chapter 2 of this document.

The development of the framework, covering Step 3 of the research framework, was the main activity of this research project. Certain discrete building blocks are
required to create a framework able to facilitate the development of secure IoT systems, as discussed in Chapter 3. Once such building blocks are solidified, and a working framework prototype has been tested as a proof-of-concept, relevant computer-aided software engineering (CASE) tools to support the framework’s application, were identified, extended or developed from scratch. The requirements of the software tool are presented in Section 3.5.

As defined by Step 4 of the research framework, an evaluation of the framework’s utility, efficacy and quality must be rigorously demonstrated for feedback to be provided back to the development phase, as part of the iterative “build and evaluate” loop [46]. There are several methods available for the evaluation of designed artifacts, with some examples mentioned in Fig. 1.1, out of which case studies are most commonly used in the field of information systems research [83].

![Fig. 1.1 The Information Systems Research Framework][46]

The evaluation of this research project is presented in Chapter 4 and follows an iterative approach. First, each of the developed components of the proposed method was applied to real-life examples as proof of concept. Several of the publications originating from this research project (see Section 1.5) include applications of a single or a combination of components, to small-scale real-life examples, for their functionality
to be assessed qualitatively and appropriate alterations to be made during the next iteration of their development. Additionally, components that have been developed from scratch were evaluated through workshop-based experiments to assess their comprehensibility and ease-of-use.

Later when a functional prototype of the framework had been developed, a large-scale case study was performed for its evaluation. For this case study, an organization active in the development of security-critical systems was contacted to model and assess the attack surface of its infrastructure. The steps required for the design and execution of this case study followed the guidelines introduced by [97]. During its initial steps, quantitative metrics were identified to obtain a good indication of the effectiveness of the developed framework. Such metrics evaluate the efficacy and error-reduction when automating an analysis process of the framework.

Moreover, qualitative evaluation approaches were explored during the case study design. More specifically, interviews with the participating stakeholders of the organization selected for the case study provided insights regarding the perceived applicability and effectiveness of the framework. Finally, another way to evaluate the contribution of the developed framework was its ability to perform tasks that were previously not feasible by similar approaches. Such aspects were identified through the literature review (see Chapter 2) and aligned with the framework’s contribution in the concluding section of this work.

The outcome of the evaluation formed the basis upon which the conclusions were drawn, regarding the quality and effectiveness of our designed artifact. This provided the primary input for completing Steps 5 and 6 of the research framework, where the added value and practical implication of the framework were identified, as discussed in Chapter 5.

1.4 Document Structure

The rest of the document is structured as follows; Chapter 2 presents a literature review which overviews related works in the area of IoT security to identify overall research gaps and limitations. Chapter 3 presents the framework developed as part of this research by first providing a general overview of its components and then presenting the theoretical background and application of each component. Chapter 4 shows the different evaluation-related activities undertaken as part of this research project. Chapter 5 provides a short discussion to recap the main contributions of this work and presents an overview of the main activities to be performed as part of our future
work. Finally, the Appendix has a list of terminology definitions used throughout the document.

1.5 Main Contributions

The framework presented in this work contributes towards the security of IoT research domain. More specifically, the contributions were made in IoT system modeling, IoT security modeling, software-aided security analysis, and decision support. The major contributions of the framework can be summarized as follows:

- A list of the fundamental characteristics and requirements of an IoT system (relates to RQ1)

- A modeling language for developing models of IoT systems to facilitate security analysis and decision support. The modeling language is described by a 1) syntax; 2) semantics, and 3) notation (relates to RQ2)

- A set of rules on how to transition from an IoT model from one engineering phase to the other. The rules can be applied to transition a design phase model to an implementation phase model, and vice versa. The rules have been developed into an algorithm that can be applied automatically. (relates to RQ4)

- A set of analysis algorithms that perform the following:
  - An algorithm to validate a model in accordance with the rules of the metamodel of the language (relates to RQ2)
  - An algorithm to perform semantic validation on the proposed threats and constraints of a model using Type checking. Threats and constraints are classified using types. Each Threat type can be mitigated by at least one specific Constraint type. For example, a type of Spoofing Threat must be mitigated by a type of Authentication Constraint. (relates to RQ4)
  - An algorithm to generation implementation phase models using information encoded on network capture files (relates to RQ3)
  - An algorithm to propose security insights on implementation phase models using information from the enumerated attributes of the model’s components. Security insights are used to facilitate the decision-making process by highlighting the attributes of the model that can result in increasing the attack surface of the system (relates to RQ3 and RQ4)
– An algorithm to identify vulnerabilities of devices and applications of an implementation phase model (relates to RQ3 and RQ4)

– An algorithm for identifying patterns on models. Models can be queried using a set of keywords (relates to RQ4)

• A software tool that fully supports the use of the framework and automates its processes and algorithms. The tool support additional features such as customization of the representation of models. Moreover, the tool incorporates the documentation of the framework in addition to a tutorial in the use of the framework. (relates to RQ4)

1.6 Publications

The following publication has been made during the project:


The author was the main contributor to the publications presented while the supervisory team provided valuable feedback and refinements on the outcome. The first publication, Apparatus: Reasoning About Security Requirements in the Internet of Things, was a position publication. In the paper, the need for a framework to design secure IoT systems was identified. The requirements of the described and the building
blocks of the framework were proposed. The second publication, *ASTo: A tool for security analysis of IoT systems*, presented the functionalities of the framework’s supporting software application and the benefits it offers regarding analysis. Additionally, the next iteration of the framework’s modeling language was shown. The third publication, *A Conceptual Model to Support Security Analysis in the Internet of Things*, presented the third iteration of the modeling language and the modeling and analysis procedures supported by the framework. The fourth publication, *Apparatus: A Framework for Security Analysis in Internet of Things Systems*, presented the fourth iteration of the modeling language, the notation of the language as well as processes to address the scalability of large-scale systems using the modeling language, its notation, and its tooling.
Chapter 2

Literature Review

In this work, a systematic literature review is performed according to the guidelines provided by [72]. The objective of this review is to synthesize the information collected by the literature in the area of security in IoT and security requirements engineering (SRE) and identify current challenges, research gaps and future directions for researchers. According to the identified guidelines, the first phase of the review consists of the planning, which includes the identification of the review protocol to be followed. Next, a review is conducted by searching, filtering and selecting the relevant works and finally, the collected data is synthesized, and the report is created.

2.1 Review Protocol

To identify relevant works for this review, certain selection criteria were established. Firstly, for an article to be considered relevant, it needed to be focused on both the overall area of security and IoT. Therefore, works focusing on IoT or information security, in general, were excluded since the structure of the keywords used made sure only works in the intersection of both areas appeared in the search results. Since the overall focus of this research is on the design of secure IoT systems, modeling is an essential aspect to be considered. The identified works had to be under the umbrella of model-driven engineering and involve “model-driven” approaches to IoT system design to be included in the review. Approaches such as algebraic modeling or other mathematical approaches were excluded from the review.

The search engine of Web of Science 1 was utilized for the identification of relevant literature. This selection was mainly due to the wide variety of relevant journals

1https://www.isiknowledge.com
indexed at Web of Science and its ability to select different filtering parameters and structure the keywords with logical operators and wildcard characters (e.g., AND, OR, *, ”). The keywords used for our searches were “internet of things security”, “internet of everything security”, “web of things security”, “security requirements engineering”, “security” requirements engineering”, “model-driven” AND “IoT Security”, “IoT*” AND “security requirements engineering”. The only exclusion criterion applied to the search results was their language, which was limited to English. No limitation on publication dates was enforced and as a result the identified literature’s spans from 1992 to 2018. The initial number of records recovered by each of the keywords used are included in Tab. 2.1, in total 2095 records were identified.

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Nu. of records</th>
</tr>
</thead>
<tbody>
<tr>
<td>“internet of things security”</td>
<td>1045</td>
</tr>
<tr>
<td>“internet of everything security”</td>
<td>129</td>
</tr>
<tr>
<td>“web of things security”</td>
<td>457</td>
</tr>
<tr>
<td>“security” requirements engineering</td>
<td>342</td>
</tr>
<tr>
<td>“model-driven” AND “IoT Security”</td>
<td>90</td>
</tr>
<tr>
<td>“IoT*” AND “security requirements engineering”</td>
<td>32</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2095</strong></td>
</tr>
</tbody>
</table>

The first stage of the selection of relevant works, according to the previously discussed criteria, was performed by checking the title and abstract of each of the identified works. During this stage for each search result, the title and abstract were read. If they were considered as relevant to the literature review they were saved for further evaluation. Some works appeared in multiple queries. The duplicated entries were removed. As a result of this process, at the end of the first stage, 122 articles had been selected for further reading.

The second stage of the selection process included reading the whole body of the selected resources and determining which should be included in the final review. For each resource, keywords were assigned, that later assisted in the categorization of the selected literature in groups. As a result of the second round of the selection process 63 articles had been selected to be included in the final review.
2.2 Security in the IoT

An IoT system may encompass any device regardless of its hardware or software, along with any communication protocol, Internet browser or operating system. IoT systems can be composed of networks of as little as three devices to networks that can contain millions of devices. IoT networks, similar to computer networks, are as secure as their weakest link. The bigger the network, the more opportunities an attacker has to compromise it. Those potential openings are named attack surface, a standardized way of evaluating security vulnerabilities in Software engineering [70].

An IoT network will inherit all the vulnerabilities of the existing technologies. For example, a vulnerability found in a wireless network could also be used to gain an entry point to an IoT system that uses the same wireless technology. While analyzing the security of IoT in [53], it is argued, that since IoT will be built upon the existing Internet, they will share the same security issues. To better reason about security, he proposes three layers of abstraction, the perception layer, the transportation layer, and the application layer. One of the first attempts to standardize security in IoT proposes to use the Constrained Application Protocol (CoAP) along with existing internet protocols and the Datagram Transport Layer Security (DTLS) [57].

A survey conducted in [15] for the IoT identifies certain security and privacy issues. A major point of concern for IoT networks is the limitation of embedded devices to run the latest communication protocols. IoT due to its dependence on wireless communications is susceptible to an array of different kinds of Denial of Service attacks and Data theft. Another survey focusing on the Industrial IoT [99] emphasizes the wide attack surface any IoT network would have. Security flaws that focus on the physical aspect of the network are highly dangerous, since they provide an entry point to the network, while Man-in-the-Middle attacks can be used to steal sensitive corporate information. As mentioned earlier, IoT will pave the way for IPv6. A large number of IoT devices will be small sensors and actuators designed with minimal cost and overhead. That will result in poor performance when implementing the latest security protocols that are designed with today’s powerful computers in mind [88]. Another security mechanism has been proposed called BlinkToSCoAp, that explicitly targets small devices. It promises to reduce the overhead needed to run more complex protocols and also boost security by giving the ability to run complex encryption algorithms with little battery drainage. A similar approach to combat the disadvantages of IP was to introduce “chirps” [32], small bits of information that would allow small devices secure communications without the burden of running current network protocols. Another survey identifies the security concerns of IoT in the following areas [67]: 1) Front-end
sensors and equipment; 2) Network; 3) Back-end of systems; 4) Privacy in device; 5) Privacy during communication; 6) Privacy in storage; 7) Privacy at processing.

Security in computer networks is influenced by the communication protocols they use. IoT ushers in a new era of network protocols. Until those protocols are standardized and supported, IoT will make use of existing technologies. To support the maximum level of network security, devices in IoT will be burdened by overhead in both processing power and battery level [39]. The effect that security mechanisms will impose to the energy of resources of an IoT device is identified in [21] as well, where a new security algorithm for Secure Socket Layer/Transport Layer Security (SSL/TLS) to address the issue is proposed.

The scalability of IoT, when compared to today’s Internet, is analyzed in [133]. In the paper states that due to the scale IoT systems, its main challenges will be the authentication, identification, and naming of devices. As a result security mechanisms and frameworks for the IoT need to consider the level of expertise and background knowledge of their adversaries.

IoT, due to its magnitude and relatively young age, may be considered the technological field with the largest attack surface today [99]. The insecurity of IoT is demonstrated by a large number of surveys that have identified a variety of security issues and challenges found in IoT systems [5, 11, 39, 53, 69]. However, there have been few attempts to address its security issues from a security requirements point of view.

Some academic works identify security challenges in IoT while proposing future directions that security researchers should take. In [63] an IoT system is conceptually separated into different layers and proceeds to identify the security challenges of each layer and states the options that researchers have to prevent such issues. Each conceptual layers imposes different requirements and challenges on the IoT.

An attempt was made to provide a framework for security and privacy in IoT systems using requirements engineering in [9]. The authors identify the complexity of analyzing security in IoT systems and states that the key components in IoT are only two: RFID systems and networks of sensors. To reason about security in IoT, they propose the use of i* framework to undertake security analysis in future case studies. In the paper, other technologies and topologies that are common in IoT systems, are not considered. For example, IoT is not restricted to RFID systems but can use any communication technology, such as Wi-Fi, NFC or Bluetooth. Moreover, architectural topologies are not limited to networks solely comprised of sensors but may include any type of device capable of using a network protocol.
IoT systems as a whole are composed of a multitude of devices. Many of those devices are embedded devices. In [41] a vision in applying security engineering to embedded systems is illustrated. In the paper, the authors identify certain security challenges faced by embedded systems that should be addressed to have a secure system. Furthermore, They reason that security requirements tools should be designed, tailored to the needs of embedded systems. In [12] another framework aiming to provide security in embedded IoT systems is proposed. The authors introduce a basic three-step security framework to elicit requirements in embedded systems, by identifying the building blocks of embedded systems in IoT. In [116] a security framework specific to wireless sensor networks is presented. The framework proposes a system architecture that is broken down into eight modules. Each module has a specific functionality to mitigate security issues.

In summary, the presented works do not view IoT in a comprehensive manner. They only aim to mitigate security issues in domain-specific areas. Accordingly, they cannot be used to offer a universal security analysis to any IoT scenario, but only aim to address particular instances of IoT systems. The ability to argue and reason about IoT security issues is necessary to address specific use cases. However, it is also necessary to be able to reason about IoT security holistically without hindering a security engineer [96].

Certain issues and open challenges with the integration of IoT and Cloud computing are identified in [26]. It is argued that IoT is only made possible through the cloud infrastructure. Some IoT systems will use sensors as a service from a third party provider, while other IoT systems may use cloud services to offload heavy processing functions. The work states that IoT will function as a middleware that will transmit all its data to the cloud for processing. It is argued that the current trend for IoT application development is based on Cloud computing.

A framework for modeling and assessing security in an IoT system is proposed in [35]. The framework supports a graphical security model that evaluates the level of security using specific security metrics. The security of an IoT system is assessed in a comprehensive manner and is not limited to a specific IoT scenario, such as embedded systems or RFID systems. Another framework that separates the security requirements of IoT systems depending on their architectural layer is proposed in [94]. In their work, they suggest a four-layer approach, with each layer having different security needs. They state that their framework can be used by other researchers to build new security solutions for IoT.
In terms of monitoring and pattern analysis for IoT, in [107] SERENITY proposes to address the security and dependability (S&D) in the world of Ambient Intelligence (AmI) by providing an infrastructure for development and application of adaptable S&D solutions in dynamic and continuously changing AmI ecosystems. In [89] the construction of complex networks preserving Security and Dependability (S&D) properties is necessary to avoid system vulnerabilities, which may occur in all the different layers of Software Defined Networking (SDN) architectures is proposed. A model-based approach to support the design of secure and dependable SDN. This approach is based on executable patterns for designing networks able to guarantee S&D properties and can be used in SDN networks. In [91], to ensure the preservation of security, which is a key requirement and challenge for Service-Based Systems (SBS) due to the use of third-party software services not operating under different security perimeters. An approach for verifying the security properties of SBS workflows and adapting them if such properties are not preserved is presented. The approach uses secure service composition patterns. These patterns encode proven dependencies between service level and workflow level security properties. These dependencies are used in reasoning processes supporting the verification of SBS workflows with respect to workflow security properties and their adaptation in ways that guarantee the properties if necessary.

ENISA’s threat intelligence report [13, 102, eni] highlights the importance of cyber threat intelligence to respond to increasingly automated attacks leveraging automated tools and skills. Low-capability organizations/end-users have no access to cyberthreat intelligence solutions exposing them to severe risks of compromise. State-sponsored activities have shifted towards a reduction in the use of complex malicious software and infrastructure to low profile social engineering attacks. In the report the following security issues are highlighted:

1. Mail and phishing messages have become the primary malware infection vector;

2. Exploit Kits have lost their importance in the cyberthreat landscape;

3. Cryptominers have become an important monetization vector for cyber-criminals;


5. Skill and capability building is the main focus of defenders. Public organizations struggle with staff retention due to strong competition with industry in attracting cybersecurity talents;
6. The technical orientation of most cyberthreat intelligence produced is considered an obstacle towards awareness raising at the level of security and executive management;

7. Cyberthreat intelligence needs to respond to increasingly automated attacks through novel approaches to utilization of automated tools and skills;

8. The emergence of IoT environments will remain a concern due to missing protection mechanisms in low-end IoT devices and services. The need for generic IoT protection architectures/good practices will remain pressing;

9. The absence of cyberthreat intelligence solutions for low-capability organizations/end-users needs to be addressed by vendors and governments.

Similar findings are presented by NIST’s specialized IoT security annual report [58]. Trustworthiness is the degree of confidence one has that the system performs as expected with characteristics including safety, security, privacy, reliability, and resilience in the face of environmental disruptions, human errors, system faults and attacks.” Cybersecurity is defined as the prevention of damage to, unauthorized use of, exploitation of, and if needed the restoration of electronic information and communications systems, and the information they contain, in order to strengthen the confidentiality, integrity, and availability of these systems. Trustworthiness of IoT systems will require active management of risks for privacy, safety, security, etc. Cybersecurity risk management for IoT systems will continue to be a major factor in the trustworthiness of IoT applications. IoT components have the capability to connect to the Internet, being Internet Protocol (IP) based, but may also be deployed in stand-alone IP networks that are not connected to the Internet. In addition, IoT includes the facilities that allow users and organizations to analyze and understand the data gathered and actions taken by the things. With the changing threat environment, the cybersecurity needs of the future including the data that informs reports and controls functionality of the IoT should be considered. Although not specific to IT security, privacy, safety, authentication, and resilience provide contributions to IT and cybersecurity. Evolutions in system security engineering approaches can aid in the reduction of susceptibility of systems to a variety of simple, complex, and hybrid threats including physical and cyber-attacks, structural failures, natural disasters, and errors of omission and commission. One ongoing challenge is to reduce the susceptibility of systems to a variety of simple, complex, and hybrid threats including physical and cyber-attacks, structural failures, natural disasters, and errors of omission and commission. This reduction is accomplished by fundamentally understanding stakeholder protection needs and subsequently
employing sound security design principles and concepts throughout the system life cycle processes.

2.2.1 IoT modeling and visualization

In the literature, certain works that visualize and model specific aspects of IoT systems have been identified. Many of those works focus on modeling the sensor aspect of IoT while other works focus on modeling the services provided by an IoT system.

In [127] the reference models of IoT are divided into 1) Semantic, 2) Internet, and 3) Things oriented. A variety of conceptual layers for IoT have been proposed by the literature. Initial work defines [77, 132]. A three-layer architecture that consists of the Application Layer, Network Layer, and the Perception Layer. Other works identify different architectures that provide more levels of abstraction. For example, a Service Oriented Architecture based approach identifies five layers, application, service composition, service management, object abstraction, objects [11]. Another approach in [111] identifies other layers, that are, application, middleware, coordination, backbone network, access layer, edge technology. The proposed architectures for the IoT have yet to fuse into a single reference model [62], for that reason we chose the three-layer approach. It provides the necessary properties for reasoning about security while allowing to be extended if more levels of abstraction are introduced into the final reference model of IoT. The layers of IoT architecture should not be confused with the OSI model [64] since the two models try to conceptualize different constructs and concepts.

In their work about Service-Oriented Middleware for the IoT [42, 114], the authors propose an ontology for IoT. Their ontology models three aspects of the real world present in the IoT. The first aspect is the “thing” described in a Device Ontology. The second aspect consists of concepts and functionality of “things”, modeled in a Domain Ontology as mathematical formulas, and third is an approximation aspect that describes models to be used to approximate unavailable services and estimate missing information. This work does not take into account the security issues that can be faced in an IoT system and does not provide a way to model the social components of IoT systems. The ontology cannot be extended for security analysis without significantly altering its core concepts. The alteration will have an impact on the semantics and syntax of the ontology. As a result, a proposed extension will not be compatible with the original work.

OntoSensor [98] constructs an ontology-based descriptive specification model for sensors by excerpting parts of SensorML [17] descriptions and extending the IEEE
Suggested Upper Merged Ontology (SUMO). Another ontology that models network sensors is SenaaS [7]. The approach of SenaaS is sensor-as-a-service by realizing the event-driven service-oriented architecture (SOA) in the IoT domain. A similar approach is used in [25] for the modeling of services in IoT. Their proposed model captures the components of the IoT domain and provides a formal representation of the interactions. Their work is based on SENSEI [121]. SENSEI was aimed at realizing ambient intelligence in future networks and service environments by developing a framework of universal service interfaces for wireless sensor and actuator networks (WSANs). The core modeling concept considered in SENSEI is the “resource”, with all sensors, actuators, and processors being modeled as resources. All those works have in common the modeling of IoT sensors, IoT services or both. The heavy focus on sensors is limiting their application to other device and network types, which are common with IoT system deployments. The social aspects of such networks are not taken into account and as such the ontologies do not provide a way to model users and people. Another missing component of those ontologies is the security aspect of those networks. Those ontologies cannot be extended with components to describe security, social and additional network paradigms without impacting their syntax and semantics. The components used to describe network connections, and other network components do not directly map with real-life network systems. As a result, those components would need to be redesigned for the extensions. The extensions will not be compatible with the original versions of the ontologies.

An ontology for security-enabled IoT with a focus on the interoperability was proposed in [8]. They suggest a functional architecture of the IoT framework that incorporates secure access provision. Their work aims to address how different security attributes and constraints lying in different administrative domains will work together to secure an integrated operation. Their paper highlights an important security issue faced in IoT systems: how the same system is affected by different administrative domains. For example, applications operating on the Cloud share resources with other tenants operating on the same physical device. Each tenant will have different requirements that in some cases may be conflicting.

In [49] IoT systems are expressed using a chemical computing approach. The authors argue that the complexity of IoT can be modeled in a similar manner to chemical computing models. Their model can express social components using the User Plane. In [65], the authors use the Cognitive Agent-based Computing (CABC) framework to model a Complex communication network. Social constructs are modeled
with the use of the Agent construct. The limitation of those works concerning the existing literature is that the security components of an IoT system are not modeled.

ThingML is developed as a domain-specific modeling language which includes concepts to describe both software components and communication protocols. The formalism used is a combination of architecture models, state machines, and an imperative action language [119]. ThingML is supported by a set of open source tools that are built using the Eclipse Modeling Framework. The goal of ThingML is to a model-based approach to develop and deploy an IoT system. A model is used to auto-generate code that can be uploaded to the devices of the system. ThingML was published during this project and share certain similarities with Apparatus. Because ThingML focuses on the development of IoT services, its modeling language is used to model systems at the design and runtime time. In comparison, the Apparatus focuses on the security assessment of IoT systems. Its modeling language is used to model systems at the design and implementation phase. Also the Apparatus modeling language can be used to model the hardware characteristics of a system and its interactions with social components. Those concepts are absent on the ThingML modeling language.

SenseSim is an agent-based and discrete event simulator for IoT [28]. It can be used to simulate heterogeneous sensor networks and observe the changing phenomena. The simulator utilizing a perception model understands phenomena such as weather changes, fires or car traffic and can react according to them. SenseSim was developed to augment the perception of sensor networks. While those networks can be configured to respond to security threats, the tool was not designed to facilitate security analysis.

UML4IoT [115] is an approach based on a UML profile for the IoT is presented to fully automate the generation process of the IoT-compliant layer that is required for the cyber-physical component to be effectively integrated into the modern IoT manufacturing environment. The approach can also be applied at the source code level specification of the component in the case that a UML design specification is not available.

A similar approach is used in MDE4IoT [22], where a Model-Driven Engineering (MDE) as a key-enabler for applications running on intelligent distributed IoT systems. MDE helps in tackling challenges and supporting the lifecycle of such systems. The MDE approach enables the modeling of things and supporting intelligence as self-adaptation of Emergent Configurations in the IoT.

A Visual Domain-Specific Modeling Language was developed for the IoT [29]. The aim of the language for IoT, which is powerful enough for professional, but at the same
time simple enough to be understood by a non-technical end user that provides the requirements. The proposed language extends the UML specification with additional concepts that are capable of expressing specific IoT concepts.

The majority of the works described were not developed for security analysis or modeling social interactions. Instead, they focused on modeling specific aspects of IoT. A common saying among security experts is that human is the weakest link in the security chain. Social components and their interaction with the other components of a system is an important part of the security analysis of any system.

2.3 Security Requirements Engineering

The degree of success of a software system is the extent to which it meets its intended purpose. The process of discovering that purpose by identifying stakeholders and their needs is the field of requirements engineering [84]. Requirements engineering advocates the identification of Security Requirements in the early stages of product development [74]. The need to use a systematic approach to producing better requirements is highlighted in “Requirements Engineering: A Good Practice Guide” [106]. Some approaches have been developed so far that provide a set of instructions to help identify the security requirements of a developing product. Security engineers use security frameworks to produce security requirements. Requirements frameworks can be classified according to their approach to how they are used to model a system from a core concept. Popular approaches are (1) Goal modeling: which uses the concept of a goal as a core concept [92] and (2) Threat modeling: which models how threats are affecting a system [101].

Common Criteria is a security framework that proposes certain steps that once followed will result in a list of the security requirements for the system. Common Criteria is one of the oldest security frameworks, and a large number of other frameworks are built upon it [45]. System Quality Requirements Engineering (SQUARE) develops security requirements by having requirements engineers interact with the stakeholders of the IT project and translating their security goals into security controls [75]. CLASP [117] is another methodology that specifies a set of processes that can be integrated into the software development circle [120]. Secure Tropos is an extension of the Tropos methodology, that aims to incorporate security concerns throughout the development stages [36]. Secure Tropos, similar to Tropos, is a goal-oriented security model. ModelSec is modeling approach to security, targeting the software engineering field. The methodology automatically generates security artifacts based on the architecture
of the software system; the tool is build using Eclipse [100]. Existing research in
counter networks can be extended to describe IoT networks from an object-oriented
approach [Kamienski et al.].

The National Institute of Standards and Technology (NIST) organization developed
a framework for cybersecurity. The main difference with the frameworks mentioned
above is that it is based on security standards namely Cobit [105] and SOX [68], instead
of system requirements [112]. The framework can be used to design models of a system
that complies with security standards. Security Requirements Engineering Process
(SREP) used Common Criteria as a basis, trying to improve it by modernizing its com-
ponents with policies for distributed networks and multiuser ownership. SREP is UML
complaint, and the resulting security models evolve along the with the development
cycle of the product by performing some activities in each iteration step [76]. Haley
proposes a security framework that identifies security goals based on the assets of the
system. From the security goal, the security requirements are derived, while they are
validated using a process named satisfaction argument [44]. Bostrom et al. proposes a
framework that views security requirements from the agile development perspective
while focusing on extreme programming [16]. Microsoft Trustworthy Computing Se-
curity Development Lifecycle (TCSDL), identifies security activities that take place
in different stages in the development cycle. Compliance with standards is of high
importance as are security requirements based on customer satisfaction [66], especially
in industrial settings. A framework proposed in [93] suggests four steps in security
analysis that should be performed by the developers instead of requirements engineers.
Those are: (1) Identify the security environment and objectives; (2) Determine the
threat model; (3) Choose a security policy that includes prioritizing according to the
information’s sensitivity; (4) Evaluate risk.

Due to the IoT usage of various communication technologies, its security can benefit
from an established body of literature specific for each technology. For example, security
requirements specific to Wi-Fi or Bluetooth networks can be used with little alteration
in an IoT scenario that uses the protocols for communication. Wang proposed a
framework to address security wireless communications in Smart Distribution Grid
(SDG) [124]. While the nature of the framework is highly specific, it can prove valuable
to the present research since it aims to mitigate issues in smart wireless networks. The
framework identifies the scenarios of a smart grid, the common security issues and how
to address them using security mechanisms. Another focused security framework is
Privacy-Enhanced yet Accountable seCurity framEwork (PEACE), tailored for wireless
2.4 Research Gaps and Challenges

From the evaluation of the works identified via the literature review performed in this chapter, the fundamental characteristics and requirements are determined. Then, the current frameworks are evaluated based on relation to IoT security to identify the research gaps.

Fundamental Characteristics of IoT systems

IoT systems possess a number of fundamental characteristics that differentiate them from other system types. The following list of characteristics is derived from the literature [3–5, 11, 12, 39, 51, 53, 69, 80] above.

- **Interconnectivity**: Concerning the IoT, anything can be interconnected with the global information and communication infrastructure. In IoT, any network or device can be connected to a new network while still maintaining its previous network connections. It may result in accepting new connections and requests from any device. Interconnectivity is also a factor in communication since a device can only communicate with devices that support the same protocols. Each communication protocol has different requirements that may reduce or increase the security vulnerabilities of a network. Given the definition of interconnectivity, a local network once part of the IoT can be connected to any device globally available. The boundaries of networks are becoming more and more blurry [5, 11, 12, 39, 51, 53, 69].

- **Things-related services**: The IoT is capable of providing thing-related services within the constraints of things, such as privacy protection and semantic consistency between physical things and their associated virtual things. To provide thing-related services within the constraints of things, both the technologies in the mesh networks (WMNs). PEACE address the security issues by using a suite of authentication and key agreement protocols [95].

Engineering methodologies do not tend to use a universally accepted terminology. Definitions of similar terms have different meaning depending on the domain of each methodology. A survey of the most prominent Security Engineering methodologies, some of them include CLASP, SQUARE, identifies that a prevalent issue in the field is that there is not a universally accepted definition of the term “security engineering” [118].
physical world and information world will change. A thing is a physical or virtual object which is capable of being identified and integrated into communication networks. Both the physical and the virtual elements of a thing need to be captured in an IoT model since the connections between them need to be shown. Things need to communicate to exchange information and provide services to each other [39, 51, 53].

- **Heterogeneity**: The devices in the IoT are heterogeneous as based on different hardware platforms and networks. They can interact with other devices or service platforms through different networks. An IoT system may include traditional devices such as laptops and mobile phones, but also fridges, coffee makers, and cars. This wide range of devices will introduce different types of security issues in a system [5, 39, 51].

- **Dynamic changes**: The state of devices change dynamically, e.g., sleeping and waking up, connected or disconnected as well as the context of devices including location and speed. Moreover, the number of devices can change dynamically. Every change in the system introduces or removes things or stakeholders. Each change can cause a security vulnerability that has to be addressed by the security analyst [12, 51].

- **Enormous scale**: The number of devices that need to be managed and that communicate with each other will be at least an order of magnitude larger than the devices connected to the current Internet. The ratio of communication triggered by devices as compared to communication triggered by humans will noticeably shift towards device-triggered communication. Even more critical will be the management of the data generated and their interpretation for application purposes. This relates to the semantics of data, as well as efficient data handling. A security analyst may need to model systems that have three devices with a single service, to thousands of devices with hundreds of services [5, 11, 12, 51, 69].

### Challenges of IoT systems

IoT systems have a number of challenges to facilitate their usage. The challenges are derived from the literature [5, 11, 39, 51, 53, 69, 133] are the following:

- **Identification-based connectivity**: Connectivity between a thing and the IoT is established based on the thing’s identifier. This includes the possibility that heterogeneous identifiers of different things are processed in a unified way [11, 51].
• **Interoperability**: Interoperability needs to be ensured among heterogeneous and distributed systems for provision and consumption of a variety of information and services. Each thing in an IoT system will have a unique identifier, similar to an IP address in networks, MAC address in hardware and Serial numbers in software applications. The unique identifier is used to determine the chain of causality during security analysis. Different systems need to be able to exchange and use information without interference. In the case of proprietary protocols, it should not affect the interoperability with other protocols and platforms. The interoperability is a responsibility of manufacturers, but also a requirement of IoT systems [11, 51, 133].

• **Autonomic networking**: Autonomic networking (including self-management, self-configuring, self-healing, self-optimizing, and self-protecting techniques or mechanisms) needs to be supported in the networking control functions of the IoT, to adapt to different application domains, different communication environments, and large numbers and types of devices. IoT systems may be left unattended or accept connections from compromised devices. An IoT system can always be considered to be deployed in a hostile environment, for that reason autonomic networking is considered a requirement for IoT systems. Besides the hostility autonomic networking is used for quality assurance in a network, where nodes that are problematic are removed from the network [5, 11, 39, 51, 53, 69, 133].

• **Autonomic services provisioning**: The services need to be able to be provided by capturing, communicating and processing automatically the data of things based on the rules configured by operators or customized by subscribers. Autonomic services may depend on the techniques of automatic data fusion and data mining. Things in IoT may provide more than one services. Not all the provided services will be required by an IoT system [5, 11, 39, 51, 53, 69, 133].

• **Location-based capabilities**: Location-based capabilities need to be supported in the IoT. Communications and services will depend on the location information of things or users. It is needed to sense and track the location information automatically. Location-based communications and services may be constrained by laws and regulations and should comply with security requirements. Different physical locations may have different regulations when it comes to digital information and hardware specifications. The legislation that the stakeholders of an IoT system must comply may have specific requirements that have to be addressed by the security analyst. Location information is also necessary for
the personalization of services since user and things have different requirements based on their physical location. For example, requests on the weather conditions are prioritized by the location of the request [5, 51, 53, 69].

- **Security**: In the IoT, every “thing” is connected which results in significant security threats, such as threats towards confidentiality, authenticity, and integrity of both data and services. A critical example of security requirements is the need to integrate different security policies and techniques related to the variety of devices and user networks in the IoT. Security issues can result from the connectivity of things with legacy devices or devices that are inherently insecure. If that type of connection cannot be avoided, the stakeholders must be aware of the security risks. “Who must secure what?” is another important security issue, given the high number of people who are involved in an IoT system. During security analysis, the assets of the system are identified in order to prioritize the allocation of resources [5, 11, 39, 51, 53, 69, 133].

- **Privacy protection**: Privacy protection needs to be supported in the IoT. Many things have their owners and users. Sensed data of things may contain private information concerning their owners or users. The IoT needs to support privacy protection during data transmission, aggregation, storage, mining, and processing. Privacy protection should not set a barrier to data source authentication. Certain IoT systems, such as health applications and home security need access to sensitive information. That information has certain privacy considerations. Stakeholders need to be aware of the sensitivity of their data and how is being handled by the various devices and applications [5, 51, 69, 133].

- **High quality and highly secure human body-related services**: High quality and highly secure human body-related services need to be supported in the IoT. Different countries have different laws and regulations on these services. Human body-related services refer to the services provided by capturing, communicating and processing the data related to human static features and dynamic behavior with or without human intervention. Laws and regulations are different in each country and may affect other requirements for IoT systems [51, 78].

- **Plug and play**: Plug and play capability needs to be supported in the IoT to enable on-the-fly generation, composition or the acquiring of semantic-based configurations for seamless integration and cooperation of interconnected things.
2.5 Evaluation of Security Frameworks

The literature provides us with certain security frameworks that are used to elicit the security requirements of systems.

IoT is an environment with specific characteristics and requirements. For a security framework to be appropriate for security analysis for IoT systems, it must be able to meet the characteristics and requirements of an IoT system mentioned above (Section 2.4).

The criteria that are derived from the literature are the following:

1. **Modeling of Computer networks:** the framework can be used to model devices, network connections, and resources common in computer networks [35, 125];

2. **Modeling of IoT systems:** the framework can be used to model devices, relationships, and resources common in IoT systems [35, 65, 119, 125];

3. **Modeling of the social aspects of a system:** the framework can be used to model the behavior of social concepts, such as user-machine interaction, stakeholders’ goals and ownership of resources [43, 82];

4. **Modeling of system Threats:** the framework can be used to model threats that are targeting a system and threat propagation inside the system [34, 65, 82];

5. **Modeling of system Assets:** the framework can be used to model the assets of a system, their value for the stakeholders, and their value for an attacker [34];
6. **Support for semi-automated analysis:** the framework can be used to semi-automate types of analysis. For example, it may enable the parsing information from the model of the system to perform an analysis with manual input from an engineer. Types of analysis include threat propagation, requirements elicitation or vulnerability identification [34, 81];

7. **Support for automated analysis:** the framework can be used to automate types of analysis. For example, it may be able to verify the correctness of a model or automated the creation of models based on captured information from a real-life system [78, 119].

8. **Software-based tool that supports the use of the framework:** A software tool helps in facilitating the application of the framework by automating mundane tasks and providing visualization aids during the analysis [81, 119];

9. **Modeling processes about extending existing implemented systems:** a characteristic of IoT is that IoT systems will be used to extend existing system architectures. That specific system architecture could include legacy systems that for some reasons cannot be upgraded. The framework should be able to be used to model extending existing systems as well as systems that are being developed from the start [16, 78];

10. **Modeling of systems during the design engineering phase:** the framework can model a system during the design phase of the engineering development circle [81];

11. **Modeling of systems during the implementation engineering phase:** the framework can model a system during the implementation phase of the engineering development circle [119].

The evaluation of the frameworks regarding the criteria is presented in Tab. 2.2. As shown in Tab. 2.2 there is not a single framework in the literature that satisfies all the criteria previously discussed. The majority of the identified works that can model IoT systems are not developed for the domain of security analysis. As a result, they cannot be used for security analysis of IoT systems. Another significant finding is that most works are developed for modeling of a system only during the design engineering phase. Implementation phase modeling is required for security analysis to reason about hardware and software vulnerabilities as well as policy implications of social constructs on the system.
### Table 2.2 Evaluation of Frameworks

<table>
<thead>
<tr>
<th>Criteria</th>
<th>[114]</th>
<th>[42]</th>
<th>[98]</th>
<th>[17]</th>
<th>[25]</th>
<th>[121]</th>
<th>[8]</th>
<th>[49]</th>
<th>[65]</th>
<th>[119]</th>
<th>[28]</th>
<th>[81]</th>
</tr>
</thead>
<tbody>
<tr>
<td>modeling computer networks</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>-</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>modeling IoT systems</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>modeling social aspects</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>modeling threats</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>modeling assets</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>semi-automated analysis</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>automated analysis</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>software tool</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>modeling extending systems</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>design phase modeling</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>implementation phase modeling</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td></td>
</tr>
</tbody>
</table>
2.6 Evaluation of the Terminology of Security Applications

The modeling language is used to create models of IoT systems for security analysis. The language is composed of a set of metamodels that define concepts to express IoT systems. The names of concepts of the modeling are based on the terminology of tools and frameworks used for network and security administration. This is done for two reasons. First, to make the terminology of the language familiar to network and security engineers. Second, to enable the processing and analysis of APPARATUS models with existing network and security tools. The list of tools and frameworks that were used to develop the terminology was based on academic literature [24, 61, 87, 109, 128]. Tab. 2.3 shows the terminology used by each tool and how it related to a high-level description of a system. The terminology of each tool and framework has been grouped based on their thematic context. For example, Maltego, a forensics and data mining application, refers to devices (physical or virtualized) as machines, while Nmap refers to them as nodes. Both terms are used to express the same entity in a system. Furthermore, in Tab. 2.3 terms that are explicitly used for each tool are excluded. Such terms are used to refer to the specific functionality of each tool and are not transferable to other tools. For example, Nmap, network discovery and security auditing tool, uses a particular terminology on the functions it offers based on the type of scan or security evasion techniques.
Table 2.3 Terminology in tools and frameworks in network security and administration

<table>
<thead>
<tr>
<th>Terminology</th>
<th>Metasploit</th>
<th>Nmap</th>
<th>Burp</th>
<th>Snort</th>
<th>Bro</th>
<th>Wireshark</th>
<th>Maltego</th>
<th>w3af</th>
</tr>
</thead>
<tbody>
<tr>
<td>physical or virtualized device</td>
<td>machine</td>
<td>node</td>
<td>target</td>
<td>target</td>
<td>target</td>
<td>node</td>
<td>machine</td>
<td>target</td>
</tr>
<tr>
<td>software application or software service</td>
<td>service</td>
<td>service</td>
<td>application</td>
<td>application</td>
<td>application</td>
<td>service</td>
<td>application</td>
<td>service</td>
</tr>
<tr>
<td>data and information</td>
<td>-</td>
<td>information</td>
<td>information</td>
<td>data</td>
<td>data</td>
<td>information</td>
<td>information</td>
<td>information</td>
</tr>
<tr>
<td>network communications</td>
<td>protocol</td>
<td>protocol</td>
<td>protocol</td>
<td>protocol</td>
<td>protocol</td>
<td>protocol</td>
<td>network protocol</td>
<td>protocol</td>
</tr>
<tr>
<td>group of devices</td>
<td>-</td>
<td>group</td>
<td>-</td>
<td>group</td>
<td>group</td>
<td>-</td>
<td>group</td>
<td>-</td>
</tr>
<tr>
<td>users and stakeholders</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>actors</td>
<td>-</td>
<td>-</td>
<td>actors</td>
<td>-</td>
</tr>
<tr>
<td>threats</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>threat</td>
<td>threat</td>
<td>-</td>
<td>threat</td>
<td>-</td>
</tr>
<tr>
<td>vulnerabilities</td>
<td>vulnerability</td>
<td>vulnerability</td>
<td>vulnerability</td>
<td>vulnerability</td>
<td>vulnerability</td>
<td>vulnerability</td>
<td>vulnerability</td>
<td>vulnerability</td>
</tr>
<tr>
<td>assets</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>asset</td>
<td>asset</td>
<td>-</td>
<td>asset</td>
<td>-</td>
</tr>
<tr>
<td>security mechanisms</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>mechanism</td>
<td>mechanism</td>
<td>mechanism</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
The security tools presented in Tab.2.3 use the same terminology to express systems. Security engineers will be familiar with such terms. A security framework that follows the same terminology will require fewer resources for training. Furthermore, it could be integrated into existing security analysis workflows, with minimum disturbance.

2.6.1 Summary

IoT systems have a number of characteristics, and research challenges that occur at different levels. From the design of an IoT system to its implementation and life-cycle management. The following list is a summary of the IoT characteristics, and research challenges that motivated the development of the research objectives of this thesis.

- **Interoperability challenges**: Heterogeneous and distributed systems of IoT systems for the provision and consumption of a variety of information and services is an important characteristic of IoT. Interoperability of IoT systems is characteristic that need to able to be expressed in the modeling language (Obj1, Obj2);

- **Identification-based connectivity challenges**: Connectivity between a thing and the IoT is established based on the thing’s identifier. IoT devices are expected to connect to systems without the assistance of users to provide usernames or passwords. In IoT scenarios, a thing’s identifiers could be provisioned at the manufacturing level. As a result, a thing’s identifier is a property characteristic that is included in the modeling language (Obj2);

- **Location-based capabilities**: Communication protocols and services depend on the location of things or users. Such protocols and services are subject to laws and regulations. The modeling language needs to be able to express the physical location as well as the spacial status (stationary, moving) of thing (Obj 3);

- **High-level Security challenges**: The connected natures of a “thing” results in significant security threats, such as threats towards confidentiality, authenticity, and integrity of both data and services. “Who must secure what?” is another important security issue, given the high number of stakeholders that are involved in an IoT system. The ability to express threats, vulnerabilities in addition to mechanisms is an important requirement of the modeling language. Furthermore, the frameworks need to support processes to different levels of security analysis (Obj 3, Obj 4);
• **Manageability challenges**: Manageability of IoT systems at the modeling level needs to be supported in the framework with the use of formal processes. IoT applications usually work automatically without the participation of people. However, the maintenance and security monitoring of IoT systems require human input (Obj 4).
Chapter 3

Apparatus Framework

In the present chapter, the APPARATUS security framework is introduced. The APPARATUS Framework is used to design and analyze IoT systems. The core premise of the Framework is that if an IoT system is modeled in a structured manner, it is possible to infer how each component of the system affects the overall system’s security.

The Framework is defined with a set of components. Different components of the Framework will be used by the security engineers to model and analyze specific aspects of an IoT system. Referencing the literature a modeling framework should consist of the following components: a modeling language, a modeling procedure, and an analysis procedure [55].

1. Modeling Language: contains the elements with which a model can be described. A modeling language is described by its syntax, semantics, and notation.

   (a) syntax: refers to the linguistic elements (concepts) of the modeling language. The syntax of the modeling language of APPARATUS is defined by Unified Modeling Language (UML) class diagrams. UML class diagrams are used to define the concepts of the modeling language.

   (b) semantics: refers to the meaning of the linguistic elements. The difference between the semantics and syntax can be illustrated by an example. The phrase “this water is triangular” does not convey any meaning but it is a grammatically correct phrase. The semantic rules will enforce the creation of models that convey their intended meaning.

   (c) notation: refers to the visual representation of the modeling language. The concepts of the language can be represented using specific symbols and colors that convey meaning to the reader.
2. **Modeling Procedure**: describes the steps necessary to create models using the modeling language.

3. **Analysis Procedure**: describes the steps required to produce results based on information derived from the models.

The Apparatus Framework has an additional supportive component, a software application, that was developed to facilitate its use.

1. **Software application**: named ASTo (Apparatus Software Tool) developed to support the framework. The software application is used to perform a variety of analysis processes on the models. The processes can be automated or semi-automated depending on the degree of human input required for quality results.

The components of the Framework are presented in Fig 3.1.

![Fig. 3.1 APPARATUS Framework components](image)

## 3.1 Modeling Language of Apparatus

The modeling language is used to create models of IoT systems for security analysis. The modeling language is composed of a set of metamodels that define concepts to
express IoT systems. An IoT system is a network of various devices, and for that reason, the first iteration of the conceptual model consisted of the concept of IoT node and the network connection [73]. The core argument of that approach was that using the information provided by the architecture of an IoT system along with the requirements of the system’s stakeholders; security requirements can be elicited. That approach was based on security analysis based on the hardware architecture of a system. Security analysis using hardware architectural information offers both advantages and limitations. The architecture of a system offers valuable information for security analysis, such as the supported protocols of network connections between nodes or the flow of data inside a network. On the other hand, non-hardware aspects of a system, such as user interaction or authentication mechanisms, are not expressed. Limitations of a hardware architectural approach can be mitigated by introducing non-hardware architectural concepts along with hardware architectural components. To that end, the conceptual model of APPARATUS is modular. The concepts of the modeling language are grouped into different modules based on their thematic context, to allow a security engineer only to use the modules that are required. The metamodel of APPARATUS incorporates concepts from computer networks such as network connections and network domains, social concepts from generic modeling languages such as actor and security concepts such as threat.

In this work a security requirement is defined as a “a restriction related to security issues, such as privacy, integrity and availability, which can influence the analysis and design of a multiagent system under development by restricting some alternative design solutions, by conflicting with some of the requirements of the system, or by refining some of the system’s objectives”. That approach is used by Secure Tropos concept of security constraint [36, 82]. A similar definition of security requirements is given by Haley [43]. A security requirement is defined as “constraints on the system’s functional requirements, rather than being themselves functional requirements”.

The modeling language is composed of two metamodels. The first metamodel provides concepts and constraints to model an IoT system during the design engineering phase. The second metamodel offers concepts and constraints to model IoT systems during the implementation engineering phase. The distinction is made due to the different requirements, and information engineers have about a system during each phase. During the design phase, an engineer models the idea of the system without being restricted by the hardware or software specifications. For example during the design phase, an engineer may require a system component that will function as an Intrusion Detection (IDS) system. The engineer may not know at the design
time whether the IDS will be a hardware device or a software application. During
the implementation phase whether the IDS will be a hardware device or a software
application is necessary since it affects both the topology of the network and its security
requirements. During the implementation phase engineers have more information about
an IoT system, such as versions of software applications, operating network ports, user
profiles, and external facing nodes. Information about the system’s architecture can be
included in the modeling instances of an IoT system to create models that accurately
represented real-life systems.

Models represent a specific configuration of an IoT system during its design and
implementation. The models are static and need to be manually updated by security
engineers. The static nature of the models is helpful during security analysis since no
dynamic changes can be introduced that may affect the output of the analysis. On
the other, if the models are used to monitor an active system, the static nature of the
models will induce additional maintenance cost since any change in the system needs
to be mirrored in the corresponding model.

Each engineering phase offers different types of security analysis. During the design
phase, an engineer can model the threats or security constraints of a system. However,
design phase security analysis cannot be used to model the vulnerabilities of the system
or security mechanisms that aim to mitigate them. Both the vulnerability and the
security mechanism are concepts of the implementation phase since they represent
specific weaknesses or improvements in the hardware or software components of a
system.

3.1.1 Design Phase Metamodel

The design phase metamodel provides a set of rules that design phase IoT models must
adhere to. The metamodel is defined via a UML class diagram. Each UML class defines
a concept that either describes a component of the system or behavior that impacts that
system. Concepts are composed of a set of attributes that capture specific information
of the model. Each concept, unless otherwise noted has the property description which
describes the component of the IoT system. The design phase metamodel has the
following concepts:

Network module

1. Device: initially named IoT node in [73]. It is an object of the physical world
(physical thing) or an object of the virtual world (virtualized thing) [51]. It is
used to represent either physical components, such as hardware-based actuators and mobile phones or virtualized components, such as cloud-based devices of an IoT system.

2. **Application**: is part of the information world (information thing) [51]. An Application represents a software component that is running on a Device.

3. **Micronet**: is a managed environment of Devices and Applications can be configured in term of their security and enable an IoT system to perform a function. Components that are part of a Micronet are not more secure than others. Security in this context should not be confused with trust. Components in a Micronet do not have inherent trust relationships between them. Stakeholders have control over the security configuration of a Micronet’s component. For example, the security of a mobile device that belongs to an employee cannot be fully configured by an organization. The employee can install malicious applications or even make changes to the system configuration of the Device. However, the employee might be security conscious and the device can be highly protected, but that security protection is provided by the employee rather than the organization. The Micronet as a concept highlights the components of a system that can be secured by the stakeholders to develop mechanisms to secure interactions with the rest of the system. Examples of Micronets are a smart home, an agricultural network of sensors or a company’s internal network. The boundaries of the Micronet are defined during the creation of a model by the engineer. For example, one Micronet can include only the devices that are part of a specific network domain, while another can include all the devices that are in the same room. The same device can belong to both Micronets, and each Micronet can impose different security mechanisms on the devices. The property of the Micronet is:

(a) **purpose**: describes the goal or the function of the Micronet. The purpose of the Micronet can abstract or as specific it is required by the security analyst. Examples of a Micronets purpose are “Provide wireless connectivity” and “Provide wireless connectivity to sensor S-181 and S-346 using 4G infrastructure”.

4. **Net**: represents environments that their security configuration is not known and their behavior cannot be configured by the security engineer. While Nets may not be malicious, they represent a level of danger to an IoT system that
must be taken into account during the model development. Similarly to the Micronet, the boundaries of a Net are defined by the engineer. Examples of Net are external networks to the IoT system that a security engineer either has little or no knowledge of, such as a third party cloud infrastructure or hostile deployment environments. It is possible that a Device can be part of Net and a Micronet. For example, an IoT system has a server that hosts a set of virtual machines for its users. While the engineer can configure the server, the usage of the virtualized assets of the servers is configured by the users. A malicious user can try to exploit the virtualized assets to compromise the server. As a result, the virtualized assets compose a Net and not a Micronet.

5. **Information**: represents data and knowledge in a system. Examples of Information are authentication logs, temperature data, access credentials, and user passwords.

**Social module**

1. **Actor**: is used to represent people or groups of people that interact with an IoT system [36]. An Actor can be a stakeholder of the system. An Actor may *never* be malicious. The concept of Actor can be used to represent groups of people with different privileges, such as root users or the administrative personnel of a University. The property of the Actor is the following:

   (a) *intent*: describes what the Actor wants to achieve or gain by interacting with the IoT system.

**Security module**

1. **Asset**: any actor, device, application or information of the system that either (1) is considered valuable by the stakeholders and needs to be protected; (2) a malicious actor wants; or (3) acts as a stepping stone to further attacks. Assets that are valuable to the stakeholders can be elicited during the requirements phase. On the contrary, assets that a malicious actor wants or can be used for further attacks are not always apparent. Examples of assets are the access credentials known by an actor, sensitive information stored in a database or a sensor that has read/write privileges to a server.
2. **Malicious Actor**: is a person with malicious intent. Malicious Actors are used to representing attackers or insider threats. The concept of the malicious actor is a generalization of the concept of Actor.

3. **Threat**: a function that can be used maliciously or a system that has the means to exploit a vulnerability of a legitimate system. A threat can only target an asset of the IoT system. The property of the threat is:

   (a) *threatType*: represents the classification of the threat according to the STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) acronym [103].

4. **Constraint**: is “a restriction related to security issues, such as privacy, integrity, and availability, which can influence the analysis and design of a system under development by restricting some alternative design solutions, by conflicting with some of the requirements of the system, or by refining some of the system’s objectives” [36]. The constraint has the following property:

   (a) *propertyType*: the classification of the constraint according to the extended CIA (Confidentiality, Integrity, Availability, Authentication, Non Repudiation, Authorization) triad [10].

### 3.1.2 Implementation Phase Metamodel

The implementation phase metamodel [73] refines the design phase with additional concepts and attributes. The added concepts and attributes represent information that is not known in the design phase and is beneficial for security analysis. For example, in the implementation phase, the security engineer knows the type of network protocols that will be used by the system. Moreover, the software versions of the devices that provide services to the system are known. That additional information can be used to elicit security issues that were not apparent in the design phase. Furthermore, information on an implementation phase model can be leveraged either automate or semi-automate certain types of security analysis. For example, the process of vulnerability identification requires hardware and software system information. During a security assessment of an existing system, vulnerability identification of a system entails penetration testing. Security engineers will enumerate information of a system through various tools. The resulting information will be used to identify the vulnerabilities of the system. In Apparatus, by incorporating that information
Fig. 3.2 Design phase metamodel
into a model, the process of vulnerability identification can be made at the model level, without affecting the actual IoT system. An added benefit is that engineers can experiment with various models that represent different system configurations to evaluate their attack surface. The implementation phase metamodel is shown in Fig. 3.3.

The refined concepts of the implementation phase are the: (1) Device; (2) Application; (3) Micronet and (4) Information. The added concepts are: (1) Vulnerability and (2) Mechanism.

The modules of the implementation phase metamodel along with their concepts are the following:

**Network Module**

1. **Device**: implementation phase concept, refines the design phase Device concept with additional attributes. The added properties of the Device are:

   (a) **layer**: the conceptual layer of the IoT architecture to which the Device belongs. APPARATUS uses a three-layer architecture that consists of the Application Layer, Network Layer, and the Perception Layer [77, 132]. Other works identify different architectures that provide more levels of abstraction. For example, a Service Oriented Architecture based approach identifies five layers, application, service composition, service management, object abstraction, objects [11]. Another approach in [111] identifies other layers, that are, application, middleware, coordination, backbone network, access layer, edge technology. The proposed architectures for the IoT have yet to fuse into a single reference model [62], for that reason we chose the three-layer approach. It provides the necessary properties for reasoning about security while allowing to be extended if more levels of abstraction are introduced into the final reference model of IoT. The layers of IoT architecture should not be confused with the OSI model [64] since the two models try to conceptualize different constructs and concepts. The value of the layer attribute can be (1) application, (2) gateway or (3) perception;

   (b) **type**: is used to define the kind of the Device. Examples of a Device type are, a sensor, a mobile phone or a server;

   (c) **service**: is the type of role or operation that the Device performs for the system. This value may include network services such as ssh, ftp, data processing filtering and relaying of data;
(d) **input**: what is required for the node to perform its role or operation. It takes an enumerated value as an input that is `dataEnvironmental`, `dataDigital`, `command`, `action`, `notification`, `trigger`;

(e) **output**: is the result of the Device operation or role. It may take the same values as the **input** property;

(f) **update**: how the software on the Device is being updated. The values of the update attribute are enumerated and can be automatic, action or false. Automatic updates are handled automatically from the system. The action attribute signifies that for the Device to update an action is required. That action usually requires human input to allow an update to download and execute. The false attribute used to represent Devices that cannot be updated in any manner.

2. **Connection**: the type of network communication protocol used between the Devices. The properties of the connection are:

   (a) **medium** the type of connection, it can either be `wireless`, signifying a connection using a wireless protocol or `cable`, signifying a connection using a wired medium. It takes an enumerated value as an input;

   (b) **listOfProtocols**: is a list of the supported network protocols by the network connection. It takes an array of string values as an input, each value in the array represents a supported network protocol.

3. **Application**: implementation phase concept refines the design phase concept of Application with additional attributes. The properties of the Application are:

   (a) **version**: the software’s version type number. For example, if the Application represents the iOS operating system, the version would be the iOS release version, such as v10.2.3.

   (b) **update**: how the Application is being updated. The update attribute of the Application is identical to the update attribute of Device.

4. **Micronet**: implementation phase concept refines the design phase concept with an additional attribute. The property of the Micronet is:

   (a) **state**: the nature of a Micronet concerning its Device network connectivity gateway layer. The **state** can either be `dynamic`, meaning that the Devices in the network change network domains during their usage or `static` meaning
that the Devices in the system do not change network domains. Examples of dynamic IoT systems are networks of vehicular fleets, drones, and other mobile devices since devices in such networks move distances geographically. Examples of static IoT are smart homes, and industrial IoT systems since devices in such systems are stationary during their lifecycle.

5. **Information**: implementation phase concept that extends the design phase concept of Information with an added attribute. The additional attribute of Information is:

(a) location: corresponds to the geographical location of the information stored in the device. It can be used to represent if the information is physically stored inside a network or are hosted by a third-party service. Moreover, different regions have different laws regarding digital information that ultimately affect the overall security of a system and the proposed constraints of the system.

**Security Module**

1. **Vulnerability**: a software, hardware or usage policy weakness that can be exploited by an adversary toward compromising a system. Hardware and Software Vulnerabilities can be identified using techniques such as penetration testing. Hardware and software vulnerabilities can be identified from public access vulnerability databases such as CVE ¹ and NVD ². Such databases store vulnerabilities using unique IDs. Vulnerabilities IDs are used among security engineers to exchange information about security incidents.

2. **Mechanism**: a Mechanism when implemented protects against one or more Vulnerabilities. If the Vulnerability is publicly identified and stored in a vulnerability database, a security engineer can use the proposed security mechanisms to mitigate it. A Mechanism could be applied dynamically when a certain event is detected by the system, or they can be a constant process. For example, during the event of a DoS attack, a system may enlist additional resources to spread the impact of the attack. Once the attack is mitigated, the system will release the additional resources to reduce its operational costs. The property of Mechanism is:

¹https://cve.mitre.org/cve/
²https://nvd.nist.gov/vuln/search
(a) trigger: is used to describe the behavior or event that will cause the application of the Mechanism. For example, a trigger could be a constant, meaning that the mechanism is continuously active, or it could be a detection of an attack;

(b) severity: the level of impact on the system if the vulnerability is exploited. It is classified based on Common Vulnerability Scoring System (CVSS) [104].

![Diagram of Implementation phase metamodel](image)

**Fig. 3.3 Implementation phase metamodel**

### 3.1.3 System State in the Apparatus Framework

The Apparatus Framework supports state management of IoT systems in the form of state diagrams. State diagrams are used to describe the behavior of a system based on internal or external events. In the domain of IoT, state diagrams are used to describe the dynamic aspect of IoT systems [122]. The aim of state diagrams in Apparatus is to allow an engineer to identify and describe the events that result in changes to a system model. For example, during the event of a Denial Of Service attack, a system could enlist the help of other devices to minimize the attack surface. Once the threat of
the attack passes, the system can revert to its original set of devices. Another example involves vehicular systems. Vehicular systems, due to their mobility, will connect to a variety of other systems during their life-cycle. Each of those connections can be described as an event that changes the configuration of the system.

Changes in the configuration of the system are not only about the addition or removal of devices and applications. During a configuration change a system may become susceptible to certain attacks and as a result, requires additional security mechanisms to preserve its security posture. For example, a smart office can restrict access to assets based on the type of users are in the office. One type of user may not have access to workstations or cabinets, but a different kind of user might. Such types of dynamic access controls can be described in APPARATUS with state diagrams.

The concepts of state diagrams [59] the Apparatus Framework are two:

1. **models**: a finite set of system configurations that are developed using the APPARATUS Framework. The visual representation of a model is a circle (node) and labeled with a unique description, symbol or badge.

2. **event**: is an occurrence that affects a model. It is used to represents transitions from one model to another. After an event has taken place, a model may transit to another model, or it may remain the same. An event could be either be an internal or external event of a model. The visual representation of an event is a directed arrow (directed edge) from one model to another model. An edge is labeled with a unique description, symbol or badge.

The graphical representation of APPARATUS state diagrams is a directed graph. Models are represented as graph nodes, while events are represented as directed edges. The source of the edge is a model before the event has taken place. The target of the edge is a model after the event has taken place. An example of APPARATUS State diagram is shown in Fig. 3.4. Model 1 represents the initial state of the system. When Event 1 is detected, the system moves to a different configuration called Model 2. In this particular example, Event 1 is a denial of service. The system when in Model 1 configuration does not have the necessary resources to mitigate the denial of service attack. To address this, the system is configured to use a third-party service with more resources to spread the impact of the attack without impacting its availability. The third-party service since it is an external component requires additional security mechanisms to be implemented to retain the security posture of the system. The model representing the system with the third-party resources and additional security mechanisms is Model 2. Once the denial of service attack is mitigated, the Event
2 is detected. In Event 2, the system releases the extra resources from the third-party service and returns to Model 1, which its initial system configuration. Event 3 represents an event that does not require the system’s configuration to change, such as an attack using an exploit that cannot compromise the system. Since the system can mitigate the attack with its existing configuration, the system remains in the Model 1 configuration.

Fig. 3.4 Example of an APPARATUS State diagram

3.1.4 Notation

Models of the APPARATUS Framework are represented in the form of graphs. Concepts are represented as graph nodes and the relationships between the nodes are represented as graph edges. Contrary to other modeling languages that use shapes or colors to distinguish concepts in a static manner [36, 81, 119], the APPARATUS Framework uses classes. The reasoning behind the class-based approach is the decoupling of the visual representation from its underlying meaning. Visual representation of APPARATUS models can differ, based on the engineers’ preferences and requirements. For example, an engineer may want to use a different shape and color for each concept in a model, while another may believe that the additional visualizations make the model less readable. In APPARATUS, visual representation corresponds to how we want models to look, while classes correspond to what we want to bring our attention on. Classes are used to add additional attributes, either visual or textual, on elements of the graph. Classes are applied dynamically on the graph, depending on the type of information that needs to be conveyed. For example, a security engineer wants to validate if all the threats in the model are mitigated. During the validation procedure, the engineer requires specific information from the model. The nodes and edges of interest are the concepts of Threat, Constraints and the relationship Mitigates. The rest of the elements of the graph are not relevant. To facilitate that type of analysis, we want
to bring attention to the elements of interest and blur all other elements. By adding different classes to the elements, we add visual and textual cues based on the analysis process the engineer is performing.

Each class has a definition and a description. The definition dictates the information that we want to convey to an engineer at a specific time. The description represents the visual and textual attributes that are being added to the element. While the definition of the classes cannot be changed, the description can be modified. The description of classes refers to the front-end representation of the models, and as a result, it can be interchangeable. For example, a set of class description can add visual components to nodes, such as colors or shapes, while another can add textual components such as labels.

The Apparatus Framework does not impose a specific way of representing models visually. However, a default front-end representation of Apparatus models are used by ASTo along with a defined set of class descriptions. The visualization of graphs ASTo uses a customizable color palette on the notation classes. Colors values are prefixed with $asto$. and the name of the color. Additional information about the front-end representation of ASTo’s models and their customization options is provided in Section 3.5. The notation classes are presented in Tab. 3.1.

<table>
<thead>
<tr>
<th>Class</th>
<th>Definition</th>
<th>ASTo Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>the element has no special condition</td>
<td>the element is colored as $asto$.text</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(light gray or black)</td>
</tr>
<tr>
<td>fade</td>
<td>the element has reduced focus</td>
<td>the element is colored as a normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>element with reduced opacity to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25%</td>
</tr>
<tr>
<td>first-selection</td>
<td>the element is the first selection of the engineer</td>
<td>the element is colored as $asto$.blue</td>
</tr>
<tr>
<td>second-selection</td>
<td>the element is the second selection of the engineer</td>
<td>the element is colored as $asto$.orange</td>
</tr>
<tr>
<td>attention</td>
<td>the element has security implications and requires the attention</td>
<td>the element is colored as $asto$.orange</td>
</tr>
<tr>
<td></td>
<td>of the engineer</td>
<td></td>
</tr>
<tr>
<td>protection</td>
<td>the element is improving the security posture of the system</td>
<td>the element is colored as $asto$.cyan</td>
</tr>
</tbody>
</table>
An example of how classes are applied dynamically in ASTo, following the threat validation described above, as an algorithm is the following:

1. The class *fade* is applied to all elements in the graph;
2. The class *attention* is applied to all threat concepts;
3. The class *protection* is applied to all constraint concepts;
4. The class *normal* is applied to all mitigate relationships.

By using that algorithm the attention of the engineer if focused on the elements of the model that are important during the threat validation process.

### 3.2 Modeling Procedure

The Apparatus Framework has a structured approach regarding its modeling procedure. Each engineering phase has a series of steps that when followed, produce Apparatus models.

#### 3.2.1 Design phase modeling procedure

Design phase models are developed when there is no existing infrastructure capable of supporting IoT systems. As a result, an IoT system has to be developed from the beginning. The model creation is undertaken by a security engineer, which during the workflow is referred to as the engineer. During the design phase model development an engineer has to elicit the requirements of the system from its stakeholders. The design phase model creation workflow is defined in a textual format as a series of steps. Moreover, a graphical representation of the steps is defined in Fig. 3.5.

1. The engineer adds the network components of the system. The components are derived from the network module of the modeling language. During this step, the engineer elicits the network components requirements from the Network Architect of the system.

2. The engineer adds the social components of the system. Social components represent the users of a system and are derived from the social module of the modeling language. The user requirements are elicited by the stakeholders of the system.
3. The engineer identifies the assets of the system based on the requirements of the stakeholders. The identification of the assets initiates the addition of security components from the security module of the modeling language.

4. The threats of the system are modeled based on the assets identified in the previous step. For example, engineers can use techniques such as threat modeling to evaluate the attack surface of the system.

5. The engineer performs security analysis on the model. Additional information about the security analysis processes is provided in Section 3.3.

6. The engineer proposes constraints to improve the security posture of the system. The constraints are based on the results of the security analysis, the stakeholder’s requirements and security interdependencies that may have resulted.

7. The proposed constraints need to be verified. The verification process can include only the engineer that performs the security analysis or the stakeholders of the system. If the verification of the model is successful the engineer proceeds to the next step. If not, the model creation process is moved to Step 3.

8. The final step is to verify the model of the system. The model verification process ensures that all the necessary components have been included in the model. If the verification is successful, the process is terminated. If not, the process is moved to Step 2.

### 3.2.2 Implementation phase modeling procedure

Implementation phase models can be developed based on an existing design phase model, by following the design-to-implementation transition rules. Another possibility for model creation is the extension of existing system infrastructure with IoT components. Similar to the design phase modeling procedure, the implementation phase modeling procedure is defined in a textual format as a series of steps. The graphical representation of the steps is shown in Fig. 3.6.

1. The engineer adds the existing network components of the system. The components are derived from the network module of the modeling language. During this step, the engineer uses the existing network diagram provided by the Network Architect of the system.
Fig. 3.5 Design phase flow chart
2. The engineer adds the additional network components of the system. The components are derived from the network module of the modeling language. During this step, the engineer elicits the network components requirements from the Network Architect of the system.

3. The engineer adds the social components of the system. Social components represent the users of a system and are derived from the social module of the modeling language. The user requirements are elicited by the stakeholders of the system.

4. The engineer identifies the assets of the system based on the requirements of the stakeholders. The identification of the assets initiates the addition of security components from the security module of the modeling language.

5. The threats of the system are modeled based on the assets identified in the previous step. For example, engineers can use techniques such as threat modeling to evaluate the attack surface of the system.

6. The engineer performs a security analysis of the model. The security analysis can be aided by tools that use information from the model to produce results. Additional information about the security analysis processes is provided in Section 3.3.

7. The engineer adds the identified system vulnerabilities based on the security analysis that has been performed.

8. The engineer proposes a series of mechanisms that protect the system from the vulnerabilities.

9. The engineer proposes constraints to improve the security posture of the system. The constraints are based on the results of the security analysis, the stakeholder’s requirements and security interdependencies that may have resulted.

10. The proposed constraints need to be verified. The verification process can include only the engineer that performs the security analysis or the stakeholders of the system. If the verification of the model is successful the engineer proceeds to the next step. If not, the model creation process is moved to Step 4.

11. The final step is for the engineer to verify the model of the system. The model verification process ensures that all the necessary components have been included.
in the model. If the verification is successful, the process is terminated. If not, the process is moved to Step 3.

3.2.3 Transition rules between the different engineering phases

During the security analysis workflow, a security engineer may have to create models of the same system in both the design phase and the implementation phase. This may be done during the standard development process of a system, from its design to implementation. In such a case an engineer will be able to use an existing design phase model to transition to the implementation phase. In that manner, engineers can reuse existing models to reduce the development process.

While the normal engineering workflow would progress from the design to the implementation phase, specific use cases would require an engineer to transition from the implementation phase to the design phase. An example of such a use case is the development of a redeployment of an existing IoT system. Redeployment of a system is more easily made in the design phase, where an engineer can abstract its components. However, if there is an existing implementation phase model, the process of transitioning to the design phase may not be considered cost-effective. In the Apparatus Framework an engineer can model the existing IoT system as an implementation phase model. Then the implementation to design transition rules can be applied to generate a design phase of the system.

To facilitate the transition process between the two engineering phases, we define a structured approach that can be used to transition between a design phase model and an implementation phase model and vice versa.

Design to Implementation phase transition rules

To transition a model from the design to the implementation phase we perform the following procedure:

1. Micronet concepts gain the state attribute.
2. Device concepts gain the layer, type, service, input, output, update attributes.
3. Design concept Devices that have the connect relationship with other Devices, replace that connection with the concept of Connection.
4. Application concepts gain the version, update attributes.
Fig. 3.6 Implementation phase flow chart
5. Information concepts gain the location attribute.

6. All other concepts remain the same.

**Implementation to Design transition rules**

To transition between an implementation phase model to a design phase model, we perform the procedure stated above in reverse:

1. Remove the attributes from the concepts that have been outlined during the steps above.

2. Remove the concepts of Mechanism.

3. Remove the concepts of Vulnerability.

Both procedures have been implemented in ASTo, where an engineer can perform them in an automated manner.

### 3.3 Model based Analysis

The purpose of IoT models in APPARATUS is to facilitate certain types of security analysis. APPARATUS Framework has processes for Automated analysis and Semi-Automated analysis. The automated analysis processes use a model as an input to produce an output, without requiring any other action from a security engineer. The semi-automated analysis processes require an action by a security engineer. That action is required either in the input or the output of the process.

**Semantic Validation of Security Concepts**

The Semantic validation is the process of validating the semantic correctness of a model. An example of Semantic validation in software development is the concept of Type checking. Type checking is a validation mechanism in statically typed programming languages. A type system is a set of rules that assign a property called type to the various constructs of a computer program, such as variables, expressions, functions or modules [90]. A similar mechanism to type checking is part of the Apparatus Framework. Certain attributes in APPARATUS models use enumerated values. Those values can be used to validate the semantic correctness of the model. The attribute of type of Threat & Constraint use enumerated values as an input. Each Threat type
can be mitigated by at least one specific Constraint type. For example, a type of Spoofing Threat must be mitigated by a type of Authentication Constraint. While it is possible for the Spoofing Threat to be mitigated by multiple Constraints of different types, for the model to be semantically correct one of those Constraint types should be Authentication Constraint. This semantic rule facilitates the analysis process of type checking, which validates the semantic correctives of Apparatus models. The type Threat-Constraint pairs are shown in Tab. 3.2.

<table>
<thead>
<tr>
<th>Threat</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Authentication</td>
</tr>
<tr>
<td>Tampering</td>
<td>Integrity</td>
</tr>
<tr>
<td>Repudiation</td>
<td>Authorization</td>
</tr>
<tr>
<td>Information Disclosure</td>
<td>Confidentiality</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Availability</td>
</tr>
<tr>
<td>Elevation of Privilege</td>
<td>Non-repudiation</td>
</tr>
</tbody>
</table>

The Semantic validation process is supported in both the design and implementation phase analysis. It can be performed either manually or automatically through the software application ASTo.

### 3.3.1 Implementation Phase Analysis

The additional information encoded in implementation phase models is used to facilitate the process of security analysis. Certain attribute concepts, such as the attribute of `update` of the concept Device can be used to identify choices or configuration that can result in vulnerabilities. While other attributes, such as the `state` of the concept, Micronet can be used to elicit the system’s security requirements.

Automated Analysis

The automated analysis that can be performed on APPARATUS models makes use of the enumerated values of metamodel’s concepts as well as the metamodel’s rules to produce results. The engineer’s input is not required in any part of the process.
Generating Implementation phase models from Network Capture files

Network capture files store network information in the form of packets. Packet analysis is used for a variety of applications in network security. Deep packet inspection is used to identify threats that are targeting the network or evidence gathering during forensic operations. A network capture file can be used to recreate the traffic of a network as well as the data that was being transmitted.

Implementation phase models in APPARATUS can be generated using information stored in network capture files. The initial challenge was to identify what type of information can be extracted from a network capture file that can be relevant to APPARATUS model generation. The most common format of network capture file is the \textit{PCAP Next Generation Dump File Format} (Pcapng) \footnote{https://wiki.wireshark.org/Development/PcapNg}.

Pcapng files can be used to identify the active machines in a system, their communication flow as well as the services they use to exchange data. In APPARATUS models that information can be modeled with the concepts of Device, Application, and Connection of the network module. Active Devices can be identified and modeled using the IP addresses encoded in a Pcapng file. Applications can be identified based on the open ports found in the corresponding IP addresses, while the Connections can be unidentified by the communication flow of each transmission.

The first step is to convert the Pcapng file into a text format as shown in Alg. 1. That will allow the extraction of information to construct the model. We display the content of the Pcapng file as \texttt{< sourceIP : port, targetIP : port, protocol >}. The Pcapng file can be converted to other formats using tcpdump\footnote{https://www.tcpdump.org/} or another similar tool.

\begin{algorithm}
\caption{Convert Pcapng to text file}
\begin{algorithmic}
\Procedure{convertPcap2Text}{pcapfile} \Comment{convert a Pcapng file into a text file}
\State \texttt{textfile = convert(pcapfile)} \Comment{write Pcapng content as \texttt{< sourceIP : port, targetIP : port, protocol >}}
\State \Return textfile
\EndProcedure
\end{algorithmic}
\end{algorithm}

The second step (Alg. 2) is to read the resulting text file and convert into a data stream for processing. Each data stream is a line of the text file that represents a network packet transmission. That data stream will be used as an input to the rest of the algorithms.
The third step (Alg. 3) is to store all the elements of the packet transmissions. All the IP addresses (sourceIPs, targetIPs) are stored in the `devices` Array. A large number of those addresses are identical since each IP address has more than one packet transmission. We store all the transmissions (sourceIP, targetIP, protocols) in the `allConnections` array.

Algorithm 3 Store Connections

```java
procedure STORECONNECTIONS(textData) ▷ stores the connections of nodes in Sets

  for all textData do
    row = eachLine.split(onElement) ▷ split each element in the line into a row
    sourceNode.push(row[1]) ▷ row[1] is the sourceIP:port
    targetNode.push(row[3]) ▷ row[3] is the targetIP:port
    connections.push(row[1], row[3], row[4]) ▷ row[4] is the protocol

  devices = newSet(sourceNodes.concat(targetNodes))
  allConnections = newSet(connections)

  return devices, allConnections
```

The fourth step (Alg. 4) is the remove the ports that each transmission is using to eliminate the repeating packet transmissions. To do that we use the first elements of the `allConnections` array and filter the `port` variable. We store the result in the `uniqueLine` array along with the protocol. After all the ports have been removed, we eliminate the repeating lines by creating a new Set named `uniqueConnection`.

```java
```

The third step (Alg. 3) is to store all the elements of the packet transmissions. All the IP addresses (sourceIPs, targetIPs) are stored in the `devices` Array. A large number of those addresses are identical since each IP address has more than one packet transmission. We store all the transmissions (sourceIP, targetIP, protocols) in the `allConnections` array.

Algorithm 3 Store Connections

```java
procedure STORECONNECTIONS(textData) ▷ stores the connections of nodes in Sets

  for all textData do
    row = eachLine.split(onElement) ▷ split each element in the line into a row
    sourceNode.push(row[1]) ▷ row[1] is the sourceIP:port
    targetNode.push(row[3]) ▷ row[3] is the targetIP:port
    connections.push(row[1], row[3], row[4]) ▷ row[4] is the protocol

  devices = newSet(sourceNodes.concat(targetNodes))
  allConnections = newSet(connections)

  return devices, allConnections
```

The fourth step (Alg. 4) is the remove the ports that each transmission is using to eliminate the repeating packet transmissions. To do that we use the first elements of the `allConnections` array and filter the `port` variable. We store the result in the `uniqueLine` array along with the protocol. After all the ports have been removed, we eliminate the repeating lines by creating a new Set named `uniqueConnection`. 
Algorithm 4 Remove Services

\begin{algorithm}[H]
\caption{Remove Services ($allConnections$)}
\begin{algorithmic}
\Procedure{removeServices}{allConnections}\Comment{removes the services (ports) for the devices}
\ForAll{allConnection}
    \State sourceIP = element[0].filter(port)
    \State targetIP = element[1].filter(port)
    \State protocol = element[2]
    \State uniqueLine.push(sourceIP, targetIP, protocol)
\EndFor
\State uniqueConnections := newSet(uniqueLine)
\State \Return uniqueConnections
\EndProcedure
\end{algorithmic}
\end{algorithm}

The fifth step as shown in Alg. 5 is to store only the one instance of each device and service port. Those instances will be used to create the model without any repeating elements. To do that we use the devices array that holds all the IP addresses and ports. Each element in the array is stored in the format of IP:port. The first instance of the element is the device IP while the second in the port. We remove the repeating instances by storing the result into Sets. The first Set, uniqueDevicesServices stores the devices and the port as key:value pairs. The second Set, uniqueDevices stores only one instance of each IP.

Algorithm 5 Store Unique Devices & Services

\begin{algorithm}[H]
\caption{Store Unique Devices & Services ($devices$)}
\begin{algorithmic}
\Procedure{storeUniqueDevicesServices}{devices}\Comment{stores the unique connections}
\ForAll{devices}
    \Comment{devices are in the form of $IP:port$}
    \State deviceIP = device[0]
    \State deviceService = device[1]
    \State uniqueDevicesServices = newSet(deviceIP[deviceService])
    \State uniqueDevices = newSet(uniqueDevices)
\EndFor
\State \Return uniqueDevicesServices, uniqueDevices
\EndProcedure
\end{algorithmic}
\end{algorithm}

The sixth step (Alg. 6) is to create the device concepts that will be part of the model from the elicited uniqueDevices. The uniqueDevices Set is used as an input. Each device is assigned an ID number. The device attribute description gets the IP of the device as a value.
Algorithm 6 creates the devices nodes in the model

\begin{verbatim}
procedure CREATE_DEVICES(uniqueDevices)    ▷ stores the unique connections
  idCounter = 0
  for all uniqueDevices do
    node.ID = idCounter
    node.description = deviceIP
    nodeContent = node.ID + node.description ▷ node information that will be
    displayed in the model
    idCounter+ = 1
  return nodeContent
\end{verbatim}

The seventh step (Alg. 7) is to create the application concepts and connect them to the corresponding devices. The \textit{uniqueDevicesServices} Set and \textit{commonPorts} are used as inputs. The \textit{commonPorts} is an Object that holds the numbers of known ports and their corresponding service name. For example, port 22 is mapped to the SSH service, while port 80 is mapped to the HTTP service. Each created application concept is assigned an ID number. If the port number can be mapped to a known service, the application description is assigned the number of the port and its service. If not, the description is only assigned the port number. Then the application concept is connected to its corresponding device concept by a \textit{runs} relationship edge. To make the connection we use the IDs of the application and the device.
Algorithm 7 creates the application nodes in the model, maps them to known services and connects them to the devices.

```
procedure CREATE_DEVICES_APPLICATIONS(uniqueDevicesServices, commonPorts)
  idCounter = 0
  for all uniqueDevicesServices do
    services = uniqueDevicesServices[i]
    for all services do
      node.ID = idCounter
      if service == commonPort then
        node.description = service + commonPort
      else
        node.description = service
      nodeContent = node.ID + node.description ▷ node information that will be displayed in the model
      edge.ID = idCounter
      edge.description = 'runs'
      edge.source = node.ID
      edge.target = device.ID
      edgeContent = edge.ID, edge.source, edge.target ▷ information about the edge that is displayed in the model
      idCounter += 1
    return nodeContent, edgeContent
```

The eighth step (Alg. 8) is to create the Connection concepts and connect them to the devices. Each network connection is assigned an ID and is connected to two devices. Two different edges need to be created and will be connected to the corresponding devices using the IDs that are stored in the `uniqueDevices` Object. The network connection’s protocol is derived from the `uniqueConnections` Array.
Algorithm 8 creates the connection nodes in the model

**procedure** CREATENETWORKCONNECTIONS(uniqueDevices, uniqueConnections)

\[ idCounter = 0 \]

**for all** uniqueConnections **do**

\[ node.ID = idCounter \]
\[ node.description = connection[2] \]
\[ node.listOfProtocols = connection[2] \]
\[ nodeContent = node.ID + node.description \]

**for all** uniqueDevices **do**

\[ sourceID = connection[0] \]
\[ targetID = connection[1] \]

\[ edge.id = sourceID + idCounter \]
\[ edge.description = 'connects' \]
\[ edge.source = sourceID \]
\[ edge.target = idCounter \]
\[ edgeContent+ = edge \]

\[ edge.id = targetID + idCounter \]
\[ edge.description = 'connects' \]
\[ edge.source = targetID \]
\[ edge.target = idCounter \]
\[ edgeContent+ = edge \]
\[ idCounter+ = 1 \]

**return** nodeContent, edgeContent

The final step (Alg. 9) is to create the model using the information of the nodeContent and edgeContent variables. The nodeContent contains the information of the nodes of the graph and the edgeContent contains the information on the edges of the graph. We concatenate both variables and use the result to generate the graph-based model of the system.

Algorithm 9 creates the implementation phase model model

**procedure** CREATEMODEL(nodeContent, edgeContent)

\[ model = nodeContent.concat(edgeContent) \]

**return** model
The presented model generation process from network capture files has been implemented in ASTo\textsuperscript{5} as an implementation phase module for an automated model generation.

**Security insights on Implementation phase models**

Attributes in the metamodels’ concepts that take enumerated values are used for providing security insights to the security engineer in an automated manner. Those insights can provide the security engineer with additional information about the security posture of the system by highlighting possible security issues of the system’s configuration. The provided insights are independent of the security mechanisms or threats the security engineer has included in the model. For example, a system could have a connection that supports the TELNET protocol, which lacks encryption during data transmission. An insight could be to “use a secure transmission channel for wireless protocols that lack encryption”. The same insight would have been provided even if the security engineer had already added an encryption mechanism to the system. The reasoning behind this approach is that during the analysis stage, the security engineer should have as much information as possible to make informed decisions. The security insights are provided based on a high-level view of the security posture of the system and are independent of the system’s implementation mechanisms. The effectiveness of the mechanisms is dependent on current best practices. For example, the DES encryption algorithm was considered a robust encryption algorithm during the first years of its implementation. Nowadays, it is regarded as an obsolete algorithm, and its use should be avoided\textsuperscript{[14]}. While assumptions on specific insights on which mechanisms are the best on the current model can be made, that does not necessarily mean that those mechanisms would be the best choice for the life cycle of the system. For the proposal of the mechanisms, the decision is up to the engineer. On the other hand, high-level security insights are not bound by the current best practices and as a result, are more suitable for the security analysis output of the Framework. For that reason, the output of the analysis process is considered as an *insight* rather than a *directive*. The aim of the security insights is to facilitate the decision-making process by highlighting the attributes of the model that can result in increasing the attack surface of the system. The list of insights is based on the security best practices of ENISA’s threat landscape survey [eni], and OWASP IoT security guide [85]. Their recommendations aim to provide a baseline security for IoT systems that is based on standards such as ISO/IEC 30141:2018 [2] and ISO/IEC 27000. However, a baseline

\textsuperscript{5}https://github.com/Or3stis/apparatus/blob/master/app/src/imp/pcapImport.js
security does not cover security requirements specific to each system. Additionally, security is an ongoing process that required regular iterations to maintain the system’s security posture. To account for this, the framework’s insights list can be updated through its supporting application ASTo. A security engineer can configure the list with bespoke security insights that only impact a particular model or a system. This function can be accessed from the settings options of the application. The security engineer specifies a pattern using the framework’s notation and then suggested security insight. A pattern follows the same algorithm as described above for pattern identification. Each pattern is composed of components. A component is a concept, its attribute, and the value of the attribute. For example, to trigger a security insight for a device at the layer of perception, the security engineer input the concept device, the attribute layer, and the value perception.

1. Insights based on the Device’s concept attributes

(a) layer: perception; Devices that are part of the perception layer have certain characteristics that can be used to elicit security requirements. Usually, perception layer devices can be physically accessed by the actors of the system. Examples of perception layer devices are light sensors, security cameras, mobiles phones or laptops. While actors do not pose a threat to the Devices, malicious actors do. The same physical access used by the actors of the system can be used by malicious actors aiming to compromise the system. Physical attacks against devices are one of the most of the effective way of compromising them since attackers can leverage attacks vectors to bypass software-based security mechanisms with ease. For example, a common attack vector with devices that include a USB port is to attach a USB stick with malicious software that will provide a control point from the compromised device to an external system [23]. Another attack vector is to use a USB loaded with an operating system. That operating system will then be loaded on the RAM of a compromised device. An attacker can then mount the hard drive system of the target machine as an external hard drive to access its secure contents [37]. Given the number of attack vectors that can be used when an attacker has physical access to a device, it is suggested to add security policies and mechanisms to protect against such attacks.

(b) layer: gateway; Gateway layer devices provide routing and other networking services. During certain system configurations, such devices can be external facing nodes. For example, a router that provides internet connectivity to
a computer network can communicate with devices from outside the said network. External facing nodes are susceptible to network attacks. A public internet facing node can be targeted by any device with internet connectivity. Due to the accessibility of external facing nodes, they are usually the target of malicious attackers. It is suggested if the gateway layer device is an external facing node, to introduce mechanisms to mitigate network attacks. Those mechanisms should include protection against information gathering and fuzzing attacks.

(c) layer: application; Application layer devices are commonly provided by third-party services. For example, an application layer device could be a cloud-based authentication server. Depending on the scenario a security engineer will be able to configure certain aspects of the device regarding security, but not all. The hardware aspect of the device, its physical security, and geographical location are a responsibility of the cloud provider. If third-party nodes are used, the system is inheriting the security posture of those parties as well as its own.

(d) input/output: dataEnvironmental; Environmental data are data gathered by the environment, such as temperature and light. Such data are not under the control of the security engineer and as a result, provide a unique attack vector. For example, the temperature monitor system of a server room might be programmed to shut down its server if the temperature gets above a certain threshold. If a malicious actor gained access to that room, a lighter could be used to heat up the temperature sensors. That would cause the system to shut down the servers despite the temperature of the room being in the allowed threshold. To mitigate such attacks, it is suggested to introduce security processes and mechanisms that take into account attack vectors that leverage environmental variables.

(e) update: action; The majority of devices require an action to be updated. That action usually requires the consent of a user. For example, operating systems updates of mobile phones require the user to accept the installation of the update. Updates include patches for security issues. If a user takes too long to install the update, the device is susceptible to vulnerabilities that could have been resolved by a firmware update. Because a user needs to update the device manually, a security engineer should propose a security procedure for handling the updating process. That procedure could flag non-updated devices as compromised until they receive the update.
3.3 Model based Analysis

(f) *update: false*; Devices that cannot be updated should be treated as compromised by the security engineer. Even if at the time of implementation phase there are no known exploits for the device, it is expected that an exploit will be developed in the future. Without a way of updating the device to address a vulnerability, a malicious actor will compromise it at some point.

2. Insights based on the Application’s concept attributes

(a) *version: <unencrypted protocols (TELNET, FTP, HTTP, SMTP, DNS)>*; Not all communication protocols use encryption by default. Some communication protocols, such as TELNET, only use encryption during the authentication phase and not during the transmission of data. The mentioned communications protocols are lacking regarding security. It is suggested, that if such protocols are used, the security engineer should introduce additional security layers to ensure secure communication. Without an additional security layer, the information transmitted could be compromised.

(b) *input/output: dataEnvironmental*; The insights of the Application regarding the environmental data are identical to the insights about environmental data of the Device.

(c) *update: action*; The insights of the Application regarding the update process are identical to the insights of updating a Device through an action.

(d) *update: false*; The insights of the Application regarding non-updatable application are identical to the insights of non-updatable Devices.

3. Insights based on the Connection concept attributes

(a) *medium: wireless*; Wireless communications due to the use of air as a transmission medium, are subject to a specific range of attack vectors. Information disclosure attacks are a common attack vector. During information disclosure attacks, a malicious actor aims to obtain information without being detected by the system. Other attack vectors include a combination of different attacks. For example, a malicious actor may spoof the identity of a legitimate system node and then perform a Man-In-The-Middle attack. Other attack vectors may not be as invasive. A malicious actor can infer a significant amount of information about a wireless network by passively listening for advertising probes. Such probes could include authentication requests or identification broadcasts. Wireless communications
mitigate such attacks by incorporating security mechanisms in their protocol specification [56]. As a security measure, wireless network connections are considered nodes that require the attention of a security engineer. The security mechanisms of certain wireless protocols are considered obsolete, as is in the case of Wired Encrypted Privacy (WEP) [113]. Other security mechanisms require additional security mechanisms to be implemented to mitigate certain attack vectors, such as the case of WPA2 [123].

4. Insights based on the Micronet’s concept attributes

(a) state: static; The network components of static Micronets retain the same gateway nodes for long periods of time. A security engineer should use that behavior to increase the security posture of the system. For example, the destination of certain types of network traffic should always be the system’s gateway layer. Layer 3 network requests are an example of such traffic. If those requests have a destination to a different node, whether inside or outside the Micronet, it could be an indicator of malicious behavior.

(b) state: dynamic; Dynamic Micronets represent a system that either moves geographically or changes gateway layer nodes at regular intervals during their lifecycle. When the nodes in the Micronet change gateway layer nodes, the possibility of a compromise is increased. The nodes must perform authentication and authorization procedures as well as establishing secure communication channels and sandboxed environments. During such moments of flux, a malicious actor can compromise legitimate system nodes. When security analysis of dynamic Micronets is taking place, a security engineer should consider the security implications that are imposed during the network movement of the system.

5. Insights based on the Micronet and Net requests relationship

(a) The system under analysis has at least one Micronet to Net or Net to Micronet requests connection.; In the Apparatus Framework, IoT systems are considered as systems deployed in hostile environments. The concept of Net is used to represent systems that a security engineer cannot configure concerning security. As such they are not considered secure networks. On the other hand, Micronets represents systems whose components can be configured regarding their security. Since Micronets’ security can be configured, Devices communicating inside Micronets share a level of security.
Micronets and Net can exchange data and establish connections. When a Device from a Micronet communicates with a Device from the Net it needs to establish a secure communication channel. If the establishment of a secure channel is not possible the Information that is being exchanged should be an Asset.

The process for providing security insights is presented in Alg. 10. The process takes two inputs. The modelElements are all the elements of the model (nodes, edges. The insightList is a list of the security insights based on the elements’ concept and attribute as shown above.

**Algorithm 10** Provide security insights based on the information on a model

1: procedure insights(modelElements, insightList)
2:     for all modelElements do
3:         if element.concept == insightList.concept && element.attribute == insightList.attribute then
4:             elementInsight = insightList.insight
5:     return elementInsight

This process has been implemented in ASTo \(^6\) as an implementation phase module. The security insights are also shown in Tab. [3.3, 3.4, 3.5, 3.4, 3.6].

**Semi-Automated Analysis**

The semi-automated analysis processes in the Apparatus Framework require human action that is performed by the security engineer. The human action will either be part of the input or output of the process. In both parts of the semi-automated analysis process, the security engineer leverages domain-specific expertise and knowledge of the system’s requirements to distinguish relevant information.

**Vulnerability identification on Implementation phase models**

The Vulnerability identification is a process that uses the values from the attributes of certain concepts to identify vulnerabilities of the system. The attribute values are used to create a list of keywords. The keyword list is then used as an input to a vulnerability dataset. The vulnerability dataset will produce a list of vulnerabilities based on the keywords.

\(^6\)https://github.com/Or3stis/apparatus/blob/master/app/src/imp/insights.js
\(^7\)https://github.com/Or3stis/apparatus/blob/master/app/src/imp/insightsList.js
### Table 3.3 Security insights based on the Device concept

<table>
<thead>
<tr>
<th>Concept</th>
<th>Attribute</th>
<th>Value</th>
<th>Insight</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer</td>
<td>perception</td>
<td></td>
<td>attacker may have physical access to device. Use security policies and mechanisms for physical security.</td>
</tr>
<tr>
<td></td>
<td>gateway</td>
<td></td>
<td>Use external facing security controls to mitigate network attacks.</td>
</tr>
<tr>
<td>Device</td>
<td>application</td>
<td></td>
<td>application nodes may be provided by third parties. Use security controls to mitigate the vulnerabilities inheritance.</td>
</tr>
<tr>
<td>input/output data</td>
<td>Environmental</td>
<td></td>
<td>Introduce security processes and mechanisms that take into account attack vectors that leverage environmental variables.</td>
</tr>
<tr>
<td>update</td>
<td>action</td>
<td></td>
<td>Use a security procedure for the manual updating of the device.</td>
</tr>
<tr>
<td></td>
<td>false</td>
<td></td>
<td>Non-updatable devices should be treated as compromised.</td>
</tr>
</tbody>
</table>

### Table 3.4 Security insights based on the Application concept

<table>
<thead>
<tr>
<th>Concept</th>
<th>Attribute</th>
<th>Value</th>
<th>Insight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>input/output data</td>
<td>Environmental</td>
<td>Introduce security processes and mechanisms that take into account attack vectors that leverage environmental variables.</td>
</tr>
<tr>
<td>update</td>
<td>action</td>
<td></td>
<td>Use a security procedure for the manual updating of the application.</td>
</tr>
<tr>
<td></td>
<td>false</td>
<td></td>
<td>Non-updateble applications should be treated as compromised.</td>
</tr>
</tbody>
</table>
### Table 3.5 Security insights based on the Connection concept

<table>
<thead>
<tr>
<th>Concept</th>
<th>Attribute</th>
<th>Value</th>
<th>Insight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>description</td>
<td>wireless</td>
<td>wireless protocols are susceptible to spoofing and tampering attacks. Use protocols with sufficient encryption.</td>
</tr>
<tr>
<td></td>
<td>listOfProtocols</td>
<td>unencrypted protocols</td>
<td>Such protocols do not provide encryption by default. Introduce additional security layers for a secure communication.</td>
</tr>
</tbody>
</table>

### Table 3.6 Security insights based on the Micronet concept

<table>
<thead>
<tr>
<th>Concept</th>
<th>Attribute</th>
<th>Value</th>
<th>Insight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micronet</td>
<td>state</td>
<td></td>
<td>nodes retain the same gateway node for long periods of time. Introduce security policies and controls using that benefit from a static configuration.</td>
</tr>
<tr>
<td></td>
<td>static</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>dynamic</td>
<td></td>
<td>System nodes do not retain the gateway node for long periods of time. Introduce additional controls to mitigate attacks during services hand-off.</td>
</tr>
</tbody>
</table>
A human action is required in the output of the process to identify which vulnerabilities are relevant to the system and which are not. Not all the identified vulnerabilities will be relevant to the system. A vulnerability may not match the system specification and be a false positive, or the existing system configuration may have mitigated the vulnerability.

The quality of the output is affected by the detail of the input. Generic attribute values in the model will generate a large number of vulnerabilities, while more specific values will generate a smaller but more relevant number of vulnerabilities. For example, if an attribute value is “Windows”, the process will output all the vulnerabilities that have the “Windows” keyword. That number could be thousands of vulnerabilities. Due to the generic term “Windows,” the majority of those vulnerabilities will be irrelevant to the model. On the other hand, if the value is “Windows XP service pack 2”, the output of the process will be a smaller number of vulnerabilities, but more relevant to the system.

The attributes whose values are used as keywords are shown in Tab. 3.7.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Attribute/Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>description</td>
</tr>
<tr>
<td></td>
<td>type</td>
</tr>
<tr>
<td></td>
<td>service</td>
</tr>
<tr>
<td>Application</td>
<td>description</td>
</tr>
<tr>
<td></td>
<td>version</td>
</tr>
<tr>
<td>Connection</td>
<td>listOfProtocols</td>
</tr>
</tbody>
</table>

Once the list of keywords is produced, the security engineer can use the list as an input to a vulnerability dataset. Each keyword will result in specific vulnerabilities. The security engineer will have to determine which of the identified vulnerabilities can affect the system and which will not.

The process of Vulnerability identification has the following steps:

1. Compile a list of keywords from the attributes of a model using the Tab. 3.7 as a reference.
2. Input the list of keywords through a vulnerability database.
3. Compile a list of vulnerabilities from the results of the vulnerability database search.
4. Remove the vulnerabilities that do not affect the system from the list.

The Vulnerability identification process can be executed either manually or automatically using the software application ASTo. The framework does not use any specific vulnerability dataset. It only provides a way to use information from a model to search for vulnerabilities from a dataset. This functionality is martialized by the tool, ASTo. A vulnerability dataset can be a third-party service or a bespoke database in the local network. The location of the database can be configured by the settings of the tool.

Pattern identification

The Pattern identification process takes user input to construct a pattern and locates the concepts in an Apparatus model that fulfill that pattern. This process is used when a security engineer wants to identify which concepts have a specific configuration or behavior. For example, a security engineer may need to identify which devices in a Micronet use the SSH protocol over a wireless medium to communicate with devices in a Net. Another example would be the identification of all the applications that use a version that can be exploited by a specific vulnerability.

The process of Pattern identification has the following steps:

1. Security engineer inputs the pattern as a keyword-based query.

2. Each keyword is an attribute or value that the user wants to be part of the pattern.

3. The attributes of the model are compared with each keyword of the query.

4. If a concept has at least one attribute or value that matches at least one keyword of the query, the concept is considered a match.

The Pattern identification process can be performed manually. However, due to the large size and complexity of models, manual application of this process can be error-prone. The complexity of the process increases with the size and complexity of the model. The suggested approach is to use the process through ASTo.

3.4 Discussion on the modeling language

In this section, we are going to discuss the design choices that were made during the development of the modeling language. The development of the modeling language
took some iterations until it has reached its present state. Each iteration was influenced by certain factors. The main driving factors were the requirements that have been identified during the literature review. Other factors included model creation, data visualization, ease of use and the need for specific types of analysis. Some design decisions impact how a model instance is developed. These design decisions are not readily apparent in the UML diagram. Certain decisions were made to highlight certain behaviors that affect the security posture of the system, while others were made to reduce complexity on large graphs.

**Design & Implementation phases:** The modeling language of Apparatus has two metamodels. One metamodel to express IoT systems during the design phase and one metamodel to express IoT system during the implementation phase. Certain works on domain-specific modeling languages focus on a specific engineering phase [36, 81, 119]. One of the requirements of the Apparatus framework was the analysis of IoT systems during the design and the implementation phase. During the development of an IoT system, the system will either extend an existing infrastructure, or it will be developed from the beginning. In the first case, an engineer has an existing implemented system and requires a modeling language able to express system during implementation. In the second case, an engineer is modeling the idea of the system first, before moving on the implementation of the system. As a result, the engineer requires a modeling language able to express abstract concepts. Moreover, each phase can be used to perform different types of security analysis. For example, identification of system vulnerabilities requires information that is accessible during the implementation phase. The implementation phase metamodel was developed as an extension to the design phase metamodel. Both metamodels have similar concepts. This was done to reduce the necessary concepts and definitions an engineer needs to be familiar with before using the framework.

**Module-based modeling language:** The metamodels of the Apparatus are divided into modules. Each module is composed of constructs that share a similar thematic context. One of the aims of the Framework is to provide a holistic approach to IoT security modeling. To achieve that, the components of the Framework would need to able express constructs such as hardware and software components of IoT systems, hard or soft assets, user-system interactions, network capabilities, threats, and vulnerabilities. Some of the constructs will contain similarities while others will not.

Furthermore, when modeling an IoT system, a single stakeholder may not have the necessary expertise to model all the aspects of a system. For example, while a
security engineer can model the high-level constraints of the system in the form of policies, the same engineer might not have access to the network infrastructure of the system. As a result, the engineer will not be able to model the network components of the system. To do that, the engineer may require input from a networking engineer or stakeholder familiar with the network architecture of the system. Stakeholders can use the different modules of the Framework with diverse expertise to provide a comprehensive model of an IoT system. Stakeholders with computer networking expertise can use the network module to model the networking components of the IoT system, while a stakeholder with system expertise can model the vulnerabilities of the system. The modular architecture of the Framework allows different stakeholders to model specific aspects of the system that relate to their knowledge.

The development of the modeling language followed a minimalist approach. The initial iterations of the language focused on modeling only the network aspect of an IoT system. IoT systems are characterized by their networking capabilities. They are bound to the capabilities of the network as well as their hardware. During the decision making the process for the necessary concepts and attributes, the question that we were trying to answer was “What does a security engineer need to know about a network to access its security?”. To answer that question we developed the network module of the modeling language.

IoT networks do not have clear boundaries. They can include devices operating from a variety of locations. They can include physical or virtualized hardware along with data stored in multiple locations. The lack of clear boundaries makes it harder for a security engineer to perform security analysis. For this reason the concepts of \textit{Net} and \textit{Micronet} were created. To address the scalability of models that are expected to be composed of thousands of nodes, the concepts of Micronet and Net were developed. They are used to reduce “noise” in visual models by grouping components and abstracting systems based on common goals and functionality. Instead of analyzing systems composed of thousands of identical nodes, security engineers can group such nodes into Micronets and Nets. When the security analysis of the individual nodes inside a Micronet or a Net needs to take place, an engineer can only view and analyze those. In \textsc{Apparatus} Framework, IoT systems are considered as systems deployed in hostile environments. The concept of Net is used to represent systems that we cannot configure regarding security. As such they are considered compromised and malicious. On the other hand, Micronets represents systems whose components can be configured concerning their security. For security purposes, Micronets and Nets have a Zero Trust Network architecture [60].
3.5 ASTo: Software Application of Apparatus

The proposed framework is supported by a software application to facilitate its usage and enable the security analysis on the IoT models. The software application is named ASTo (Apparatus Software Tool).

3.5.1 Requirements of the software application

The software application was designed and developed with certain high-level requirements in mind.

- **R1: Cross-platform application**: the application should be able to function in the major operating desktop systems. The targeted operating systems are Windows, macOS, and Linux distributions.

- **R2: Developed using standardized technologies**: the application should be built using technologies that are standardized. Technologies that are part of a standard provide a more stable development environment. Their improvements are based on the consensus of the community, rather than the goals of a single corporation. For example, the development of web technologies is made through a variety of consortiums, where individuals, organizations, and corporations are members. On the other hand, the development of the Java\(^8\) or Dart\(^9\) programming languages are being made by a single corporation. Oracle\(^10\) in the case of Java and Google\(^11\) in the case of Dart.

- **R3: Developed using open source software**: the application should only use libraries and framework that are licensed under open source license. Open source licenses do not require payment or any other reimbursement. They also allow freedom in the distribution of software. Open source software has larger and more active development communities than proprietary software. The software application will also be developed under an open source license.

- **R4: Modular approach**: the functionality of the application should be separated into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality. This will allow

---

8https://www.oracle.com/java/index.html 
9https://www.dartlang.org/
10https://www.oracle.com/index.html 
11https://www.google.com
reuse of modules across the code base. The modular approach will make the testing of the application easier since each module can be tested independently regarding its functionality. Moreover, extending the functionality of the application will only require the development of specific modules, instead of major rewrites of the code base.

- **R5: Easy to extend its functionality:** the functionality of the application should be able to be extended without impacting the overall state of the application. For that reason, the application will allow extensions in the form of modules. Those modules will provide the API of the application to communicate safely.

- **R6: Supports analysis of Apparatus models:** the primary requirement of the software application is to provide support in using Apparatus framework. The application should facilitate the creation of models, the use of the processes and types of analysis supported by the Framework. Functionality that is aimed at the analysis of other framework and modeling languages will not receive first-class support.

- **R7: Configurable working environment:** the environment of the application should be configurable to the needs of the engineer. The application should allow the configuration of different color themes, fonts and User Interface element sizes. Configurable environments improve the productivity of the users and increase the functionality of the application [110]. Moreover, users that are color blind or light sensitive can configure the environment to their preferences.

The requirements R1, R2, and R3 will define the frameworks and libraries that will be used to develop the application. Requirements R4, R5, R6, and R7 will define the architecture and the development style of the application.

To comply with the requirements above, the software application is developed using web technologies. Web technologies offer a wide range of libraries and frameworks that are Open Source and actively maintained. The software application is based on the Electron [50] framework. Electron is a framework for creating native applications with web technologies such as JavaScript, HTML, and CSS. It is compatible with all major operating systems such as macOS, Windows, and Linux. Electron is an open source project under the MIT license. It is maintained by GitHub and an active community of contributors. Electron offers certain advantages not found in other frameworks. Because it is a native application built on web technologies, the same code base can be used to create a web-based application. JavaScript, HTML, and CSS are interpreted
languages, and they do not require a compilation step. That dramatically improves the speed of development and debugging process.

The front-end representation of the Apparatus model instances is based on graphs, where each graph is made up as nodes that are connected with edges. Two widely used JavaScript libraries are used to create graph diagrams. Sigma.js [52] and Cytoscape.js [33], both licensed under MIT. The initial version of the software application used Sigma.js but later migrated to Cytoscape.js. Cytoscape.js offered more functionality when analyzing a graph, such as the dynamic application of CSS rules on a graph.

The Electron framework and the Cytoscape.js library fulfill the R1, R2, and R3 requirements.

The architecture of the application is based on modules as stated in R4. Each module will have a specific function. To satisfy R5, the modules will be able to interact with the application using an API. The application will be able to be extended by creating new modules. The primary focus of the application would be to support the Apparatus framework. Due to the modular nature of the application, it would be easy to extend it with modules that will allow the support of other modeling frameworks. The software will not support other modeling frameworks that are not incorporated directly into the Apparatus framework, as stated in R6. To enable the configuration of the environment, to satisfy R7, the values of the color themes and the fonts will be stored in a global configuration file.

3.5.2 ASTo’s architecture

ASTo’s initial page is the home screen. From the home screen, a user can navigate to the engineering phases of security analysis supported by Apparatus. The home screen of the Tool is shown in Fig. 3.7.
Once a user selects a phase, the tool loads the view of that particular phase. Each phase supports different analysis options. Each main view window is divided into three sections. The first section is the Control Tab. In the Control Tab, a user can select the functions of the tool by pressing buttons. Functions of the Control Tab are the addition/deletion of nodes and edges, highlighting specific node groups and validation of the system. The second section of the window is the Graph Tab. In the Graph Tab, the IoT system under analysis is presented as a graph diagram. The Graph Tab supports dynamic user interaction. The third section is the Action Tab. In the Action Tab, any information about the state of the graph is displayed. In the Action Tab, a user can access the command prompt of the application. The command prompt can be used to input search queries for the graphs or type in commands. The Graphical User Interface (GUI) layout of the tool during the design phase is shown in Fig. 3.8 and the GUI layout of the implementation phase is shown in Fig. 3.9.
Fig. 3.8 ASTo design phase GUI
Functionality of ASTo

The tool is developed using a modular approach. Each module contains everything necessary to execute only one aspect of the desired functionality. The packages as shown in Fig. 3.10, are divided into Core packages, Design phase packages, Implementation phase packages, and State packages. The Core packages are shared between the different phases of analysis, while each phase uses unique packages for specific analysis.
Core packages

Core packages provide the majority of the functionality of the Tool. Core packages are divided into two parts. Core application packages and configuration packages. Core application packages are used to perform analysis on the IoT graphs. Functionality that display node information on hover, save, load graphs are part of the core packages. Other core packages that provide generic functions, such as the addition of nodes and edges, are exported to the analysis phases packages. This is done to reduce code complexity and increase modularity since each phase introduces its own logic but uses the same underline code.

Configuration packages

Configuration packages are used to configure the appearance of the tool. The configuration of the application is made through a GUI window. Users can change the appearance and settings of the tool. A user can specify different colors or shapes to the elements of the language. Moreover, users with vision impairments to configure the tool according to their needs and specifications [108].

- **GUI configuration**: every element in the GUI of the application is configurable. A user can change the color values of the elements, the font style, and size or choose to which elements of the GUI to display.
• **Graph configuration:** the style of the graph is configurable. A user can change the size, color, and shape of the nodes. The size, style, and color of the edges, along with the font style, size and color of the graph’s labels can be changed. Users can configure the visual representation of Apparatus notation classes.

• **Tool settings:** allow users to configure other aspects of the tool, such as frequency of backups or the maximum number of notifications or default window size.

**Analysis packages**

The following list provides a short description of the analysis packages that are available both in the design phase, the implementation phase, and the system state and their functionality.

1. **graph manipulation:** the tool supports manipulation of the graph in a graphical manner. The user can move nodes, add nodes and edges and make changes on the properties of the nodes.

2. **highlight nodes:** the desired nodes are highlighted while the rest of the graph has its opacity reduced.

3. **highlight modules:** the nodes that are part of the same metamodel module are highlighted. For example, the engineer can choose to highlight only the network module nodes or the security module nodes.

4. **highlight neighbors:** the neighbors of the selected node are highlighted.

5. **attribute search:** the nodes with the selected attribute are highlighted.

6. **pattern identification:** the nodes that share the identified pattern are highlighted. This functionality is useful when looking for patterns in the graph. For example, an engineer might be interested in all the Devices that use wireless network connection which support the telnet communication protocol. This package implements the Pattern identification algorithm described in Sec. 3.3.

7. **hover node information:** while hovering a node, its properties are displayed in an adjacent container.

8. **model checking:** graphs are validated according to the rules of metamodels asynchronously. The tool disallows the usage of concepts in ways that are not permitted in the metamodels.
9. *layout placement:* the layout of the graph can be configured using placement algorithms. Those algorithms are provided by the Cytoscape library [33].

10. *export/import models:* graphs can be exported or imported as JavaScript Object Notation (JSON) files. The JSON files can be then shared between engineers or imported in other tools for further analysis.

The design phase analysis packages are presented in the following list:

1. *model overview:* the tool displays a quick overview of the model. The information shown is the total number of nodes in the graph, the number of nodes of each concept and the number of nodes in the design phase graph.

2. *Threat verification:* the tool can verify if the identified Threats are mitigated by Constraints. The package displays an overview of the number of Threats of the graph along with the mitigated Threats number. This package implements the Semantic validation of threats algorithm described in Sec. 3.3.

3. *model checking:* graphs are validated according to the rules of metamodels asynchronously. The tool disallows the usage of concepts in ways that are not permitted in the design phase metamodel.

4. *imported model checking:* the tool can check if imported models comply with the rules of the design phase metamodel.

The analysis packages that are available only in the implementation phase are the following:

1. *model overview:* the tool displays a quick overview of the model. The information shown is the total number of nodes in the graph, the number of nodes of each concept and the number of nodes in the implementation phase graph.

2. *Threat verification:* the tool can verify if the identified Threats are mitigated by Constraints. The package displays an overview of the number of Threats of the graph along with the mitigated Threats number. This package implements the Semantic validation of threats algorithm described in Sec. 3.3.

3. *model checking:* graphs are validated according to the rules of metamodels asynchronously. The tool disallows the usage of concepts in ways that are not permitted in the implementation phase metamodel.
4. *imported model checking:* the tool can check if imported models comply with the rules of the implementation phase metamodel.

5. *Vulnerability verification:* this package is only supported on the Implementation phase. The Vulnerability verification package returns an overview of the total Vulnerabilities of the graph and which Vulnerabilities are mitigated.

6. *Vulnerability identification:* using information encoded in the model, the tool generates an array of keywords that are used as search terms in vulnerability databases. The tool sends a search request to the specified database. If the search returns positive results, a Vulnerability node is created connected to the vulnerable device. This package is used to automate the identification of a system’s vulnerabilities based on its attributes. The package implements the Vulnerability identification algorithm described in Sec. 3.3.

7. *model generation from network capture files:* network capture files, such as .pcapng files, have information about a network that can be used to generate **Apparatus** implementation phase models. That information includes the active devices in the network, their network connections, and the running services. Active devices are used to generate the Device concept in the module. Their connections are used to generate the Connection concept, and the running services are used to generate the Application concepts. This package implements the model generation algorithm described in Sec. 3.3.

8. *security insights based on attributes of the model:* information encoded on a model is used to propose security insights based on the algorithm described on Section 3.3.1.


The system state analysis packages are the following:

1. *model overview:* the tool displays a quick overview of the model. The information shown is the total number of nodes in the graph, the number of nodes of each concept and the number of nodes in the system state graph.

2. *model checking:* graphs are validated according to the rules of metamodels asynchronously. The tool disallows the usage of concepts in ways that are not permitted in the system state model schema.
3. **imported model checking**: the tool can check if imported models comply with the rules of the system state model schema.

### 3.6 Proof of Concept Applications

Proof of Concept (PoC) applications of the framework’s components have been presented through the publications (Section 1.5) that have been produced during this research project. The publications offer the evaluation of individual framework components through their application in small-scale examples. The PoC applications present the evolution of the framework throughout the research project, as well as, the reasoning behind each change. An overview of the small-scale evaluation of the framework’s components will be provided in the rest of the section while a discussion of the overall lessons learned will follow in Section 4.2.

#### 3.6.1 First PoC - Initial approach to IoT security modeling

The first application of Apparatus proposed a computer network modeling approach to security analysis that laid the foundation of the security insights algorithm. In the PoC, a temperature monitor application was modeled using the metamodel shown in Fig. 3.11. The model mapped into a structured JSON schema that was used to elicit security requirements.

![Fig. 3.11 First metamodel of Apparatus](image-url)
The IoT system that was analyzes consisted of the following: a (1) temperature sensor; a (2) router; a (3) server, that would function as a database and a data formatter for the database; and a (4) laptop. The temperature sensor would collect environmental data and would send them to the server through the router. The server would format the data and then stored them in its database. The laptop was used to request the needed information from the database. The JSON representation of the system is shown in Fig.3.12
Fig. 3.12 JSON representation of the PoC
The final security analysis was made using the information of the JSON file. The resulting analysis is shown in Tab.3.8.

**Table 3.8 Security requirements elicited from PoC properties**

<table>
<thead>
<tr>
<th>Security requirements</th>
<th>IoT system properties (line)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes should be physically protected</td>
<td>layer: perception (11, 112)</td>
</tr>
<tr>
<td></td>
<td>connection: wireless (19, 119)</td>
</tr>
<tr>
<td></td>
<td>user/deployer id:01 (all nodes)</td>
</tr>
<tr>
<td></td>
<td>developer/manufacturer id:02 (all nodes)</td>
</tr>
<tr>
<td>System can only be used by authorized users and devices</td>
<td>connection: wireless (19, 119)</td>
</tr>
<tr>
<td>Environmental data should not tamper with the system</td>
<td>input: data_environmental (14)</td>
</tr>
<tr>
<td>Database data should only be modified by authorized users or devices</td>
<td>type: database (74)</td>
</tr>
<tr>
<td>Input of the database should subject to sanitation</td>
<td>type: database (74)</td>
</tr>
<tr>
<td>Nodes must keep chronological records of any activity that affects operations, procedures or events of the system</td>
<td>Default security requirement of IoT systems</td>
</tr>
</tbody>
</table>

The PoC was an initial approach to IoT modeling that was used to identify the characteristics and requirements of IoT systems. Nevertheless, it laid the foundations of the framework were each succeeding iteration refined. The central premise of the PoC was that different IoT systems share a set of characteristics and requirements. That set of characteristics and requirements can be used to define a baseline security approach to IoT systems. The resulting baseline security approach would be independent of the requirements of the stakeholders. Instead, it would be based on the hardware and software architecture of the IoT system.

The Fig.3.11 was the basis of the implementation phase metamodel of the language. The IoT node (later renamed to Device), the Network Connection (later renamed Connection) and the Micronet were the building blocks of the network module of the language. The approach to baseline security using information of a model was refined to the Security Insights automated analysis process.
3.6.2 Second PoC - Implementation phase analysis

The second application of APPARATUS proposed the modular implementation phase metamodel. Concepts of the metamodel were divided to network, security, and social modules, as shown is Fig.3.13.

![Fig. 3.13 PoC 2 - Implementation phase metamodel](image)

The metamodel was used to model a smart home, as shown in Fig.3.14, that had the following devices: 1) Laptop, 2) router, and 3) monitor camera. Once the model was created, the vulnerabilities of the system were identified based on the attributes of the components of the system.
The security analysis was made on each component. A part of the study is shown in Fig. 3.15. The security requirements of the stakeholders regarding the camera was “Camera feed from the baby monitor should only be accessible by authorized users.”. To satisfy the requirement, an authorization Constraint on the users was imposed on the users of the system. The camera was a Philips in Sight B120/37 which is used to monitors infants. The Asset was the Data that was being transmitted by the camera. That particular model is vulnerable to a direct browsing vulnerability, CVE-2015-2884\(^1\). The camera would create a remote viewing stream by a proxy connection to the camera’s internal web service via the cloud provider. The stream was bound to a public hostname and port number without any access control mechanisms. An attacker could locate the hostname and port number and access the camera feed. To mitigate the vulnerability, whitelist rules should be created with only the allowed users. The intent of the Malicious Actor was to *view the camera feed*.

\(^1\)http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2884
The PoC 2 metamodel shares similarities with the final version of the metamodel. The significant issue of that metamodel was that software applications could not be modeled in a concise manner. Software applications were modeled as the service attribute of the Device concept. If a Device had more than one applications running, each application had to be modeled as a separate Device node. This led to complicated models with many similar components, that complicated the security analysis. For that reason, the concept of Application was introduced to the metamodel.

Additionally, the concept of Unidentified node was removed from the metamodel. The Unidentified node was used to model a Device that was not directly connected to a Micronet, and a security engineer has little knowledge of its requirements and behavior. Unidentified nodes could only have a relationship with the concept of Net. As a result, a model could not convey any information on the communicating between Devices and Unidentified Nodes. Information on the flow of communication is vital to assess the attack surface of a network. To mitigate that issue the concept of the Unidentified node was incorporated into the concept of Device. Devices can have relationships to the concepts of Connection, Net, Micronet and Application, which result in more accurate representations of IoT systems.

3.6.3 Third PoC - Security analysis of a smart city

The third application of the framework was a security analysis of a smart city. In the third PoC, the design phase metamodel, the phase transition algorithms, and the dynamic notation were introduced. One of the aims of the application was to evaluate
the scalability of the framework while modeling the transport infrastructure of a smart city. The network components of the smart city are shown in Fig. 3.16.

![Network Diagram](image)

Fig. 3.16 Design phase network module of the transport system

The security analysis initiated from the design phase, following the security requirements of the stakeholders. The stakeholders provide a list of security requirements and assets of the system. Based on the requirements and assets, the constraints and threats to the system are derived.

1. **SR1**: passengers should not be able to tamper with the bus’s hardware systems.

2. **SR2**: passengers personal data should not be able to be exposed without authorization through the bus’s resources.

3. **A1**: the physical aspect of the bus’s microcomputer.

4. **A2**: the personal information of the passengers.

5. **T1**: physically attack the microcomputer to perform a DoS attack.
6. **T2**: exploit the smart transport application to obtain passengers’ personal data through social engineering.

7. **C1**: physically protect the microcomputer to mitigate T1.

8. **C2**: smart transport application must notify the user every time request for his personal information is made from an untrusted source, to mitigate T2.

The attributes mentioned above are incorporated in the model of the system to facilitate the security analysis, as shown in Fig.3.17.

![Design phase model of the transport system]

Fig. 3.17 Design phase model of the transport system

To identify the vulnerabilities and mechanisms of the system, the model needs to be transitioned to the implementation. To do that, the design-to-implementation transition algorithm is applied to the model, as shown in Fig.3.18. The additional information on the implementation phase enables the identification of security issues that were not apparent in the design phase. For example, the layer attribute of the Device concept can be used to infer both the conceptual use of the Device and its inherited physical location. Devices belonging to the application layer are usually processing servers either are located in the cloud or a physically secure area. Devices that belong to the perception layer are typically found in close proximity to actors. They represent sensors and end-point devices. A Constraint that is proposed based on that knowledge
3.7 Summary

is that all Devices of the system that have the layer: perception attribute value, should be physically protected. Devices belonging to the perception layer of IoT offer physical access to both actors and malicious actors. They are vulnerable to threats that require physical access.

This chapter starts by introducing the APPARATUS security framework. The APPARATUS Framework is used to design and analyze IoT systems. The Framework is defined with a set of components. Different components of the Framework will be used by the security engineers to model and analyze specific aspects of an IoT system. The Framework consists of the following components: a modeling language, a modeling procedure, and a set of analysis procedures. Then, the supporting software application of the Framework ASTo is presented. ASTo is network security analysis and visualization tool, which automates, and semi-automates certain functions of the Framework. Examples of automated functions are the generation of implementation phase models from network capture files, and the transition of models from one engineering phase to the other. The chapter ends with the presentation of three proof of concepts applications of the Framework. The proof of concepts, refined aspects of the Framework through its application.
Chapter 4

Evaluation

The evaluation of the APPARATUS framework followed an iterative “build and evaluate” approach throughout its development. Each language module and framework process has been applied to proof of concept case studies. Proof of Concept applications, presented in Section 3.6, facilitated the refinement of each component of the framework. ASTo was evaluated through its application to projects from the Open Source security community and the University of Applied Sciences of Piraeus, as presented in Section 4.2. During the later stages of the research, the overall framework was evaluated through its application to a case study, as shown in Section 4.3. The APPARATUS framework was applied to a real-life Security Company to assess the security posture of its infrastructure and propose improvements. Both quantitative and qualitative insights from the large-scale framework application through the case study were collected via previously defined metrics and stakeholder interviews. The rest of this chapter presents the different evaluation efforts undertaken as part of this research project and concludes with a discussion regarding the lessons learned from such attempts.

4.1 Case Study

Case studies are a common approach to empirical evaluation in the field of information systems research [83]. The objective of present the case study is to identify whether the use of the APPARATUS framework can facilitate the creation of models that describe real-life IoT systems and facilitate the security analysis of such systems. While individual components of the framework have been applied to small-scale examples throughout the development process (Section 4.1), a large-scale empirical evaluation can provide unique insights regarding its overall applicability and effectiveness.
4.1.1 Case Study Process

The process of designing and executing a case study involved five basic steps, according to [97].

- **Case study design**, where objectives are defined, and the case study is planned. In this case, the overall objective of the case study is to identify whether the developed framework can produce secure business process designs when applied to a real life information system. The selected system and the stakeholders involved in this case study are discussed in the next section.

- **Preparation for Data Collection**, where the data collection procedures are defined. In our case data is collected during the application of the framework’s component of the studied system. This is performed in close cooperation with some of the system’s stakeholders following a specific set of steps for the application of the developed framework. In addition to that, some quantitative metrics are also defined to provide us with conclusions regarding the framework’s effectiveness, as presented in the next section.

- **Collecting Evidence**, where data is collected from the studied system during the execution of the case study. For the case study presented in this work, this step involves the application of our framework to the studied system for the creation of different system models, as presented in Section 4.3.2.

- **Analysis of Collected Data**, where the data is analyzed for the extraction of conclusions. In this case study, this step includes a qualitative evaluation of the framework’s application through a semi-structured interview with the involved stakeholders, as well as the evaluation of certain quantitative metrics.

- **Reporting**, where the results of the case study are summarized to draw conclusions. In our case, the reporting consists of a brief discussion of the main points raised by the stakeholders during their exit interview and the results of the metrics evaluation, as presented in Section 4.3.5.

4.1.2 Case Study Design

The case study selected for the application of the framework involved a technical company. The name of the company and certain results of the case study are under a confidentiality agreement between the author and the company. The company’s primary
source of revenue is the installation and management of physical security systems as well as computer networks. The company has IoT centric network infrastructure and devices. They use a mixture of Cloud and in-house services for day to day operations. The case study aimed to evaluate the analysis processes of the framework. The analysis processes are used to assess the current security policies and infrastructure of the company, identify security issues and propose security mechanisms.

The case study was developed and performed in close cooperation with two engineers working at the company. The engineers acted as the stakeholders of the system, while the author acted as the security engineer that applied the Framework. The first engineer was responsible for physical security systems operations, while the second was responsible for the computer networks operations. None of the engineers were familiar with the APPARATUS Framework or any other similar security modeling framework, before the application of the case study. The engineers contributed knowledge of the system based on their expertise. The steps followed to elicit information about the system and apply the Framework, are as follows:

1. An initial discussion was held with the stakeholders to provide them with a high-level overview of the Framework, explain the goals of the case study and initiate communications.

2. A high-level description of the studied system was provided by the stakeholders via teleconferencing, providing details about the infrastructure, soft and hard assets and actors of the system.

3. A number of network captures of the system were provided by the stakeholders to generate APPARATUS models.

4. An initial set of APPARATUS models were created and submitted to the stakeholders for feedback.

5. The models were refined according to the received feedback, until an accurate system representation is captured, as per the stakeholders’ instructions.

6. The security requirements of the system are elicited after communication with the stakeholders. The assets of the system are identified and the APPARATUS models are updated accordingly.

7. The threats that target the assets of the system were identified. Then the vulnerabilities of the system are enumerated based on information on the implementation phase models.
8. Based on the identified threats, constraints that protect them against are proposed. Specific mechanisms that mitigate the vulnerabilities are proposed and implemented in cooperation with the stakeholders.

9. Generate a security analysis report using the ASTo utility to share with it with the stakeholders for evaluation.

The data collected through the use of the framework are then analyzed both qualitatively and quantitatively. An exit interview with the involved stakeholders of the system provided qualitative insights regarding the perceived applicability and effectiveness of the framework. Additionally, a series of values for metrics was calculated to provide quantitative insights regarding the framework’s performance in this case study. For that reason, the case study is not an exhaustive security analysis.

More specifically, the quantitative metrics, which is calculated at the end of the case study, measures the automated and semi-automated functions of the framework. The functions that are being evaluated are: 1) model generation through network capture files; 2) model-based vulnerability identification and 3) model-based security insights. In more detail, the specified metrics are the following:

- **Function’s intention:** is used to determine whether the function achieve their intent. The evaluation metric is a Boolean value. The effectiveness of the function is evaluated using the metrics below.
- **Function’s required resources:** is used to evaluate the number of the resources needed to achieve the function’s intention in comparison to a human undertaking the same process. Resources, in this case, are work hours that translate to financial costs saved by the application of the function.
- **Function’s effectiveness:** is used to evaluate the effectiveness of the function. What can be done with the function that could not be done before? Automated functions were assessed regarding their correctness and error rate.

### 4.1.3 Case Study Privacy and Security considerations

Security assessment on real-life systems requires sensitive information of the system and its stakeholders. It is common practice to bind security engineers with agreements that prevent them from revealing such information to others. Such information can be used by malicious actors to compromise the system. For example, knowledge of specific hardware models or device firmware can be used to develop custom-made exploits and
4.1.4 Framework Application

Over the rest of this section, the application of the framework to the infrastructure of the company under analysis will be described in full detail, along with the produced intermediate and final modeling outputs. For the remainder of this section, the company under analysis will be referred to as the Company.

System description

The Company’s primary source of revenue is the installation and trading of physical security systems, computer networks and the remote monitoring of security installations. The remote monitoring of security installation is made through referencing of clients to a third-party security monitoring company.

The infrastructure of the Company has both physical and software assets. Due to security considerations, specific information of the infrastructure has been omitted. This limits the scope of the case study to a particular set of assets and information. In certain parts of the Case study instead of using the specific software versions or hardware models, generic versions and names are used. Such information can be used by a malicious actor to target the infrastructure of the company and develop specific exploits.

The Company is using a Main Office for its operations. Those operations include day to day administration, storing of merchandise and equipment and meeting with clients. The majority of the equipment is used by the employees while on the field. The only stationary hard assets are located in the Main Office. Due to the privacy and security considerations of the stakeholders, the actual models of the hardware are omitted. Instead, the name of the brand is used in the case study. That information, if made public can be used to compromise the systems of the Company. Malicious actors will be able to target the system with bespoke exploits. The Main Office is equipped with a basic alarm system to secure windows and other entrances. The alarm system
Evaluation

is installed and monitored by a third-party company that oversees all the other offices in the area.

The Company’s employees use the following equipment:

1. 1 Desktop Dell, Windows 7, Main Office

2. 1 Laptop Turbo X, Windows 10, Main Office (IP 192.168.0.12)

3. 1 Laptop Acer, Windows 7, mobile (IP 192.168.0.13)

4. 1 Laptop Dell, Windows 10, mobile (IP 192.168.0.13)

5. 1 iPhone 5s, iOS 11.2.3, mobile (IP 192.168.0.20)

6. 1 iPhone 7, iOS 11.2.3, mobile (IP 192.168.0.21)

7. 1 iPad, iOS 11.2.3, mobile (IP 192.168.0.22)

8. 1 HP Printer, Main Office

9. 1 Router ASUS, Main Office (IP 192.168.0.1)

The Company’s employees use certain services to perform business operations such as email. Specifically, the services that are used are the following:

1. iCloud and Google Drive

2. Google G Suite

3. Apple’s Messages

4. Apple’s Facetime

5. Apple’s Notes

6. Apple’s mobile Safari

7. Chrome 64
Main Office

The Main Office is used for administrative purposes, and non-authorized personnel does not have access to the premises. The first stationary machine, the Desktop Dell PC, is used for running the accounting tools. The Desktop is not connected to a network port and is located in a locked compartment. Access to that compartment is only available to the Company’s lead engineer. The second stationary machine is the Laptop Turbo X. That Laptop is the workstation of the lead engineer. The Office is used for meetings with clients and other non-administrative duties, as well as the storing of equipment and merchandise. Both engineers of the Company have physical access to the Office.

Model generation thought network capture files

To create the initial system, an interview via teleconferencing was conducted with the engineers of the Company. The engineers offered a high-level overview of the hardware and software of the system. The contents of the system that were derived from the interview are the ones listed above.

During the interview, the engineers were asked to capture network traffic from their daily operations. The purpose of the network captures was to generate implementation phase models. The Company requested not to publish the network captures files and delete the files once the case study has been completed. Furthermore, it was requested not to release the initially generated models without the models be reviewed by the engineers. The author agreed to the terms.

The engineers provided the author with six different network capture files from the Office at different times of the day. The files were in pcap-ng format. The six files were concatenated into a single file. Then, the file was imported into ASTo to generate the implementation system model. The resulting APPARATUS model was sent to the engineers for review. During the review, the engineers noted which constructs of the model should be removed due to security considerations. The constructs that were not removed from the model were of two types. The first type were the ones that were in the IP range of the Company’s network. The second type were the ones that had the IP addresses of the main services that were used by the Company, such as Google and iCloud. The resulting system model after the review process is shown in Fig.4.1 and Fig.4.2. During the review process, devices and applications that were not directly related to the security analysis were removed.
Fig. 4.1 System model derived from network capture files (concepts)
The Fig. 4.1 all the components of the system as described by the engineers are displayed, besides the desktop DELL and HP Printer which are not connected the network. The model shows two devices in the Main Office IP network range, which have not been provided in the initial description of the system by the engineers. The devices had the IP addresses of 192.168.0.15 and 192.168.0.16. After consolation with the engineers, they identified the devices using the IP addresses of the unidentified devices that were on the model. One device was a Wi-Fi repeater, and the other device was an alarm system. Both of those devices were located in the network of the Main Office. The devices were not in the description of the system because they were not included in the devices’ manifest of the Company. Another finding shows the router of
the Office making networks requests to an IP address that was not of the provided list of services used by the engineers. The IP of the device was the 204.74.99.100. Utilizing the `nslookup` utility that queries Domain Name System (DNS) for DNS records, the IP was determined to belong to an alarm security company service. Further investigation with the engineers revealed that the alarm system of the Office was connected to the Internet as well as the land-line telephone system. The Internet access was installed by the third-party security monitor company to allow remote control of the alarm of the office through a mobile application for the engineers.

The process of model creation through network capture files shows an additional benefit, besides reduction in resources. It can be used to create more accurate system models to their real-life counterparts, than solely relying on the stakeholders.

After consultation with the stakeholders, the concepts of the model were mapped to their counterparts, and the rest of the attribute fields are filled. The devices that are part of the Main Office compose the Micronet of the system. The third-party devices (iCloud, GSuite, and Caddx web servers) are part of the Net. The Fig. 4.3

![Fig. 4.3 System model with the network components (concept description)]
Model-based vulnerability identification

In a real-life application of the framework, the analysis would be performed on a single model using ASTo. For the purposes of the thesis, where there are constraints on the visualization options and medium, the system is broken down into smaller models that easier to showcase in a paper format.

One of the leading security concerns of the stakeholders is the physical theft of the stored merchandise and equipment. To represent that security concern the stakeholders provide the following security requirement, “Physical alarm system should not be compromised by remote attacks.” A plausible attack scenario, since the alarm system is networked, is a malicious actor remotely compromising the alarm system to gain access to the premises. The security requirements translate to a Constraint concept that is imposed on the Micronet. That Constraint is the basis of the threat analysis of the system. The two Devices of the Micronet, the router, and the alarm system, are the ones that will be targeted by a malicious actor. Those Devices are Assets that will be targeted by the identified Threats by a malicious actor.

The finalized list of security requirements that are provided by the stakeholders are the following:

- Physical alarm system should not be compromised by remote attacks.
- Devices should be protected against remote attacks.
- Data in transit and data at rest should be protected against tampering attacks.
- Engineers should be protected against social engineering attacks.
- Accounts used for services such as email and office utilities should be protected.

The assets of the system are the following:

- **A1**: Network services provided by the router.
- **A2**: Physical Devices and equipment.
- **A3**: Remote access of the alarm system.
- **A4**: Wi-Fi password.

Based on the security requirements of the stakeholders the following constraints are proposed:
**C1**: Devices and their applications should only be used by authorized personnel with the least required privileges. If a process is compromised, the attacker has the same privileges in the system as that process. By using the least privileges approach the amount of access an attacker has in the system is limited.

**C2**: Devices should either be logged off or powered down when not in use. If a device is not active, it cannot be used for malicious purposes.

**C3**: Incoming communications must be reviewed before being accepted. The main attack vector of social engineering attacks are emails and chat applications. They are rottenly used to deliver malicious links and executable. Companies are susceptible to such attacks since they commonly communicate with people that they are not familiar with, such as first-time clients.

**C4**: Inspect outgoing traffic for blacklisted destinations. In the event of a successful attack, the compromised system will aim to establish a communication channel between it and a command center. If outgoing traffic is inspected, malicious destinations can be identified and blocked.

**C5**: Downloaded files must be reviewed in a sandbox. A common attack with social engineering attacks is to navigate a user to towards a malicious website. Once on that site, a malicious executable in the form of a plugin or extension is automatically downloaded. Despite the download of the malicious code, for the system to be compromised, the user still needs to execute the executable. If downloaded files are executed first in a safe environment, they can be reviewed without consequences.

**C5**: Authorized personnel must be notified when a service is being used. When a serviced is used, such as access to an email client or a workstation, the authorized personnel needs to be notified. If the authorized person wasn’t the one that used the service, that means that the account is compromised.

**C6**: Only authorized personnel should be able to access the alarm system remotely.

**C7**: The Wi-Fi network should only be used by the stakeholders. Non-stakeholders should use the wireless network setup for visitors.

**Threat 1**: Remote compromise of the alarm system. The router is exposed to the public Internet through the IP provided by the ISP. Any number of ways can be used to obtain the public IP address of the router. For example, an attacker can use a
service such as Shodan\(^1\), which scans devices that are connected to the Internet. Or use a custom written application that transposes the public Internet to document devices that advertise as a router. Since the IP of the router is public, it is assumed that malicious actors can obtain it. An example of the attacks steps a malicious actor can perform to compromise both Assets is the following:

1. The malicious actor uses an exploitation framework, such as the Metasploit\(^2\) framework, to deploy their attack. The Metasploit framework is penetration testing platform that enables the finding, exploitation, and validation of vulnerabilities.

2. The malicious actor enumerates the vulnerabilities of the router. For this step, the malicious actor could deploy Metasploit’s automated scanning and vulnerability enumeration scripts.

3. Once a vulnerability is identified, the malicious actor would need to use an exploit of the vulnerability. The attacker could develop an exploit or use an existing exploit. Once the exploit is identified, it will be deployed with a reverse shell payload. The reverse shell payload will enable the attacker to control the router remotely. Once the exploit is delivered to the router, the payload will be executed, and the router will be compromised.

4. The malicious actor has control of the router over the remote shell connection. The attacker will be able to leverage the router’s network enumeration functions to identify the alarm system in the local network.

5. The malicious actor performs the same process of vulnerability enumeration to the alarm system as the one performed to identify the router’s vulnerabilities.

6. Based on the identified vulnerabilities the malicious actor will develop or find a specific exploit for the alarm system. The malicious actor will deploy the same reverse payload along with an exploit targeting the alarm system. Once the payload is executed, the malicious actor will be able to control the alarm remotely.

7. The malicious actor will physically approach the premises of the Secondary Office and disable the compromised alarm system. The malicious actor will then attempt to steal the merchandise.

\(^1\)https://shodan.io
\(^2\)https://metasploit.com
Evaluation

**Threat 2: Compromise of the password of the wireless network.** The malicious actor aims to log in the network as a legitimate client. The wireless network uses the Wi-Fi Protected Access II (WPA2) security protocol. The process of obtaining the password of a WPA2 network is by obtaining the four-way *handshake* during the authentication of a client. The four-way handshake is used to establish the PTK (Pairwise Transient Key). Then a brute force attack is applied on the *handshake* to reveal the process. The chances of success depend on the uniqueness and strength of the password as well as the processing power at the hands of the attacker. To obtain the *handshake*, the attacker needs to be in the area of the wireless network. To obtain the password, a malicious actor can do the following:

1. Passively captures the communication of the wireless network using a wireless security testing framework such as Aircrack-ng\(^3\). The attacker is only interested in the wireless management frames of the communication, specifically the authentication beacons and probes. The wireless management frames are not encrypted in WPA2 and are subject to Spoofing attacks [48]. Through the use of management frames, tools can detect which data frame to capture.

2. The handshake is encrypted through the cryptographic PBKDF2-SHA1 hash function. To derive the password from the hash, the attacker performed a brute force type of attack, known as the rainbow table. A rainbow table is a precomputed table for reversing cryptographic hash functions. If the hash of the password exists in the table, the attack will match it and reveal the password of the network.

An issue with that type of attack is that it is opportunistic. For the attacker to remain undetected, a legitimate connection with a device needs to take place. If the attacker forces an authorized device to reconnect using a deauthentication request, it increases the risk of detection from the security mechanisms of the system. Since the Secondary Office is not always staffed, a malicious actor needs to wait until one of the security engineers enters the premises.

**Threat 3: Remote compromise of a device or application.** An attack vector that malicious actors can compromise a device or application by exploiting a vulnerable network resource, such a network database or a web server. Another attack vector is to execute arbitrary code on the target device either through social engineering, a compromised application or by physically deploying the code. As a result, such a

\(^3\)http://aircrack-ng.org
threat has multiple attack paths that need to be mitigated. However, the goal of the attacker remains the same. The attacker wants to be able to execute system level operations remotely. An example of such a path is the following:

1. A malicious actor uses the publicly available information to identify company email accounts.

2. The malicious actor constructs an email either posing a potential client or a business associate. In that email, the attacker adds an attachment with a reverse connection payload.

3. The malicious attachment once executed, will start a reverse connection to another device operated by the malicious actor.

4. At this stage, the device is compromised. The reverse connection will allow the malicious actor to remotely execute system level operations as if a legitimate actor was performing them.

Once the threats of the system have identified, the next step is to determine the vulnerabilities that can be exploited. For greater clarity, the vulnerability identification step is broken into groups of devices. The first group of devices is the ASUS router and the Caddx alarm system. The second group of devices is the iPhone 5s, iPhone 7 and iPad. The third group of devices is the desktop Dell (Windows 7), laptop Acer (Windows 7) and HP Printer. The fourth and final group of devices are the laptop Dell (Windows 10) and laptop Turbo X (Windows 10).

**Group 1: ASUS router, Caddx alarm system.** The first device group that is being analyzed is the Device ASUS router, Device Caddx alarm system, and their corresponding Applications. As shown in the model 4.4, those components have certain properties. Those properties are used by the Vulnerability Identification algorithm of the APPARATUS Framework to automatically generate a list of vulnerabilities. The algorithm uses the actual hardware and firmware models of the router and the alarm system, which are omitted from the model. By executing that algorithm, the following vulnerabilities are identified from the CVE public API:

1. CVE-2017-14698 on the ASUS router, which allows remote attackers to change passwords of arbitrary users via the `http_passwd` parameter to `mod_login.asp`.

---

4[https://cve.circl.lu/api/search/](https://cve.circl.lu/api/search/)
2. CVE-2017-14699\(^6\) on the ASUS router, which allows remote authenticated users to read arbitrary files via a crafted DTD in (1) an UPDATEACCOUNT or (2) a PROPFIND request.

The identified vulnerabilities affect a family of ASUS router models, part of which is router used in the Office. No vulnerabilities were identified that could exploit the Caddx alarm system.

The CVE-2017-14698 can be exploited by an attacker, without the need for them to be authenticated to the network. The \textit{http passwd} parameter is part of an HTTP POST request that can be sent to the router. By modifying the parameter attackers can change the administrator's password. After the password is changed, attackers can log in into the router as administrators. For the attack to be successful attackers are not required to be in close physical proximity to the network. To mitigate the CVE-2017-14698 vulnerability an update to the firmware of the router is required.

The CVE-2017-14699, an attacker, needs to be authenticated to the network. That requires an additional step of obtaining the password for the wireless network. If the malicious actor is assumed to have acquired the password of the wireless network, the vulnerability can be exploited. For the exploit to be successful, the AiCloud service, which operates on port 449, need to be enabled on the router. The AiCloud is a service from ASUS that allows users that have installed it, to access their files from remote locations. That service is not enabled on the router of the case study, and as a result, the vulnerability is not exploitable. As a result, the CVE-2017-14699 is not included in the model.

The resulting security of Group 1 is shown in Fig. 4.4.

\(^6\)https://nvd.nist.gov/vuln/detail/CVE-2017-14699
4.1 Case Study

The next group of devices that are analyzed is the Apple devices that use the same software. The devices are the iPhone 5s, iPhone 7 and iPad using iOS 11.2.3. The attributes of the devices and applications are shown in Fig. 4.5.
Using the model in Fig. 4.5 as an input, the vulnerabilities of the system are enumerated. The vulnerability identification algorithm used in ASTo identifies 1932 vulnerabilities. The vulnerabilities and their related information are stored on a structured JSON file. The majority of those vulnerabilities are not related to the system or do not directly impact its security. For example, a number of the identified vulnerabilities refer to the exploitation of Cisco IOS\textsuperscript{7}, which is not related to Apple’s iOS operating system. Certain vulnerabilities exploit software bugs to cause iOS devices not to operate normally. Such attacks can render an application, such as Safari to crash, but otherwise cannot compromise the device. Other vulnerabilities only affect specific third-party applications that are not installed on the iOS devices of the system. The next step is to identify which vulnerabilities are relevant. ASTo does not support a utility to parse the vulnerabilities of the resulting JSON file. For that task, a dedicated Open Source utility, named VulnWhisperer\textsuperscript{8}, is used. The VulnWhisperer is a utility to parse JSON files of vulnerabilities only to identify the ones that are relevant to the system. The utility uses a parser with flexible keywords and ElasticSearch queries as inputs. To comply with the requirements of the stakeholders, the vulnerabilities

\textsuperscript{8}https://github.com/austin-taylor/VulnWhisperer
of interest are the ones that can be exploited to execute arbitrary code remotely. Specifically, the search terms are the following:

- iOS 11.2.3
- NOT tvOS, macOS, watchOS
- remote arbitrary code execution
- Apple iOS applications
- NOT Cisco IOS

After using the utility, the resulting relevant vulnerabilities were the following:

- CVE-2018-4085

The CVE-2018-4085 targets a rendering component in the WebKit web browser engine called QuartzCore. QuartzCore is used for rendering animations in web content. In iOS, WebKit is used in Safari, Mail, Music, Messages and any other application that renders HTML content. To exploit the vulnerability, a malicious actor needs to create a specially crafted website and inject the exploit on an element on the website. Then, a legitimate user needs to navigate to that website with some form of social engineering attack. To deploy the payload of the exploit, the user must interact with the malicious element of the website. Otherwise, the malicious script will not be executed. The aim of the payload, in this case, is to either download and install a rootkit or upload sensitive information back to the attacker.

The iOS security model benefits from existing mechanisms that prevent such attacks from escalating on the system. Specifically, according to the iOS security guide those mechanisms are: 1) App Sandbox; Each application is operating in a separate sandbox environment without access to system level resources and data. If an application was to be compromised, an attacker could not escalate their privileges to exploit the system further, as the attack would be contained within the application container. 2) Data protection class. The user would need to allow access to sensitive data on their device explicitly. Otherwise, that data cannot be available to the application, even if that application is compromised. The model and the security components of the analysis are shown in Fig. 4.6.

---

Group 3: desktop Dell, laptop Acer, HP Printer. The third group of devices has different requirements from the rest of the equipment and a bound by a different policy. The desktop Dell is only used for the accounting purposes. The only third-party application that is installed is the accounting program. The desktop Dell is not connected to the Company network. It is only connected to the HP Printer. Those two devices are part of an air-gapped network and are never connected to the rest of the network. The devices are located in a locked room, and only one of the engineers has access to it. The laptop Acer is only used when the engineers need to maintain a specific version of an alarm system model. The laptop is never connected to any network and has its network interfaces disabled. The laptop is located in the same secure room as the Dell and HP Printer.

The desktop Dell and laptop Acer use the same version of Windows 7. The HP Printer is an Officejet Pro model. The model of the system and the attributes of the concepts are shown in Fig. 4.7
The vulnerability identification algorithm is performed using the attributes of the model as an input. The result is 752 vulnerabilities. The 751 affects the Windows machines and 1 vulnerability affects the HP printer.

The parse through the Windows vulnerabilities the VulnWhisperer is used. As in the previous security analysis, the only vulnerabilities of interest are the ones that can remotely execute arbitrary code. The search terms that are used are:

- remote arbitrary code execution
- Windows 7
- NOT Microsoft Office

The resulting vulnerabilities of interest are 37. The mitigation mechanism of those vulnerabilities is to apply the security updates of Windows 7. The vulnerabilities of the Windows devices, due to their large number, are grouped into a single vulnerability
concept named *Windows 7 vulnerabilities*. The full list of the vulnerabilities is shown Tab.4.1 and detailed explanation can be found in this link\(^\text{11}\).

<table>
<thead>
<tr>
<th>No.</th>
<th>CVE</th>
<th>No.</th>
<th>CVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CVE-2017-11885</td>
<td>20</td>
<td>CVE-2017-8528</td>
</tr>
<tr>
<td>2</td>
<td>CVE-2017-11819</td>
<td>21</td>
<td>CVE-2017-8527</td>
</tr>
<tr>
<td>3</td>
<td>CVE-2017-11780</td>
<td>22</td>
<td>CVE-2017-8464</td>
</tr>
<tr>
<td>4</td>
<td>CVE-2017-11771</td>
<td>23</td>
<td>CVE-2017-8463</td>
</tr>
<tr>
<td>5</td>
<td>CVE-2017-11763</td>
<td>24</td>
<td>CVE-2017-0294</td>
</tr>
<tr>
<td>6</td>
<td>CVE-2017-11762</td>
<td>25</td>
<td>CVE-2017-0283</td>
</tr>
<tr>
<td>7</td>
<td>CVE-2017-8727</td>
<td>26</td>
<td>CVE-2017-0279</td>
</tr>
<tr>
<td>8</td>
<td>CVE-2017-8718</td>
<td>27</td>
<td>CVE-2017-0278</td>
</tr>
<tr>
<td>9</td>
<td>CVE-2017-8717</td>
<td>28</td>
<td>CVE-2017-0277</td>
</tr>
<tr>
<td>10</td>
<td>CVE-2017-8699</td>
<td>29</td>
<td>CVE-2017-0272</td>
</tr>
<tr>
<td>11</td>
<td>CVE-2017-8696</td>
<td>30</td>
<td>CVE-2017-0260</td>
</tr>
<tr>
<td>12</td>
<td>CVE-2017-8691</td>
<td>31</td>
<td>CVE-2017-0250</td>
</tr>
<tr>
<td>13</td>
<td>CVE-2017-8683</td>
<td>32</td>
<td>CVE-2017-0199</td>
</tr>
<tr>
<td>14</td>
<td>CVE-2017-8682</td>
<td>33</td>
<td>CVE-2017-0161</td>
</tr>
<tr>
<td>15</td>
<td>CVE-2017-8620</td>
<td>34</td>
<td>CVE-2017-0109</td>
</tr>
<tr>
<td>16</td>
<td>CVE-2017-8589</td>
<td>35</td>
<td>CVE-2017-0108</td>
</tr>
<tr>
<td>17</td>
<td>CVE-2017-8588</td>
<td>36</td>
<td>CVE-2017-0090</td>
</tr>
<tr>
<td>18</td>
<td>CVE-2017-8565</td>
<td>37</td>
<td>CVE-2017-0089</td>
</tr>
<tr>
<td>19</td>
<td>CVE-2017-8543</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The vulnerability of the HP Printer is the *CVE-2013-4845*\(^\text{12}\). The CVE-2013-4845 is Cross-Site Scripting (XSS) that requires a user to interact with a malicious web page. If the attack is successful, the attacker can modify the content of the document that is being printed. Since the HP Printer is located in an air-gapped network, an attacker has no way of deploying an attack vector. As a result, that vulnerability is omitted from the analysis.

The vulnerabilities, as well as the mechanism, of the model, are shown in Fig.4.8.

\(^{12}\)https://nvd.nist.gov/vuln/detail/CVE-2013-4845
4.1 Case Study

The laptops Dell and Turbo X are the workstations of the engineers. They are used for day-to-day administration activities such as word processing, appointments, emails, and task management. The office suite that is used for the administration activities is the web-based G Suite\textsuperscript{13} of Google. The applications of G Suite are web applications that are being accessed through the Chrome 64 browser.

\textsuperscript{13}https://gsuite.google.com
The model of Group 4 is shown in Fig. 4.9. The few concepts of devices and applications are connected to the concept of Net. The Net, in this case, represents the Internet. The majority of the applications used by the laptops are accessed through a third-party service, such as Google.

Executing the vulnerability identification algorithm on the model, a total of 834 vulnerabilities are found that can affect Windows 10 and the Chrome 64 browser. In the output of the vulnerabilities file, there are no vulnerabilities that can affect the G Suite. That is to be expected since G Suite is a web application. Web applications always have the most recent version of the applications since the application is fetched by the server every it is loaded on a web browser. A vendor can resolve any known issues of applications and directly deploy them on the client. Furthermore, the web applications operate in the sandbox environment of the web browser, with limited access to the file system.

The VulnWhisperer is used to parse the vulnerabilities file. The vulnerabilities of interest are the ones that can be used for remote arbitrary code execution in Windows 10 and Chrome 64. The vulnerabilities that affect Microsoft Office, as well as those that affect Chrome extensions, are excluded from the model. The keywords that were used as input are:
4.1 Case Study

- remote arbitrary code execution
- Windows 10
- NOT Microsoft Office
- Chrome 64
- NOT Chrome extension

The vulnerabilities that can be used for remote arbitrary code execution are 62 in total. Of those vulnerabilities 52 affect Windows 10 and 10 affect Chrome 64. The identified vulnerabilities can be exploited via social engineering attacks. The vulnerabilities are mitigated in the next security patch of Windows 10 and the next version of Chrome browser. Since both Windows 10 and the Chrome browser are updated automatic, as shown in Fig. 4.10, no action is required by the engineers.

Model-based security insights

The Security insights algorithm uses some properties from the model to produce suggestions to the engineers that perform the security analysis. The suggestions are
independent of the mechanisms, threats, and constraints of the models. A security engineer may identify additional threats and constraints based on the output of the insights.

The security insights algorithm is applied to the mode shown in Fig. 4.3. The model encapsulates all the network constructs of the system that were shown in previous figures. Security and social constructs are not included in the model since they are not used by the algorithm.

By executing the security insights algorithm on the model as shown in Fig. 4.11, the following insights are provided. Those insights are provided independently of the mechanisms and constraints of the system.

1. n2: Turbo X, n3: laptop Dell, n5: Caddx alarm system, n6: iPhone 5s, n7: iPhone 7, n8: iPad, n50 desktop Dell, n51: HP Printer. Devices in the perception layer require physical security.

2. n1: ASUS router, n4: Repeater Devices in the gateway layer are usually externally facing nodes. Malicious actors can target them with network attacks.

3. n15: iCloud service, n16: Gsuite service, n37: Caddx web service. Devices in the application layer are usually provided by third parties. The security configuration

---

![Fig. 4.11 Security insights of the model](image-url)
of third-party devices must be taken into consideration since it affects the security posture of the overall system.

4. \( n_{15} \): iCloud service, \( n_{16} \): Gsuite service, \( n_{37} \): Caddx web service. Treat devices that cannot be updated as compromised.

5. \( n_{9}, n_{10}, n_{12}, n_{13}, n_{17}, n_{18}, n_{19}, n_{38}, n_{41} \): connection. Wireless connections are subject to information disclosure attacks. Use encrypted protocols.

6. \( n_{1} \): ASUS router, \( n_{2} \): Turbo X, \( n_{3} \): laptop Dell, \( n_{4} \): Repeater, \( n_{5} \): Caddx alarm system, \( n_{6} \): iPhone 5s, \( n_{7} \): iPhone 7, \( n_{8} \): iPad, \( n_{50} \): desktop Dell, \( n_{51} \): HP Printer. Define a policy to update Devices that require user action.

7. \( n_{42}, n_{43} \): iOS 11.2.3 Define a policy to update Applications that require user action.

### 4.1.5 Case Study Results

The results gathered from the application of the framework are discussed in this section. First, quantitative values for the metrics introduced in Section 4.3.2, based on the intermediate and final outputs of the framework, are calculated. Next, the exit interview of the involved stakeholders is summarized to extract some empirical conclusions regarding their experience during the framework’s application.

#### Metric evaluation

The metrics specified in Section 4.3.2 are divided into three different areas. Each area evaluated a different aspect of the functions of the framework.

**Function’s intention:** evaluates if the function does what it was intended to do. The metric is a Boolean value. 1) The first function that is evaluated is the *generation of implementation phase model through network capture files*. The initial model of the case was generated through a concatenation of network capture files that were provided by the stakeholders. The model generation algorithm only creates models using the network module of APPARATUS which are available from the network capture files. The algorithm created a model through network capture files and as a result, the algorithm does what it was intended to do; 2) The second function that is evaluated is the *model-based vulnerability identification*. Unlike the other functions of the framework, the algorithm is semi-automatic. The algorithm intends to automate the vulnerability identification of the system and store the output into a structured JSON file for further
The algorithm was applied to three different models and performed its intended function. 3) The third function that is evaluated is the *model-based security insights*, which outputs security suggestions and issues using information on a model. The algorithm was applied to a model containing all the network module constructs of the system. The algorithm generated a list of suggestions based on the attributes of the constructs.

The evaluation of the metrics are presented in Tab.4.2.

<table>
<thead>
<tr>
<th>Function</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model generation</td>
<td>True</td>
</tr>
<tr>
<td>Model-based vulnerability identification</td>
<td>True</td>
</tr>
<tr>
<td>Model-based security insights</td>
<td>True</td>
</tr>
</tbody>
</table>

**Function’s required resources:** evaluates the resources needed for the function to be completed. 1) The model generation algorithm requires a network capture file as an input. Using the network capture file, the algorithm can generate an implementation phase **APPARATUS** model of the system. The model only contains the device, application, and connection constructs. Other conceptual constructs, such as Micronet, Net along with security and social constructs are not included in the model. Those constructs cannot be elicited from a network capture file and require the input of a security engineer. However, the generation of a model using the algorithm when compared with a manual approach is instant. Furthermore, the necessary input from the stakeholders is reduced, since they do not need to produce a map of their network infrastructure. The manual process of eliciting the network information from stakeholders is arbitrary and cannot be accurately measured. For example, stakeholders may have a manifest with the devices and applications that they use, but as demonstrated in this case study, it is not always accurate. In such a scenario, the algorithm reduces the time and resources required to create a model while improving on the accuracy of the model. In other situations, stakeholders may not know the network architecture they use, or they may need time to gather the necessary information. In such cases, the algorithm reduces the time and resources from both the security engineer, as well as the stakeholders. The model generation algorithm is not constrained by such issues. It allows an engineer to spend more time eliciting the security requirements from the stakeholders rather than the existing architecture of the system; 2) The model-based vulnerability identification algorithm
requires a vulnerability database as a resource. The algorithm reduces the amount of time needed to identify vulnerabilities by a security engineer since it automates the process. The manual process of vulnerability identification is a task with heavy repetition. An engineer will need to gather keywords of the components of the system, such as services and applications versions. Then, each keyword will be placed in a vulnerability database to produce a list of vulnerabilities. The amount of time that is saved by the use of the algorithm is dependent on the size of the model. Manual enumeration of systems containing hundreds or thousands of nodes would require days while a system with as little as ten nodes could be performed in a few hours. Either way, the resources that are saved are significant; 3) The model-based security insights does not require any external input. The only requirement is an implementation phase model. As such, its application does not require the expenditure of any time or resources by a security engineer. The Tab. 4.3 shows the time needed for a function to execute in relation to the same task being undertaken manually.

<table>
<thead>
<tr>
<th>Function</th>
<th>Automated function time</th>
<th>Manual time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model generation</td>
<td>173.09msec</td>
<td>243min</td>
</tr>
<tr>
<td>Model-based vulnerability identification</td>
<td>8.51sec</td>
<td>572min</td>
</tr>
<tr>
<td>Model-based security insights</td>
<td>75msec</td>
<td>58min</td>
</tr>
</tbody>
</table>

The automated functions provided by the framework substantially reduce the amount of time required to complete the analysis process. One issue is the Model-based vulnerability identification, as shown in Tab. 4.3. It requires a significant amount of time when compared to the other functions. The reason is the implementation of the algorithm in ASTo. Each vulnerability keyword is sent as a single request to the vulnerabilities database server. The implemented process is similar to this.

- Send first vulnerability keyword to vulnerability database.
- Wait for the response by the server.
- Display and process the response.
Evaluation

- Send first vulnerability keyword to vulnerability database.
- Wait for the response by the server.
- Display and process the response.
- Repeat until all keywords have been sent to the vulnerability database.

The present implementation means that the time needed for the function is related to the number of vulnerability keywords identified on a model. Since each keyword is processed individually by the server and the application, the time required increases the more keywords need to be processed. To increase performance in such a scenario, a solution is to host a vulnerability database in the machine that does the processing. This eliminates the network latency imposed by the use of an online database, which is the main bottleneck of this function.

Tab. 4.4 shows the improved performance of using a locally hosted vulnerability database over an online vulnerability database. To perform the comparison a local copy of the CVE database was downloaded from the link\(^\text{14}\). The downloaded file was made accessible to the tool by changing the URL\(^\text{15}\) of the vulnerability database to the path\(^\text{16}\) of the file’s from the tool’s settings. The cveDatabase.html file is an HTML copy of the vulnerability database. Since there is no server to handle the request, the tool automatically handles URLs with the “file:///” prefix with regular expression searches, instead of HTTP requests. To produce the results shown in Tab. 4.4, the same list of keywords as the ones of the case study were used.

Table 4.4 Comparison of online and local hosted vulnerability databases

<table>
<thead>
<tr>
<th>Type of database</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hosted Online</td>
<td>8.51sec</td>
</tr>
<tr>
<td>Hosted Locally</td>
<td>187msec</td>
</tr>
</tbody>
</table>

Function effectiveness; evaluates the effectiveness of the function compared to the same task being done manually. 1) The model generation, as demonstrated by the case study, can provide more accurate information. In the example of the case, the algorithm generated a model with additional components. The components were

\(^{14}\text{https://cve.mitre.org/data/downloads/index.html}\)
\(^{15}\text{https://cve.circl.lu/api/search/}\)
\(^{16}\text{file:///Users/Downloads/cveDatabase.html}\)
devices and applications that were not included in the manifest or were part of a third-party infrastructure. Regardless of the reason, if the model had been created only using the stakeholders’ input, those components would not have been included in the model. The accuracy of the model would have been affected, resulting in a small security analysis. The benefits of model generation algorithm include reduced errors, as well as, more accurate models; 2) The vulnerability identification algorithm main benefit, besides the reduction workload, is a reduction of human-induced errors. A security engineer manually enumerating vulnerabilities on models is prone to errors. Components of the system can be overlooked, or vulnerabilities can be connected to the wrong node. On the other hand, the algorithm transposes the whole graph, ensuring that all of the components are analyzed. A calculated missing feature of the algorithm is that the resulting output requires additional analysis. In the case study, a third-party utility was used to parse the output. Security engineers have different workflows on identifying vulnerabilities that are relevant to the system. Identification of relevant vulnerabilities requires knowledge of the architecture and the security requirements of the system. Such requirements are expressed in natural language as require significant effort to be used in an automated manner without errors. For that reason, that of identifying which vulnerabilities are relevant to the system is delegated to the security engineer. Engineers use dedicated utilities, such as the VulnWhisperer, regular expressions or manual search operations through the online versions of vulnerabilities databases. The output of the algorithm is a structured JSON file, that is supported by VulnWhisperer and any other utility with a JSON parser; 3) The model-based insights is a way to encode security analysis expertise in the Framework. The main benefit of the application of the algorithm in the case was the identification of certain behaviors of the system that could result in security issues. For example, a large number of devices and applications required an action to be updated. The system did not have a constraint that dictated a policy on how those constructs would be updated. In the case of the development of new vulnerabilities, the constructs would be vulnerable until an update has been applied. A different security insight validated an existing constraint of the system. Perception layer devices might require physical security mechanisms. Two devices in the system, the desktop Dell, and HP Printer belonged to the perception layer. The devices were considered assets of the system. For that reason, they were located in a physically secure location without any connection to the rest of the network. The security insights helped establish a baseline security approach on the system.
Stakeholders’ Interview

A short interview was performed with the participating engineers to: 1) capture their experiences regarding the application of Framework; 2) identify what they perceived as its contributions and shortcomings.

First, the engineers were asked about their opinion on the modeling language of the framework. The engineers noted that they were not familiar with similar model-based security frameworks. The APPARATUS framework was their first experience with that type of analysis. Overall, they found the modeling language easy to understand and its concepts mapped to real-life architectures. A significant benefit for them was that understanding models did not require any cognitive shift from their map of the architecture. The majority of the concepts were self-explanatory. The only concept that needed further explanation during the initial interview was the concept of Micronet. Overall, they found the modeling language natural and easy to adopt without the need for specific training.

Second, the engineers were inquired about the use of the functions provided by the framework. They noted that the automated nature of those function made the benefits of the framework more substantial. Furthermore, the functions reduce the amount of time required to manually perform those operations, as well as human-induced errors. The function that had the most benefit to them was the model generation through pcap files. That function significantly reduced the friction of adopting the framework in addition to creating models that are an accurate representation of the actual system. As they noted during the case, the function was able to identify additional devices and applications that were not supplied by the stakeholders. This is a significant benefit during security analysis since all components of the system need to be known to access its attack surface. Devices that are “forgotten” or unknown are in most cases the point of entry during an attack. By including such devices in the model, the security analysis is comprehensive. According to their opinion, the model generation function was a significant factor in adopting the APPARATUS framework in their infrastructure.

Thirdly, the engineers were asked about their opinion on the tooling offered by the framework and whether it improved their experience in using the framework. The engineers stated that the tool was a deciding factor on only using but understanding the framework. They noted the configuration options of the UI improved the overall experience. Another benefit of the tooling was the user-assisted model functions. Those functions prevent users from making syntactical and semantical errors in the models. The functions helped to reduce the learning curve of the framework. The engineers provided a suggestion regarding the modeling language tooling of the framework.
Specific attributes of concepts can only take enumerated values as input. For example, the layer attribute of the device can only be perception, gateway or application. ASTo (at the time of the case study), when editing those attributes do not provide a list of the possible values and instead takes any string value as input. If ASTo instead of raw string input, provided a list of values, it would significantly improve the experience of users.

The feature has been implemented in ASTo after the conclusion of the case study.

The overall experience with the usage of the framework was positive. The engineers noted that the framework was easy to use and to apply to network systems. Moreover, when applied to existing, it offers valuable insights into the security posture of the system and determining the actual components of the system. They noted that the framework reduces friction with stakeholders by establishing a common language on describing the components of the system. Their suggestion on future work on the framework was to integrate into a real-life system to provide continuous integration and security analysis. Devices and application used by the system could be shown on the real-time model of the system in addition to the active mechanisms and constraints.

### 4.1.6 Threats to the validity of the Case Study

The case study was performed to evaluate a specific set of analysis functions offered by the framework. The framework used to model and assess the attack surface and security posture of a real-life system. The participants in the case included two security engineers that were employed by the company that used the system. While the engineers were familiar with information security concepts, they had no experience with model-based security analysis. Nevertheless, the generalisability of the outcomes of the specific case study can be considered limited due to the involvement of a small number of stakeholders using the proposed framework and its application to a single real-life system. The limited generalisability issue was partially mitigated by the previous smaller-scale applications of the framework, as described in Section 4.1, the findings of which were in accordance with the outcomes of the case study presented in this section. Furthermore, the protocol of the case study, as presented at the beginning of this section, can facilitate its replication in other large-scale information systems in future work to further solidify the findings.

Finally, even though some quantitative metrics were identified for the evaluation of the results of the case study, the majority of the insights originated from the interviewing the case study participants and, therefore, were qualitative. While the quantitative metrics were able to capture the conceptual and security-related completeness of the
produced artifacts, they were not able to provide any further indication of their quality as there was no previous baseline for comparison. Thus, the opinions and experiences of the involved stakeholders, while potentially subjective, were the primary source for the evaluation of the proposed framework’s application to the system. To mitigate such issues in future work, researchers could identify legacy IoT systems which can be redesigned using the framework and compare their new design with the previous baseline. Alternatively, if a similar approach for the modeling and security analysis of IoT systems is identified in future literature, it can be applied to the same system selected for our case study and have the results of both applications compared in a quantitative way.

4.2 Lessons Learned

The different evaluation activities, presented in this chapter, facilitated the refinement of the framework to its current state. The PoC applications of parts of the framework, performed in the earlier stages of the research project, provided valuable insights which led to the improvement of each component iteratively. Next, the case study, which constituted the last step of the framework’s evaluation process, facilitated the creation of the final version of the different framework components. This was due to the nature of the selected system, as it allowed to observe the application of the different framework components in a relatively large-scale and complex real-life scenario and identify potential shortcomings.

The initial application of the framework could only express devices, their connections, and their conceptual domain. That application acted as a statement of the main components framework. Those components, the device, connection, and Micronet along with the attributes, were the basis of the framework. As it was noted by the publication at that time, the concepts could not be used to model the security analysis, but based on their attributes could be used to infer specific security issues. The first PoC, described in Sec. 3.5 introduced the modular language of the framework. Concepts were divided between network, security, and social modules.

The additional network concepts were required to represent systems more accurately. Such concepts included the application, to represent software components, information, to represent data and other knowledge in a system. The security concepts were introduced to facilitate security analysis. The approach to security in the framework is asset-centric. A threat can only target assets of the system, which is not a common approach in security frameworks. The reasoning behind that choice is domain specific.
requirements of IoT. IoT systems do not have clear boundaries or requirements. As a result, resources are committed to protecting the aspects of the system that are considered assets. For example, the temperature of an area is an asset in specific systems, as is the case of a climate controlled room storing valuable relics. In other scenarios, the temperature is not considered an asset, such as the temperature of along an urban road. By adopting an asset-centric approach, the framework can be used to constrain the security analysis based on the requirements of the stakeholders and the assets of a system.

The social module was introduced to model users and stakeholders of a system. The aim of the social module was to capture the user-machine interaction and evaluate targets of social engineering attacks. For example, passwords and other access controls are information possessed by specific stakeholders of the system. Such stakeholders and the information they know are prime targets of social engineering attacks. Models need to be able to capture those relationships since they play an essential role in a security analysis of social engineering attacks.

The last PoC introduced the design phase metamodel of the framework. The previous PoCs were developed with the aim of analysis during the implementation phase. However, there is a need to provide mechanisms for security analysis during the whole development process, especially during the design phase. A characteristic of the design phase is the high-level concepts that omit low-level details that are not known at the time. To facilitate the analysis between the two-phase, the design phase metamodel was designed as a high-level version of the implementation phase metamodel. To further reduce friction during analysis between the different phases, the transition algorithms were developed. Those algorithms provide an automated process where a model is transitioned to the other phase. During the last PoC, the final version of the modeling language was presented. Some concepts were removed or remodeled since they added complexity without significant value. One such concept was the unidentified node. The unidentified node represented a device that was not part of a Micronet. While not necessarily malicious, it was not known by the system, and a such was considered compromised. An unidentified node could only have a relationship to the concept of Net. The use of that concept was difficult for other engineers, but because its only relationship was to the net, it produced wrong models. A model did not capture which devices that unidentified node was connected to, and as a result, the proposed mechanisms were not accurate. Additionally, the unidentified could not be used to represent applications that were communicating with the Micronet. The
unidentified node was incorporated to the concept of Device, by allowing a device to have a relationship to a Micronet and a net.

4.3 Usage of ASTo

4.3.1 ASTo’s Open source community contribution

ASTo is an Open source project under the MIT license. The tool’s code repository is hosted on a public repository on Github\textsuperscript{17}. Although ASTo was labeled as a prototype application with the aim of supporting the APPARATUS Framework, it has gained significant recognition from the security community. It was voted as one of the Top 5 security tools of July 2017 by Hack with Github\textsuperscript{18}. During the course of writing this document, ASTo has been featured in a variety of blogs and social media from around the world.

A significant number of developers follow the development process of ASTo through the application’s public repository. Throughout the application’s development, the project used a public roadmap that prioritized the development of features for each milestone. The roadmap was used to inform developers, what type of new features would be implemented in the master branch and when. Additional use of the repository was the issue tracker. The issue tracker was used by developers as well as the author to report and resolve issues of the application. Issue reporting was beneficial for stabilizing the application in the different operating systems. The operating system that the application was being developed was macOs. The testing of the application in Windows and Linux operating systems was done by the community. The majority of the cross-platform issues that were reported from users were about User Interface (UI) inconsistencies and file operations between the different operating systems.

The repository has been forked 69 times, by the time of writing. A fork is a copy of the repository from another account. Forking a repository allows someone to experiment with changes without affecting the original project freely. Developers fork a project to contribute to it or to start another project using the fork as a development base. A number of software security companies have forked the repository of ASTo and then continued the development in private repositories. Those repositories closely follow the development of the ASTo public repository, although the development status of private repositories cannot be known.

\textsuperscript{17}https://github.com/Or3stis/apparatus
\textsuperscript{18}https://medium.com/hack-with-github/top-5-security-tools-july-2017-43d86db37135
ASTo does not use analytics software to record usage statistics. Furthermore, there are no plans to integrate any usage statistics in the tool. The information presented is based on the publicly available usage statistics of the repository. Those statistics were provided by Github and only hold a 7-day history of stats. Because the tool does not record any usage statistics, the actual number of active users or developers that have privately forked the project cannot be accurately measured. The Fig.[4.12, 4.12, 4.12, 4.15] show certain statistics of the network activity of the repository of ASTo from 28/06 to 27/07. The spikes in Fig.4.12 and Fig.4.14 are consistent with the release of new versions of the application, while the of the graph shows the behavior of developers that follow the master branch.

Fig. 4.12 ASTo traffic statistics from 28/06 - 11/07
Fig. 4.13 ASTo referrer webpages from 28/06 - 11/07

Fig. 4.14 ASTo traffic statistics from 14/07 - 27/07
Due to the ASTo’s modular architecture and ease of extension, it has been used as the basis for the development of tools in other research projects. One project modified ASTo to support the Secure Tropos methodology for cloud security, while another project modified ASTo to support the Secure Tropos methodology and business process diagrams. A different project modified ASTo for 5G system security analysis. Other notable forks of the ASTo repository include IoTSploit and Infosecsecurity. IoTSploit and Infosecurity are penetration testing companies with heavy involvement in open source projects. Their forks follow the development cycle of ASTo and have not been modified.

A further testament to ASTo’s popularity is its incorporation to the teaching curriculum of Piraeus University of Applied Sciences. The department of Electronics offers courses, at the Bachelor and Master level, of IoT development and security. ASTo is planned to be used during tutorial sessions for students to develop IoT systems and reason about their security. Students will use ASTo to design and analyze IoT systems utilizing the APPARATUS Framework.

\[19\) https://github.com/NOMNUDS/apparatus
\[20\) https://github.com/nickarg/apparatus
\[21\) https://github.com/CapriTechLimited/5G-SAT
\[22\) https://github.com/iotsploit/apparatus
\[23\) https://github.com/infosecsecurity
\[24\) https://maddevs.io
4.3.2 Feature comparison of ASTo and relevant Security Tools

ASTo has been developed to make the adoption and usage of the APPARATUS Framework easier by the security community. The following Tab. 4.5 is a feature comparison between ASTo and relevant security tools, that has been identified during the literature review.
### Table 4.5 Feature comparison of ASTo and relevant Security Tools

<table>
<thead>
<tr>
<th>Feat.</th>
<th>Tool</th>
<th>ASTo</th>
<th>Metasploit</th>
<th>Nmap</th>
<th>Burp</th>
<th>Snort</th>
<th>Bro</th>
<th>Wireshark</th>
<th>Maltego</th>
<th>w3af</th>
</tr>
</thead>
<tbody>
<tr>
<td>system visualization</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>graph-based analysis</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>security insights</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>vulnerability identification</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>vulnerability parsing</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>network scanning</td>
<td>-</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>pattern identification</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>automated generation of models</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>exploitation of systems</td>
<td>-</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>network packet analysis</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>-</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
4.4 Summary

This chapter presents the case study of the APPARATUS Framework. The case study was performed in an undisclosed security company with the participation of two engineers of the company. The case study selected for the application of the framework involved a technical company. The name of the company and certain results of the case study are under a confidentiality agreement between the author and the company. The case study aimed to evaluate the analysis processes of the framework. The analysis processes are used to assess the current security policies and infrastructure of the company, identify security issues and propose security mechanisms. The engineers acted as the stakeholders of the system, while the author acted as the security engineer that applied the Framework. The functions of the Framework were evaluated according to their intention, required resources, and effectiveness. The functions were evaluated with qualitative and quantitative methods. The qualitative method involved an exit interview with the stakeholders. Next, the usage of the supporting software application ASTo by the open source is discussed. ASTo has been adopted in the curriculum of Piraeus University of Applied Sciences. The department of Electronics offers courses, at the Bachelor and Master level, of IoT development and security. Additionally, the repository of ASTo has been forked by a number of cyber security companies for internal use. Forks of ASTo have been modified to support the Secure Tropos methodology for Cloud systems, secure business processes, and 5G systems. The chapter ends with a feature comparison between ASTo and the security tools identified in Chapter 2.
Chapter 5

Conclusion

In this work, a security framework for design and analysis of IoT system was presented. The framework is comprised of different components. The components of the framework have different functionalities which, when applied in a sequence can produce an IoT system model, compliant with the security requirements of the system’s stakeholders. The framework is composed of the following components: (1) a modeling language to represent IoT systems; (2) a modeling methodology to create models; (3) processes to assess the security of the models and (4) propose countermeasures to increase the security posture of the models. The framework was applied to case studies to evaluate its effectiveness in performing security analysis in IoT systems.

The modeling language of the framework is composed of two metamodels. A metamodel to create models in the design engineering phase, and a metamodel to develop models in the implementation phase and an approach to model the different states of models. The distinction is made due to the different requirements, and information engineers have about a system during each phase. During the design phase, an engineer models the idea of the system without being restricted by the hardware or software specifications. For example during the design phase, an engineer may require a system component that will function as an Intrusion Detection (IDS) system. The engineer may not know at the design time whether the IDS will be a hardware device or a software application. During the implementation phase whether the IDS will be a hardware device or a software application is necessary since it affects both the topology of the network and its security requirements. During the implementation phase engineers have more information about an IoT system, such as versions of software applications, operating network ports, user profiles, and external facing nodes. Information about the system’s architecture can be included in the modeling instances of an IoT system to create models that accurately represented real-life systems.
To apply the framework, a security engineer can follow the modeling methodology of the framework. The modeling methodology describes the necessary steps an engineer needs to apply to create IoT system models and evaluate their attack surface. The methodology provides different actions depending on the level of analysis and engineering phase of the system. Specifically, the modeling methodology offers the following approaches: (1) development of an IoT model in the design phase; (2) a transition of an IoT model from the design to the implementation phase; (3) development of an IoT model in the implementation; (4) a transition of an IoT model from the implementation to the design phase.

The security analysis can be assisted by a set of algorithms that can be applied either in an automated manner or a semi-automated manner. The algorithms can be used to verify the proposed constraints and mechanisms of the system, identify vulnerabilities or offer suggestions to improve the security posture of the system. Specifically, the framework has the following algorithms: (1) threat coverage based on the proposed constraints; (2) vulnerability verification based on the proposed mechanisms; (3) model validation based on the rules of the modeling language; (4) automated implementation phase model generation using network capture files; (5) automated vulnerability identification based on the attributes of a model; (6) automated model transition from design to implementation and vice versa; (7) a pattern identification process based on user input;

The framework is supported by a dedicated software application named ASTo. ASTo can be used to facilitate the creation of models and the application of the algorithms. Furthermore, the tool offers customization options on the front-end representation of the language. The software application has been adopted by various members of the security community and was voted as one of the top 5 security tools of 2017.

The various components of the framework have been evaluated in proof of concept applications. Each application was designed to assess specific aspects of the framework such as the expressiveness of the language. Moreover, the framework was applied to a case study on a real-life system. The case study was used to evaluate the effectiveness and resources required by the algorithms.

5.1 Research Outputs

The contributions of the different framework components can be matched to the objectives and research questions this research project aims to tackle (see Sections 1.2 and 1.3). More specifically, in regard to the first research question, on the fundamental
characteristics and requirements of an IoT system, it was answered by performing a literature review (Obj1). The results of the literature review were used in Obj2 to identify the necessary components of the modeling language. The terminology of the language was based on existing network security tools and frameworks. This was made to facilitate the usage of the language by security engineers. Another reason for this is to enable models, tools, and processes of the framework to be used by existing tooling and workflows. For example, some tools are used to parse through vulnerabilities databases. Using a similar terminology of the vulnerability in the framework and developing tooling that exports vulnerabilities from models in a standardized format that can be imported to other tools, the framework can be incorporated in existing security analysis workflows.

Regarding the RQ3, on the type of security requirements of an IoT system that can be elicited from its hardware architecture, is addressed in Obj3. In the majority of IoT use case, the IoT system will be designed on top of an existing legacy system. Such legacy systems have existing requirements and constraints. To address this, a legacy system is considered a stakeholder during the requirements elicitation phase. Its requirements are derived from its hardware architecture. The identified requirements are then used in during the design of the IoT system.

The RQ4 refers to the types of security analysis that need to be performed in an IoT system to offer baseline security, is addressed by the use of formal processes. The processes fulfill the Obj4. A process is used to verify a model’s conformance with the modeling language. The process can be performed when an engineer is developing a model or need to check a model created by other means. A second process performs semantic validation of security concepts. This is done by checking the types of security concepts and their relationships. A third process provides an automated approach to generate implementation phase models from network capture (Pcap-ng) files. A fourth process is used to highlight possible security issues and propose steps for mitigation using information from a model, in an automated manner. A fifth process is used to identify vulnerabilities using information of a model and a vulnerability database. The process creates a list of keywords based on attributes of concepts and then inputs that list to a vulnerability database. The result is a JSON file of the possible vulnerabilities of a model. A sixth process is used to identify patterns on models. A security engineer can enter a pattern as a list of keywords to highlight nodes that fulfill the criteria.
5.2 Future Research Directions

Securing an IoT system is an extensive undertaking that requires proficient knowledge of multiple domains. The development of APPARATUS, as any research project, started with bold aims that were humbled into realistic targets.

A goal of this research project was to provide solid foundations for security analysis in IoT for researchers to build on. For that reason, both the modeling language of the framework and ASTo are modular and extendable. The framework can be extended or adopted by other methodologies to provide a link to security analysis in IoT. Furthermore, during the development and evaluation of the framework, some exciting research directions for the framework had been identified.

The APPARATUS framework was developed with an emphasis on security rather than privacy analysis. However, due to the framework’s modular architecture, it can be extended with privacy components in future work. Privacy is often considered as a security requirement during the development of IoT systems. Nevertheless, research in the area of privacy requirements engineering has identified discrete types of privacy requirements which require further consideration. Recent regulations, such as the General Data Protection Regulation (GDPR\footnote{https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679}, enforce strict rules on how users’ personal information is to be handled. Such regulations can be incorporated as individual modules of the framework. For example, a GDPR extension module can be developed, where its requirements will apply to specific constructs of the modeling language. Moreover, the module can introduce additional processes which will automatically check the compliance of the system with the GDPR, using the security verification process supported in ASTo. Compliance based modules and processes offer significant benefits to security engineers using the framework since it will reduce the number of tools and resources required during the development of an IoT system.

A significant issue during the literature review and the development of the framework was the lack of accessible models from other frameworks and methodologies. While specific frameworks and methodologies had been identified, the majority of them did not have a repository of models or examples of their analysis other than the ones included in publications. The majority of models covered in publications either presented a particular usage of the proposed framework or an extension to address a specific issue. Models that included extensive analysis of systems were either hard to find or outdated. As a result, comparison with existing frameworks was challenging. However, the lack
of resources for learning and using a framework deters other researchers from using and contributing to it. For that reason, ASTo, incorporates the documentation of APPARATUS in addition to a tool-based tutorial on the usage of the framework.

A separate project from ASTo is an open source repository of APPARATUS models. The repository holds all the models that had been created by the author during this research project. The aim of the repository is to create a centralized archive of models for education and development. The repository could be used for training and experimentation by engineers that want to familiarize themselves with the framework. Once, the repository has a significant number of models; it can be used as a dataset for natural language processing and machine learning. Both options offer exciting possibilities for future research. For example, while some attributes of constructs of the modeling language have enumerated values, others use a string as an input. The strings are written in the natural language of the security engineer. To elicit meaning from those attributes, natural language processing needs to be applied. By being able to extract the intent of security engineers from the models, the security insights algorithm can be enhanced significantly.

Additionally, other algorithms can be developed, that will facilitate the security analysis even further. Such algorithms could evaluate the effectiveness of proposed constraints concerning threats, identify additional threats based on the system’s requirements and propose access control mechanisms based on an actor’s intent. Machine learning could be used to automate further functions of the framework. For example, an engineer could supply as in input a network capture file and a list of security requirements. ASTo could create an APPARATUS model with the resulting security concepts and an accompanying security report of the vulnerabilities and issues of the system. Similar automation can significantly reduce the number of resources for the security analysis of systems.

The main issue of the repository of APPARATUS models, it to create enough models that could be used as a useful dataset. The current number of models in the repository is small since the framework was finalized in the final year of the research project. All the models so far have been provided by the author. However, due to the popularity of the ASTo and its adoption in the curriculum of Piraeus, the community can help by donating models to the repository. To further facilitate that process, the functionality of uploading models to the repository will be incorporated in ASTo. The author plans to engage with the community by providing support, teaching materials, and other tools for using the dataset. Security researchers and students will be able to leverage the repository for their purposes and further the development of the framework.
ASTo as a tool was vital for the adoption of the framework by the security community. A notable factor for that success was its modularity and integration with third-party utilities. For example, ASTo supports two operating modes. A stable mode for normal usage of the tool and a developer mode for testing additional features. Further development will introduce a formal Application programming interface (API) that will increase the extensibility of the tool. The API will allow engineers to directly apply machine learning and natural language utilities to the tool without the need to modify its source code. The API will increase the adoption of the tool in other research domains by reducing the friction of tool integration.
References


Appendix A

Terminology

Due to the many different research areas that are part of the Internet of Things Security, it is essential to define a unified terminology. Words tend to have various meanings when used by people in multiple disciplines. What follows is the definition of any relevant term for this research. The terms are listed alphabetically.

- **Actor**: is used to represent people or groups of people that interact with an IoT system [36]. An Actor can be a stakeholder of the system. An Actor may never be malicious. The concept of Actor can be used to represent groups of people with different privileges, such as root users or the administrative personnel of a University.

- **Auditing**: Auditing is the information gathering and analysis of assets to ensure such things as policy compliance and security from vulnerabilities.

- **Antivirus and antimalware software**: Software that is used to scan for malware upon entry in the network and track files to detect anomalies and fix damages on files or network configurations.

- **Application**: is part of the information world (information thing) [51]. An Application represents a software component that is running on a Device.

- **Application security**: is the process of identifying and mitigating vulnerabilities in applications.

- **Asset**: any actor, device, application or information of the system that either (1) is considered valuable by the stakeholders and needs to be protected; (2) a malicious actor wants; or (3) acts as a stepping stone to further attacks. Assets
that are valuable to the stakeholders can be elicited during the requirements phase. On the contrary, assets that malicious actor wants or can be used for further attacks are not always apparent. Examples of assets are the access credentials known by an actor, sensitive information stored in a database or a sensor that has read/write privileges to a server.

- **Attacker**: a malicious entity who is intending to compromise a system. It is represented by a malicious actor in Apparatus.

- **Attack surface**: The number of different points an attacker can use to gain access to a system.

- **availability**: on-demand access to the IoT system by legitimate device or actor.

- **Backdoor**: A backdoor is a tool installed after a compromise to give an attacker easier access to the compromised system around any security mechanisms that are in place.

- **Behavioral Analytics**: is software that identifies network behavior that deviates from the expected network behavior.

- **concept**: an element (UML class) of the metamodel.

- **Broadcast**: To simultaneously send the same message to multiple recipients.

- **broadcast Address**: An address used to broadcast a datagram to all hosts on a given network using UDP or ICMP protocol

- **Confidentiality**: information that is not disclosed to unauthorized entities.

- **Data loss prevention**: to ensure that sensitive information of an organization is not sending outside its network.

- **Cipher**: A cryptographic algorithm for encryption and decryption.

- **Connection**: the type of network communication protocol used between the Devices.

- **Constraint**: is “a restriction related to security issues, such as privacy, integrity, and availability, which can influence the analysis and design of a system under development by restricting some alternative design solutions, by conflicting with some of the requirements of the system, or by refining some of the system’s objectives” [36].
• **Daemon**: A program which is often started at the time the system boots and runs continuously without intervention from any of the users on the system. The daemon program forwards the requests to other programs (or processes) as appropriate. The term daemon is a Unix term, though many other operating systems provide support for daemons, though they’re sometimes called other names. Windows, for example, refers to daemons and System Agents and services.

• **Datagram**: Request for Comment 1594 says, "a self-contained, independent entity of data carrying sufficient information to be routed from the source to the destination computer without reliance on earlier exchanges between this source and destination computer and the transporting network." The term packet has generally replaced the term. Datagrams or packets are the message units that the Internet Protocol deals with and that the Internet transports. A datagram or packet needs to be self-contained without reliance on earlier exchanges because there is no connection of fixed duration between the two communicating points as there is, for example, in most voice telephone conversations. (This kind of protocol is referred to as connectionless.)

• **Denial of Service**: The prevention of authorized access to a system resource or the delaying of system operations and functions.

• **Dictionary Attack**: An attack that tries all of the phrases or words in a dictionary, trying to crack a password or key. A dictionary attack uses a predefined list of words compared to a brute force attack that tries all possible combinations.

• **Digital Certificate**: A digital certificate is an electronic 'credit card' that establishes your credentials when doing business or other transactions on the Web. It is issued by a certification authority. It contains your name, a serial number, expiration dates, a copy of the certificate holder’s public key (used for encrypting messages and digital signatures), and the digital signature of the certificate-issuing authority so that a recipient can verify that the certificate is real. **Digital Signature**: A digital signature is a hash of a message that uniquely identifies the sender of the message and proves the message hasn’t changed since transmission.

• **Device (IoT node)**: initially named IoT node in [73]. It is an object of the physical world (physical thing) or an object of the virtual world (virtualized thing) [51]. It is used to represent either physical components, such as hardware-
based actuators and mobile phones or virtualized components, such as cloud-based devices of an IoT system.

- **Eavesdropping**: Eavesdropping is simply listening to a private conversation which may reveal information which can provide access to a facility or network.

- **Encryption**: the Cryptographic transformation of data (called "plaintext") into a form (called "ciphertext") that conceals the data’s original meaning to prevent it from being known or used.

- **Event**: An event is an observable occurrence in a system or network.

- **Email Security**: is software to block incoming attacks and control outbound messages to prevent the loss of data.

- **firewall**: are a virtual barrier between a trusted network and untrusted networks. They either block or allow network traffic depending on rules.

- **Flooding**: An attack that attempts to cause a failure in (especially, in the security of) a computer system or other data processing entity by providing more input than the entity can process properly.

- **Hardening**: Hardening is the process of identifying and fixing vulnerabilities on a system.

- **Hash Function**: An algorithm that computes a value based on a data object thereby mapping the data object to a smaller data object.

- **Information**: represents data and knowledge in a system. Examples of Information are authentication logs, temperature data, access credentials, and user passwords.

- **Internet of Things system**: A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies. Note-1: Through the exploitation of identification, data capture, processing and communication capabilities, the IoT makes full use of things to offer services to all kinds of application, while ensuring that security and privacy requirements are fulfilled. Note-2: From a broader perspective, the IoT can be perceived as a vision with technological and societal implications.
• **Intrusion prevention systems (IPS)**: scans network traffic to block attacks actively. IPS can correlate global threat intelligence information to track the progression of suspect files and malware to prevent the spread of outbreak and reinfection.

• **Malicious actor**: is a person with malicious intent. Malicious Actors are used to representing attackers or insider threats. The concept of the malicious actor is a generalization of the concept of Actor.

• **Mechanism**: a Mechanism when implemented protects against one or more Vulnerabilities. If the Vulnerability is publicly identified and stored in a vulnerability database, a security engineer can use the proposed security mechanisms to mitigate it. A Mechanism could be applied dynamically when a certain event is detected by the system, or they can be a constant process. For example, during the event of a DoS attack, a system may enlist additional resources to spread the impact of the attack. Once the attack is mitigated, the system will release the additional resources reduce its operational costs.

• **Mobile device security**: mobile security also refers to the Bring Your Own Devices (BYOD) paradigm. Since organizations allow employs and users to use personal mobile to access organization data, the connections and usage need to be protected.

• **Micronet**: is a managed environment of Devices and Applications can be configured in term of their security and enable an IoT system to perform a function. Components that are part of a Micronet are not more secure than others. Security in this context should not be confused with trust. Components in a Micronet do not have inherit trust relationships between them. Stakeholders have control on the security configuration of a Micronet’s component. For example, the security of mobile device that belongs to employee cannot be fully configured by an organization. The employee can install malicious applications or even make changes to the system configuration of the Device. However, the employee might be security conscious and the device can be highly protected, but that security protection is provided by the employee rather than the organization. The Micronet as a concept highlights the components of a system that can be secured by the stakeholders to develop mechanisms to secure interactions with the rest of the system. Examples of Micronets are a smart home, an agricultural network of sensors or a company’s internal network. The boundaries of the
Micronet are defined during the creation of a model by the engineer. For example, one Micronet can include only the devices that are part of a specific network domain, while another can include all the devices that are in the same room. The same device can belong to both Micronets, and each Micronet can impose different security mechanisms on the devices.

- **Net**: represents environments that their security configuration is not known and their behavior cannot be configured by the security engineer. While Nets may not be malicious, they represent a level of danger to an IoT system that must be taken into account during the model development. Similarly to the Micronet, the boundaries of a Net are defined by the engineer. Examples of Net are external networks to the IoT system that a security engineer either has little or no knowledge of, such as a third party cloud infrastructure or hostile deployment environments. It is possible that a Device can be part of Net and a Micronet. For example, an IoT system has a server that hosts a set of virtual machines for its users. While the engineer can configure the server, the usage of the virtualized assets of the servers is configured by the users. A malicious user can try to exploit the virtualized assets to compromise the server. As a result, the virtualized assets compose a Net and not a Micronet.

- **Network Access Control (NAC)**: to block noncompliant endpoint devices or give them limited access to the resources of the network.

- **Network module**: is used to model network objects of IoT systems. The Network module is considered the core module of the metamodel. Every other module is designed as an extension to the Network module.

- **Network segmentation**: is software that classifies network traffic into predefined roles. This classification is used to allow better security policies.

- **Packet**: A piece of a message transmitted over a packet-switching network. One of the key features of a packet is that it contains the destination address in addition to the data. In IP networks, packets are often called datagrams.

- **Phishing**: The use of e-mails that appear to originate from a trusted source to trick a user into entering valid credentials at a fake website. Typically the e-mail and the website looks like they are part of a bank the user is doing business with.

- **Security Goal**: security concerns and objectives set by the stakeholders.
• **Security Requirement**: refined security goal. It should not be tied to a specific technology, but it should allow flexibility when the system is being implemented.

• **Security Framework**: a defined approach composed of various components, that intends to make a system free from security risks and privacy threats.

• **Stakeholder**: a potential owner of an IoT system or someone who expects to receive a benefit from utilizing part or an entire IoT system. They are classified into four categories: (1) manufacturers, (2) developers, (3) deployers, (4) users.

• **Security agent**: a device capable of running software or hardware, that is able to detect security event.

• **Security changes**: changes in the state of the network that results in the current security mechanisms insufficient to satisfy the security constraints and need to be reevaluated. (adaptive module)

• **Security manager**: a device capable of running software or hardware, that is able to detect security event and can also impose security mechanisms.

• **Security Event**: an event that is specified by a security engineer, that may affect the security requirements of the IoT system.

• **Security information and event management**: is documentation of information that the security personnel requires to identify and respond to threats.

• **Security module**: extends the Network and Social modules with security concepts. The security concepts are used to model threats, assets, security controls and attackers in an IoT system.

• **Social Engineering**: A euphemism for non-technical or low-technology means - such as lies, impersonation, tricks, bribes, blackmail, and threats - used to attack information systems.

• **Social module**: extends the Network module in an object-oriented manner with social concepts. Social concepts are used to model users and stakeholders.

• **Tamper**: To deliberately alter a system’s logic, data, or control information to cause the system to perform unauthorized functions or services.
- **Thing**: is an object of the physical world (physical things) or the information world (virtual things), which is capable of being identified and integrated into communication networks.

- **Timestamp**: time measurement.

- **Threat**: a function that can be used maliciously or a system that has the means to exploit a vulnerability of a legitimate system. A threat can only target an asset of the IoT system.

- **Tunnel**: A communication channel created in a computer network by encapsulating a communication protocol’s data packets in (on top of) a second protocol that normally would be carried above, or at the same layer as, the first one. Most often, a tunnel is a logical point-to-point link - i.e., an OSI layer two connection - created by encapsulating the layer two protocol in a transport protocol (such as TCP), in a network or inter-network layer protocol (such as IP), or in another link layer protocol. Tunneling can move data between computers that use a protocol not supported by the network connecting them.

- **Unidentified node** (deprecated): is a Device that is not directly connected to a Micronet and a security engineer has little knowledge of. It may be a malicious device or a legitimate device the is not known by the system. For example, it can be an unauthenticated laptop from a legitimate user trying to connect to an office network, or it can be a laptop operated by a malicious attacker trying to compromise the system.

- **Virtual Private Network (VPN)**: is encrypts the transmitted data from one endpoint to another network.

- **vulnerability**: a software, hardware or usage policy weakness that can be exploited by an adversary toward compromising a system. Hardware and Software Vulnerabilities can be identified using techniques such as penetration testing. Hardware and software vulnerabilities can be identified from public access vulnerability databases such as CVE \(^1\) and NVD \(^2\). Such databases store vulnerabilities using unique IDs. Vulnerabilities IDs are used among security engineers to exchange information about security incidents.

\(^1\)https://cve.mitre.org/cve/  
\(^2\)https://nvd.nist.gov/vuln/search
• **Web Security**: is used to control the web use, block web-based threats and deny access to malicious websites. Web security also refers to protection mechanisms for a website.

• **Wireless Security**: wired networks are more secure than wireless ones since wireless transmissions can eavesdrop. To ensure the security of the network, wireless transmissions require more security layers.

• **Zero Day exploit**: The "Day Zero" or "Zero Day" is the day a new vulnerability is made known. In some cases, a "zero-day" exploit is referred to an exploit for which no patch is available yet. ('day one' - day at which the patch is made available).