UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Modeling Attacks in IoT to Assist the Engineering Process

Luís Carlos Mendes Rodrigues
Candidate

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º cycle studies)

Orientador: Professor Doutor Pedro Ricardo Morais Inácio
Co-orientador: Professor Doutor Tiago Miguel Carrola Simões

Covilhã, setembro de 2020

Acknowledgments

First I would like to thank my family and friends for all the support, motivation and encouragement they have given me during all these years.

Then, I would like to thank my supervisor, Professor Doutor Pedro Inácio for his guidance and always being available to help when questions and issues arose.

I am also very thankful to Mestre Bernardo Sequeiros and Professor Doutor Tiago Simões for all their help and availability during the almost one year it took to complete this dissertation, their assistance was vital to ensure that I could complete this chapter of my academic life.

Last but not least, I would like to thank everyone at TIMWE Lab, as the knowledge I acquired there during this last year was critical for the development of this dissertation.

Resumo

A Internet das Coisas (*Internet of Things*, do inglês e abreviado para IoT), é o nome dado às tecnologias que permitem que qualquer dispositivo (que neste contexto é apelidado de coisa) comunique com outro, tal como com máquinas, aplicações, bases de dados, entre outras tecnologias, de maneira direta. Isto permite que dispositivos num dado ambiente interajam uns com os outros e que sejam inclusivamente capazes de tomar decisões de forma autónoma com base nos dados que recebem. Esta integração de novas tecnologias em dispositivos do dia a dia permite que os utilizadores tenham um controlo mais refinado sobre o que cada aparelho é capaz de fazer, aumentando assim a utilidade e eficiência dos mesmos.

Alguns exemplos da aplicabilidade deste controlo adicional podem ser observados em casas inteligentes, na qual os utilizadores conseguem controlar remotamente os equipamentos da sua casa, ou até definir o seu controlo de forma automática com base em certos parâmetros determinados pelos equipamentos. Isto significa que tarefas como controlo de temperatura, luminosidade, abertura/fecho de portas, ligar ou desligar eletrodomésticos, ou até mesmo encomendar automaticamente um produto quando este termina podem todas ser efetuadas de maneira automática, quando as condições certas são verificadas. Outro exemplo poderia ser no ramo da Medicina, na qual sistemas baseados na IoT podem permitir a criação de sistemas mais eficientes de monitorização e diagnóstico de pacientes, o que acaba por acarretar benefícios a nível da gestão de recursos hospitalares, visto que os pacientes poderiam simplesmente possuir consigo sensores que faziam a sua monitorização permanente, assistindo nos processos de diagnóstico e em casos de emergência.

Contudo, este último exemplo chama à atenção para o problema óbvio e muito importante com estas tecnologias, que é a segurança (ou a falta dela). Casos os sistemas de IoT não cumpram com as medidas de segurança mais adequadas, um atacante poderia potencialmente aceder ou modificar dados dos pacientes, do hospital, ou até mesmo fazer modificações no próprio equipamento. Isto seriam violações gravíssimas da segurança do sistema, que poderiam mesmo provocar prejuízos ao nível de bens materiais ou em casos extremos, de vidas humanas.

Dada a velocidade com que estas tecnologias estão a evoluir, e à qual novos sistemas estão a ser desenvolvidos e implementados, a segurança dos sistemas costuma ser algo esquecida a acaba por ser dos últimos aspetos a ser considerado aquando do *design* dos mesmos. Isto resulta em insuficiências e falhas ao nível de segurança, o que acaba por permitir que atacantes consigam provocar alterações no funcionamento normal ou até mesmo roubar dados do sistema. É extremamente importante efetuar um bom levantamento dos requisitos de segurança que o sistema deve implementar logo desde as fases iniciais de *design* e planificação da arquitetura. Só quando se compreende na integra os requisitos de segurança é que é possível planear e implementar as medidas de segurança adequadas para o sistema a ser desenhado.

O objetivo principal desta dissertação é explorar os princípios por detrás da modelação dos sistemas e das ameaças. Desta forma pretende-se desenvolver um protótipo de uma ferramenta capaz de assistir os utilizadores - mesmo aqueles com conhecimentos limitados na área de segurança - na identificação de requisitos de segurança e ameaças ao sistema, assim como fornecer informação pertinente para colmatar estes aspetos. Esta ferramenta deverá ser capaz de auxiliar os *developers*, *designers* e engenheiros de software com os processos de definição de requisitos e medidas de segurança preventivas, desde as etapas iniciais da planificação dos sistemas. Este protótipo foi desenvolvido no contexto do projeto SECURIoTESIGN, o mesmo integra duas outras ferramentas que auxiliam na identificação de requisitos a partir de informações fornecidas pelo utilizador e que vão ser vir de *inputs* do protótipo aqui desenvolvido.

Esta dissertação produziu uma aplicação *web* capaz de receber os *inputs* do utilizador contendo a informação com os requisitos e recomendações do sistema e a partir do seu processamento é possível obter a modelação de sistema e de ameaças. O processo de validação do protótipo aqui desenvolvido consistiu em comparar uma análise de modelação de sistema e ameaças produzidas manualmente por um perito, com as análises obtidas por voluntários através do protótipo desta aplicação *web*, e verificar o quão correta é a análise produzida pelo protótipo. De forma geral os resultados foram satisfatórios, tendo o protótipo sido capaz de alcançar uma análise bastante correta face à produzida pelo perito. Desta forma pode-se concluir que os objetivos desta dissertação foram alcançados com sucesso.

Palavras-chave

Internet das Coisas, Modelação de Ameaças, Modelação de Sistemas, Segurança em IoT

Resumo alargado

Dado que esta dissertação foi na sua maior parte elaborada na Língua Inglesa, este capítulo serve para explicar na Língua Portuguesa de forma breve, mas algo detalhado, o trabalho elaborado ao longo desta dissertação.

Introdução

Este primeiro capítulo não só contextualiza o leitor para a estrutura da dissertação, mas também para o contexto e motivação científica sob a qual esta foi elaborada, fornecendo adicionalmente uma definição clara do problema e objetivos a serem desenvolvidos, bem como da abordagem tomada para alcançar os mesmos e das contribuições feitas neste processo, do ponto de vista académico e científico.

Enquadramento, Descrição do Problema e Objetivos

Os avanços tecnológicos da última década abriram portas à integração da Internet e outros protocolos de comunicação em praticamente qualquer dispositivo, permitindo que estes comuniquem entre si bem como com o mundo ”exterior”. A Internet das Coisas (*Internet of Things*, do Inglês e abreviado para IoT) já se infiltrou de forma discreta em inúmeras situações e contextos do nosso dia a dia, contudo, a tendência é que se torne cada vez mais relevante e o seu impacto cada vez mais visível. Dado o potencial de expansão desta tecnologia e a sua elevada aplicabilidade em cenários domésticos, comerciais e industriais, não há sinais de que vá haver um abrandamento da sua adoção nos mais diversos contextos.

Infelizmente, os progressos na área da segurança nesta área não têm conseguido acompanhar a evolução e adoção rápida desta tecnologia, pelo que uma grande parte dos sistemas e dispositivos produzidos atualmente não implementam medidas de segurança adequadas ao tipo de dados e de trabalho a que os sistemas se destinam. Isto deve-se em parte aos sistemas não serem pensados desde o início tendo em consideração requisitos de segurança adequados. É neste contexto que entra o projeto SECURIoTESIGN, que visa promover boas práticas de design e segurança ao auxiliar *developers*, mesmo aqueles com conhecimento limitado de segurança, a identificar os requisitos e medidas mais adequadas ao seu sistema.

Esta dissertação atua no domínio deste projeto, ao desenvolver um protótipo de uma ferramenta que atua como ponte entre dois outros protótipos desenvolvidos anteriormente, auxiliando *developers* nos processos de modelação de sistema e de ameaças. Este protótipo funciona a partir dos requisitos de segurança identificados pela da análise de dados fornecidos pelo utilizador relativamente à arquitetura do sistema.

O objetivo final é produzir uma ferramenta capaz de fornecer uma análise de sistema e de ameaças útil e correta para o sistema do utilizador, a partir dos resultados de inquéritos feitos previamente nas outras ferramentas.

Principais Contribuições

Tendo em conta o trabalho realizado nesta dissertação, as principais contribuições consistem nos seguintes pontos:

- Um estudo completo acerca das principais ameaças de segurança em sistemas de Internet das Coisas, assim como das suas medidas preventivas. Análise do estado da arte relativamente a outros projetos de investigação e protótipos sobre modelação de sistemas e ameaças, assim como dos protótipos relevantes previamente desenvolvidos no contexto do projeto SECURIoTESIGN;

- Análise e levantamento do estado da IoT relativamente a protocolos de comunicação, tipos de dispositivos, considerações gerais acerca de segurança e e análise das metodologias de modelação de ataques e ameaças;

- O planeamento e implementação do protótipo de uma aplicação *Web* constituída por *back-end* e *front-end*, capaz de aceitar os requisitos e guias de segurança do sistema de um utilizador, e a partir desses dados fornecer uma análise de modelação de sistema e de ameaças, completa com informações adicionais que ajudem o utilizador a compreender o motivo pelo qual cada requisito é necessário para proteger o seu sistema.

Background e Trabalho Relacionado

Este segundo capítulo destina-se a explorar o atual contexto tecnológico e de segurança relativo à Internet das Coisas. A primeira seção contextualiza o leitor acerca do estado atual da IoT, incluindo motivos que levaram à sua rápida adoção, áreas de elevada aplicabilidade assim como uma revisão dos protocolos de comunicação e categorias de dispositivos existentes.

De seguida, é feita uma análise de princípios de segurança informática para melhor contextualizar o leitor para a próxima secção, que explora os ataques (e as suas medidas preventivas) que podem afetar sistemas de IoT.

Para finalizar o capítulo é feita uma análise das metodologias de modelação de segurança mais utilizadas e relevantes atualmente, assim como uma revisão de outros trabalhos e projetos de investigação que incidem sobre as áreas de modelação de segurança em sistemas informáticos e de IoT. É também feita uma contextualização do projeto SECURIoTESIGN, visto que duas das ferramentas desenvolvidas naquele contexto foram necessárias para o desenvolvimento desta dissertação.

Proposta e Fluxos de Sistema

O terceiro capítulo elabora uma planificação dos requisitos funcionais e não funcionais do sistema, assim como do seu *design* e os fluxos de utilização pretendidos.

Relativamente aos requisitos funcionais estes consistiam em desenvolver dois protótipos de ferramentas, uma direcionada à modelação de segurança de sistemas e outra à modelação de ataques/ameaças. Os dois protótipos podem estar inseridos numa única plataforma, mas têm que funcionar e obter resultados com base nos *outputs* das ferramentas do projeto SECURIoTESIGN, nomeadamente da SECURIoTEQUIREMENTS e SECURiotPRACTICES, que são respetivamente responsáveis por obter os requisitos e boas práticas de segurança de um sistema.

Os requisitos não funcionais abordam principalmente a usabilidade da ferramenta a desenvolver. Esta deveria ser simples de usar e bastante explicativa da informação disponibilizada, para que qualquer utilizador ou *designer* de sistemas seja capaz de compreender o seu conteúdo, mesmo que tenham conhecimentos limitados acerca das áreas de segurança. Dado que o protótipo deverá ser integrado com as restantes ferramentas do projeto SECURIoTESIGN, é também esperado que todo o trabalho seja bem documentado e elaborado de maneira a facilitar a integração com outras ferramentas.

Quanto à definição do *design* do sistema, este consistiu em integrar ambas as ferramentas de modelação de sistema e de segurança numa só aplicação *Web*. Aqui os utilizadores possuem a opção de introduzir os ficheiros com as suas análises dos requisitos e recomendações ou então de selecionar um dos exemplos disponibilizados. Independentemente da opção escolhida estas análises passam por um processo de mapeamento com dois ficheiros preparados *à priori*, um com os requisitos, os ataques e recomendações para esse sistema e outro com os ataques e as suas medidas preventivas. Com este mapeamento apenas os requisitos e ataques relevantes para o sistema são selecionados e mostrados nas respetivas secções ao utilizador.

Por fim neste capítulo são demonstrados em diagramas de fluxo de dados os fluxos principais do sistema, um decompondo o funcionamento do sistema desde o início ao fim nos seus passos principais e outro que decompõe o sistema nas ações que são efetuadas pelo utilizador ao utilizar a plataforma.

Implementação

O quarto capítulo refere-se na sua integridade aos aspetos técnicos envolvidos no desenvolvimento do protótipo da plataforma, de forma a obter uma ferramenta com os comportamentos e funcionalidades descritos no capítulo anterior.

A primeira secção começa por especificar que a plataforma é constituída por dois componentes. O *front-end* que acaba por ser a parte principal, visto que é através da aplicação *Web* que o utilizador interage e obtém os resultados da análise do seu sistema; E o back-end que é responsável por fornecer dados ao *front-end*, nomeadamente os exemplos disponíveis e os resultados da análise do conteúdo dos ficheiros introduzidos pelo utilizador.

O componente de *back-end* foi desenvolvido em Java, fazendo uso da *framework* Spring e com o auxílio das ferramentas Maven (para gestão de dependências) e Swagger (para documentação das *Application Programming Interface (API)*). O *front-end* da ferramenta, ou seja, a aplicação *Web*, foi desenvolvido na *framework* de JavaScript Angular, fazendo uso das bibliotecas d3-Graphviz, Bootstrap e PrimeNG para auxiliar com elementos visuais e organização do conteúdo relevante para o utilizador.

Esta divisão do projeto em dois componentes, bem como esta escolha de tecnologias permitiu que o projeto fosse desenvolvido de forma bastante modular, facilitando qualquer modificação ou trabalho futuro.

Na secção seguinte é realizado um resumo da lógica de implementação da ferramenta, ou seja, dos passos e decisões que foram tomados de forma a conseguir implementar um protótipo que cumprisse com os requisitos. O primeiro ponto a ser explicado consiste na parte preparatória da implementação, que consistiu em produzir dois ficheiros em formato *JavaScript Object Notation (JSON)*, um contendo a informação relativa a todos os possíveis requisitos de segurança, onde se aplicam no sistema, os seus ataques e por fim os seus guias de recomendações relevantes, e o outro ficheiro contendo todos os ataques possíveis, decompostos de acordo com a parte do sistema que afetam, bem como as suas medidas preventivas. Para melhor ilustrar a estrutura destes ficheiros são fornecidos dois excertos abreviados.

De seguida são fornecidas informações acerca da organização estrutural dos projetos de *front* e *back-end*, explicando os motivos e funcionalidades de cada pacote Java ou componente do Angular.

Na terceira secção é feita uma análise mais detalhada a nível técnico da implementação do *front* e *back-end*, fornecendo excertos de código adaptados bem como imagens das partes correspondentes da aplicação *Web*. Todas as funções, serviços e componentes principais são aqui explicados de forma simplificada. É ao longo desta secção que são explicados os processos através dos quais os ficheiros com as informações produzidas pelas outras ferramentas são mapeados para os objetos utilizados dentro do projeto, e que quando filtrados com os ficheiros JSON produzem o conteúdo relevante a mostrar ao utilizador.

Para concluir este capítulo, foi feita uma análise dos desafios técnicos que surgiram aquando da implementação do protótipo. Estes consistiram em **I)** decidir como fazer o mapeamento entre os ficheiros produzidos pelas ferramentas SECURIoTEQUIREMENTS e SECURiotPRACTICES, dado ser necessário combinar a informação produzida pelas duas para mostrar ao utilizador final a análise do seu sistema; **II)** de que maneira mostrar a informação ao utilizador tendo em conta este mapeamento referido no problema anterior; **III)** como implementar o protótipo de forma a facilitar a integração com o resto do projeto SECURIoTESIGN, assim como de qualquer possível trabalho futuro. É aqui neste ponto que é realçado o papel das tecnologias escolhidas e a forma como elas contribuíram para tornar o protótipo bastante modular.

Testes e Validação do Sistema

Tendo sido discutidos todos os aspetos relativos à implementação do protótipo, é possível passar à sua avaliação e análise. Ao longo deste capítulo é discutida a metodologia de testes adotada, que consistiu na primeira parte, em definir vários cenários de sistemas de IoT a serem usados como exemplos, e a aplicá-los na ferramenta de forma a verificar se o protótipo na sua globalidade funciona bem e disponibiliza a informação da forma pretendida.

A segunda parte dos testes consistiu em selecionar um desses exemplos (neste caso um exemplo de uma casa inteligente), desenvolver as especificações do mesmo, pedir a um perito que fizesse de forma manual uma análise da modelação dos requisitos e das ameaças do sistema e por fim, pedir a um grupo de voluntários para testar o sistema preenchendo as ferramentas SECURIoTEQUIREMENTS and SECURiotPRACTICES e de seguida introduzirem os ficheiros produzidos no protótipo. No final comparar a análise obtida pelo perito com as análises obtidas pelos voluntários e avaliar o sucesso da ferramenta, ou seja, se os resultados obtidos são comparáveis aos obtidos pelo perito.

A conclusão é que o protótipo desenvolvido foi capaz de produzir resultados muito semelhantes aos obtidos pelo perito. Existem pequenas diferenças nos resultados, contudo estas devem-se a erros no mapeamento dos requisitos de segurança em relação a onde são aplicados e aos ataques que lhes dizem respeito. Estas diferenças podem ser resolvidas através de um processo mais extenso de validação e correção dos mapeamentos.

Conclusões

O sexto e último capítulo desta dissertação reflete acerca dos objetivos propostos e em que medida cada um deles foi atingido. São também apresentadas sugestões de trabalho futuro que pode ser realizado na continuação ou seguimento desta dissertação.

O primeiro objetivo, que consiste na realização de uma análise geral da segurança nas tecnologias de IoT bem como a definição de como a modelação de ataque e sistema podem contribuir para criar um sistema seguro, foi cumprido ao longo do capítulo 2. Neste capítulo foram detalhados diversos aspetos relativos ao ecossistema IoT, a segurança dentro do mesmo, metodologias de modelação, e por fim uma análise de ferramentas e projetos de investigação que se debruçaram sobre temas semelhantes.

O segundo e terceiro objetivo consistiam no desenvolvimento e implementação de dois protótipos de ferramentas para auxiliarem com processos de modelação de sistema e de ataque respetivamente. Este objetivo foi também atingido, sendo que os passos do desenvolvimento da ferramenta estão descritos ao longo dos capítulos 3 e 4. No capítulo 5 foi feita uma análise à metodologia de testes efetuada para verificar a qualidade da modelação de sistema e de ataque produzida pela ferramenta. De acordo com os resultados obtidos também este objetivo foi cumprido, já que os resultados do protótipo são comparáveis aos produzidos por um perito.

É um pouco mais difícil de avaliar o sucesso do quarto e último objetivo, que consiste na avaliação da usabilidade da plataforma por utilizadores com conhecimentos limitados na área em questão. Isto deveu-se ao facto de a situação pandémica atual ter dificultado o processo de encontrar um número significativo de voluntários para testes. Contudo, a partir do *feedback* informal obtido pelo pequeno grupo de voluntários a ferramenta é de fácil utilização e compreensão.

Fica no final definido o seguinte trabalho futuro: efetuar testes baseados em casos de sistemas físicos de forma a melhorar os mapeamentos e a ferramenta em geral; efetuar testes de usabilidade com uma amostra de utilizadores significativa, de forma a aplicar esse *feedback* para melhorar a plataforma; desenvolver exemplos técnicos para cada uma das medidas preventivas referidas na ferramenta de modelação de ataques; efetuar as modificações nesta e nas outras ferramentas do projeto SECUR IoT ESIGN, de forma ser possível integrá-las numa só plataforma; e por fim, adicionar a possibilidade de centralizar os dados dos ficheiros, de forma a simplificar os processos de análise e a permitir que os utilizadores possam ter acesso às análises de segurança dos seus sistemas sem terem que ter os ficheiros correspondentes sempre consigo.

Abstract

The Internet of Things is the broad name given to technologies that allow for any devices (known in this context as things) to communicate with each other as well as machines, applications, databases, among others in a seamless manner. This allows for devices in an environment such as in a home, a factory or a hospital, to interact with each other and even to autonomously perform actions based on the information they receive. This integration of technology in regular, everyday devices allows for the people that interact or otherwise are affected by them to have a finer degree of control over what is happening around them, allowing for this technology to improve other existing ones by improving their usefulness and efficiency.

As a simple example, in the context of a smart home, a user can be able to manually command actions or to set conditions that trigger said actions according to his preferences. This means that things such as controlling room temperature and lighting, opening doors, ordering something when it runs out or turning appliances on, can be automatically performed when the conditions approved by the user are met. In medicine for example, Internet of Things (IoT) systems allow for the creation of more effective patient monitoring and diagnostic systems as well as resource management in general, as patients could potentially carry sensors that allow for constant monitoring thus assisting in diagnostics and in emergency situations.

This last example raises an obvious and very important issue with this type of technologies, which is security. If IoT systems are not properly secured, a malicious actor could potentially access or modify private patient or hospital data as well as disable or tamper with the sensors, among other malicious scenarios that could potentially result in harm to equipment or even human lives.

Given the speed at which this technology is evolving and new systems are being created and implemented, security is often seen as an afterthought, which results in insufficient or poorly implemented security measures allowing for attackers to easily disrupt the functioning or even to steal sensitive data from the system. Therefore, it is critical to perform an adequate security analysis right from the start of the system design process. By understanding the security requirements relevant to a system, it is possible to implement adequate security measures that prevent attacks or other malicious actions from occurring, thus safeguarding data and allowing for the system to perform as originally intended.

The goal of this dissertation is to explore the principles behind system and threat modeling to be able to develop a prototype tool to assist users - even those with limited security knowledge - in the identification of security requirements, threats and good practices. Hopefully, this prototype should prove to be able to assist developers better define security requirements early in the system design stage, as well as including the correct defensive measures in the development stages. This prototype was developed in the context of the SECURIoTESIGN project, as it integrates two other tools created in its context to assist in the identification of the requirements from information provided by the user.

This dissertation produced a web application capable of handling the user inputs containing relevant system requirement and recommendations information, and then processing them in order to extrapolate the relevant system and threat modeling information. The validation process for this prototype consisted of comparing a manual system and threat analysis created by an expert, with the results obtained by volunteers using the prototype application, and verifying how correct is the analysis by the tool. The results were satisfying and the proposed objectives were successfully achieved.

Keywords

Internet of Things, System Modeling, Security in IoT, Threat Modeling

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

ACM	Association for Computing Machinery
AES	Advanced Encryption Standard
AI	Artificial Intelligence
API	Application Programming Interface
ARP	Address Resolution Protocol
C2	Command and Control
CSS	Cascading Style Sheets
DDoS	Distributed Denial of Service
DoS	Denial of Service
DFD	Data Flow Diagram
DOM	Document Object Model
DTO	Data Transfer Object
GSM	Global System for Mobile communication
HMAC	Hash Message Authentication Code
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICS	Industrial Control System
IP	Internet Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
MAC	Media Access Control
MVC	Model View Controller
M2M	Machine 2 Machine
NAS	Network-Attached Storage
NIST	National Institute of Standards and Technology
NFC	Near-field communication
PII	Personally Identifiable Information
PKI	Public Key Infrastructure
PUF	Physical Unclonable Function
REST	Representational State Transfer
RFID	Radio-Frequency Identification
SAMM	Software Assurance Maturity Model
SCADA	Supervisory Control And Data Acquisition
SoC	System on a Chip
TCP	Transmission Control Protocol
UI	User Interface
UML	Unified Modeling Language
XML	Extensible Markup Language
WSN	Wireless sensor network
6loWPAN	IPv6 over Low -Power Wireless Personal Area Networks

Chapter 1

Introduction

This dissertation describes the work that was performed in the development of a system and threat modeling tool, as the final requirement to obtain the master's degree in Computer Science. One of the main subjects of this dissertation is to find out how one can extrapolate security requirements from a dataset, as well as attack vectors and other possible threats from the analysis of user provided data regarding the system. This chapter contextualizes the dissertation and provides an overview of how the problem was tackled, what were its main contributions and lastly, how this dissertation is organized.

1.1 Motivation and Scope

Over the last decade the IoT has seen major increases in usage and has been successfully adopted and implemented across all sectors. From our homes to factories and industrial settings, to sectors as different as agriculture and healthcare, and even to our own clothes and accessories, these smart devices have become part of our day-to-day lives, and the trend is for this field to keep expanding and be used in new innovative and creative ways.

Experts believe the number of IoT devices active for the year of 2020 to be anywhere between twenty-five and thirty billion, and this number will only increase faster in the upcoming years. Several factors have contributed to the success of IoT, these will be further analyzed later on in this dissertation, however, the main contributing factors were the technological improvements which allowed for increasingly powerful devices to become much smaller, improvements to battery technologies, the development of new networking protocols, improvements and widespread implementation of Cloud technology and infrastructure and lastly the overall versatility that the Internet of Things provides.

This aggressive development and growth of the industry led to many security issues, vulnerabilities and concerns, as the development of new and adequate security measures is typically a slower process. The problem is that commercial applications and IoT systems were (and very often still are) produced with minimal concerns for security. This is obviously problematic as it can directly endanger user or enterprise data, privacy and even cause physical damages to devices and material property.

This issue can partly be fixed by incorporating security design and requirements analysis right from the system architecture and design stages, this allows for a seamless integration of proper security mechanisms and guidelines. If this process is not done from the start of the project, what usually happens is that security is only designed and added somewhere during the development and implementation of the system. In IoT, if security is rushed and not thought out completely before development starts, the system is prone to having major flaws caused by oversights, incompatibilities between algorithms, devices and communication protocols or even just glitches and other unintended behaviors.

This purpose of this dissertation is to explore the processes involved in both the attack and the system modeling processes, and from the analysis of user inputs regarding the system (provided by two tools developed in the scope of the SECURIoTESIGN project) be able to extract the systems attack vectors, security requirements, as well as identifying attack scenarios and system entry points. This information obtained should be displayed to the user in a friendly way, even for users with a very basic background in security in order for it to be easily used during the system design stage.

Following the 2012 version of the Association for Computing Machinery (ACM) Computer Classification System, the main topics of this dissertation would fall under the following categories:

- **Security and Privacy - Software Security Engineering;**
- *Security and Privacy - Security Requirements;*
- Security and Privacy - Distributed Systems Security.

1.2 Problem Statement and Objectives

As can be understood from the aforementioned motivation, the main objective of this dissertation is to perform an analysis of the IoT security environment in order to define attack and system modeling processes, and how these can be implemented in order to assist developers with defining the correct security requirements for their system. For this to be achieved, the outputs from two other tools in the SECURIoTESIGN project would be consumed, as these are the tools that provide the system information to be used and analyzed in the context of this project.

Thus, we can define the following main objectives:

1. Perform an analysis of security in the IoT sector, and define how attack and system modeling can implement this knowledge in order to create a secure system;

2. Develop as part of the system modeling process a tool to assist with this process. This tool would receive as inputs the system details obtained from other SECURIoTESIGN tools and should output the corrected security requirements, guidelines and any other relevant information regarding the security architecture of the system;

3. Develop as part of the attack modeling process another tool, which would take as input information regarding the architecture of the system, obtained from another tool which is also part of the SECURIoTESIGN project. This new tool should extrapolate and output the potential attacks and attack vectors that are relevant to the system, according to its security requirements;

4. Because these tools should be usable by even a developer with limited security knowledge, special care should be taken regarding their usability, and how clear and self-explanatory they should be.

1.3 Adopted Approach for Solving the Problem

In order to tackle the problems proposed by this dissertation the following top level execution plan was defined:

1. First, it was necessary to perform an extensive analysis and definition of security in the field of IoT. This analysis should carefully define all the relevant security requirements in this area, as well as relevant attacks and their countermeasures;

2. In a second stage, a review of the most commonly used security modeling methodologies was performed, in order to establish what the best system and attack representation would be in the context of this project. In the end, the choice was to develop the attack modeling tool following the attack tree methodology, as it provides the best balance between ease of understanding and the clarity of the information displayed. The system modeling tool followed a more traditional Unified Modeling Language (UML) diagram model;

3. The system modeling tool could then be developed. This process began by mapping the outputs from the SECURIoTESIGN tools to all the relevant system and security information obtained in the first step. From this map the security requirements, attack vectors and security considerations could be extrapolated and displayed;

4. As a fourth stage, the attack modeling tool was developed. Attack trees were built for each attack that can occur in a typical IoT system, containing the countermeasures to prevent it. Lastly, based of the extracted system requirements and architecture information the relevant attack trees are displayed by the tool;

5. Finally, the last step of the development process was to evaluate the tools, in order to check for any possible issues and to confirm the completeness of the information extracted and displayed.

Over the course of this dissertation, a more in-depth analysis of these steps will be performed.

1.4 Main Contributions

The main contributions provided by the work developed over the course of this dissertation can be resumed into the following points:

- A complete study of the main kinds of security threats in IoT systems as well as their countermeasures. A review of some experimental technologies and prototypes developed in the SECURIoTESIGN project was also made;

- A review of the current state of IoT in aspects such as applications, communication protocols, types of devices, general security considerations and finally of threat and attack modeling methodologies;

- The design and implementation of a prototype web application (complete with back-end) that allows users to upload their requirements and recommendations files from two other SECURIoTESIGN tools, in order to obtain a complete and correct system attack and threat modeling analysis. Additionally, his web application is organized and implemented in a way that allows it to easily become integrated with the remaining SECURIoTESIGN projects.

1.5 Dissertation Organization

This dissertation is organized in six chapters, consisting in the following:

1. Chapter 1 – **Introduction** – presents an overview of the main objectives, goals, as well as the general direction taken in order to achieve them;

2. Chapter 2 - **Background and Related Works** - provides information regarding security and the IoT ecosystem in order to contextualize the dissertation. The review of the state of the art is also conducted, to assess the direction of recent research in similar topics;

3. Chapter 3 – **System Proposal and Flow** – contains a description of the system outputs, objectives, requirements and the overall usage flows of the modeling tools produced in the scope of this project;

4. Chapter 4 – **System Implementation** – provides an analysis of the implementation details of the tools, as well as the technologies that were used and how they were employed during that implementation;

5. Chapter 5 - **Testing and System Validation** – discusses the system testing methodology and results in order to evaluate if the goals of the dissertation were achieved;

6. Chapter 6 - **Conclusions and Future Work** - contains reflections regarding what conclusions were drawn from the tools developed in the context of this dissertation. At the end of this chapter future work is detailed and thoroughly discussed.

Chapter 2

Background and Related Works

2.1 Introduction

This chapter will discuss the state of IoT, security and modeling methodologies since these are the main topics approached by this dissertation. Therefore, having a good basic understanding of these subjects is vital to the comprehension of the context underlying this dissertation. In order to achieve this goal, this chapter is broken down into the following sections:

1. Section 2.2 – **Background** – presents a brief context and overview of the IoT field;

2. Section 2.3 – **Security Threats, Attacks and Countermeasures** – introduces security concepts and issues that affect the Internet of Things;

3. Section 2.4 – **Related Works** – performs an overview of the current landscape regarding threat and security modeling, analyzing the most common methodologies used;

4. Section 2.5 – **Prototypes Produced in the Scope of the Main Project** – analyzes recent research and existing threat modeling software developed in the context of this research group;

5. Section 5.4 – **Conclusions** – contains the conclusions obtained during this chapter.

2.2 Background

This section provides a basic introduction to the current IoT context as well as a brief overview of some security concerns in this field.

2.2.1 Overview of IoT

The Internet of Things is the term created to describe the connection of devices (often called "Things") that communicate and interact with each other in a network. These devices are typically specialized, focusing on a very small number of tasks, for example, a temperature sensor will only be able to record temperatures and then send that data to a database node. That node then triggers a command to turn on/off an air conditioning unit. Another practical example could be a card access control unit, that when it registers an unauthorized card it sends a warning to an application in the administrators phone. Also, in an urban context there could be smart meters, they would register payments using contactless payment methods and then validate parking for a certain vehicle, warning the user when the validation is about to expire.

These are extremely simple examples of “things” interacting in a network. In reality, IoT networks can have dozens or even hundreds of devices interacting between each other and with minimal human supervision. This also raises the issue of the complexity of setting up these large networks, as it is necessary to set up the physical and logical infrastructure to support them.

With this in mind, the Internet of Things has grown exponentially over the last decade due to a number of factors, such as:

- **Development of cloud technologies** – Cloud services provided a solution to some of the main limitations of IoT infrastructures. Due to the amount of IoT devices in a network and the fact that each of them is able to produce data, it is necessary to have a backbone within the infrastructure with data processing, storage and back-end services. In an industrial and commercial setting this would require the owner of the IoT network to spend money on costly servers and labor to set up and maintain the infrastructure.

  Cloud service providers solve this issue by being the ones responsible for handling the hardware related costs and providing interfaces for clients to integrate their devices and networks in. Cloud service providers typically work with pay as you go billing methods, so clients are billed according to the amount of storage, processing power, number of devices supported, among other options.

  This means that companies can have access to the necessary back-end infrastructure, without the high overhead costs involved, while at the same time making the configuration process easier. With these commercial options becoming widely available, more companies started developing and integrating products and solutions in the IoT sector, thus leading to more research and innovation being conducted in the area;

- **Technological improvements** – Technological advancements regarding the miniaturization and the improvement of the processing and communication capabilities of components has allowed for new and more powerful devices to be created and employed for commercial purposes. One such example of a System on a Chip (SoC) is the Raspberry Pi, which is a single board computer with very good specifications for its small size. This has led it to become one of the most popular and commonly used devices for home-made and enthusiast projects.

  Advancements in energy storage solutions (namely batteries) was also a critical aspect, as this allowed for devices to be able to function in more diverse and harsh scenarios while at the same time making them more practical, without the need for excessive power cabling and all the problems associated with that. Over time batteries grew smaller, more efficient and new, faster charging methods also became commonly available, as it is the case of fast charging and wireless charging technologies.

  New communication and networking protocols were also developed. This was necessary because IoT networks can be very heterogeneous as the “things” can have very different specifications. Some devices are made to communicate across small distances (such as inside a house), while others may have to communicate across several hundred meters or even kilometers (for example in the context of smart cities). Some devices only have to be actively communicating a couple of times per hour, while others may have to do so a hundred times an hour. Battery and processing power can also be wildly different, but most "things" are usually limited in both. Optimizing communication protocols in order to save power was one of the major concerns, as battery life is usually a critical factor in many devices. According to [ASAAA17], a few examples of these protocols are:

- **SigFox** – Designed for low power objects such as sensors and Machine 2 Machine (M2M) applications. It is designed to transfer small amounts of data, so it only supports very low amounts of bandwidth. What makes it stand out is the fact that it can transmit data to a maximum range of 50 km;

- **Cellular** – Cellular communication protocols have been around for quite some time. Communication using Global System for Mobile communication (GSM)/3G/4G allows for high throughput connectivity across basically any distance on Earth, as long as there is reception. These protocols usually have a fairly high power consumption associated, so they are unsuitable for most local or M2M networks. However, they are optimal for mobile devices, as these take the most advantage of this protocol since mobile device batteries are fairly big as well as easy and quick to charge;

- **ZigBee** – is a protocol based on the IEEE802.15.4 standard, that defines the operation of short range, low-rate wireless personal area networks. These are the networks typically found in a smart-home, or in small sensor networks for example. Its usage is appropriate for networks with small, battery powered devices, with limited transmission capabilities and with a low data-rate;

- **IPv6 over Low -Power Wireless Personal Area Networks (6loWPAN)** – Is a competing protocol with ZigBee. It is also based on the IEEE802.15.4 standard, so it targets the same kind of network and devices. The main difference is that it is built on the traditional IPv6 protocol, which allows for a much easier configuration and set-up process in order for the devices to be able to connect to the internet;

- **Bluetooth Low Energy** – Another short range communication protocol. It is adequate for devices with limited battery power and that require low latency and bandwidth to transfer data. It is compatible with the standard Bluetooth (which is supported by nearly all mobile devices) so it is also the protocol most commonly used for consumer smart devices (i.e. smart watches, lamps, locks, etc);

- **Radio-Frequency Identification (RFID)** – RFID consists of two components: a powered, active element – the reader – and a (usually) passive and unpowered element – a tag. The tag can be read and written to with information by a reader (one way communication), which uses radio waves to energize the tag, so it can transmit its data. The operational ranges it operates in are typically very small, from a few centimeters in devices such as card readers, to a few meters in the case of electronic toll collectors;

- **Near-field communication (NFC)** – NFC is another very short range communication technology that allows for data to be transmitted across devices only a few centimeters away from each other. Despite working in a very similar way to RFID, NFC allows for one-way and two-way communication between devices. There are also fully passive NFC devices called tags, which can be read and in certain circumstances written to, in a similar way to RFID tags;

  - **Z-Wave** – Primarily designed for smart homes and other small networks, it allows for the wireless communication of small data packets between devices up to 10 meters, making it adequate for energy/heating/lighting control, as well as other periodic tasks. It is based on a slave – master architecture therefore master nodes can initiate and freely communicate with slave nodes, which can only reply and execute commands sent by the master nodes.

- **Business potential** – The commercial potential of IoT technologies is still far from being fully understood and explored. This is mainly due to the many areas and possible ways in which IoT devices can be integrated into a task. According to [DZS19], over the past decade research in IoT has been conducted in many fields, such as:

  - **Manufacturing** – One of the basis of the so called 4th industrial revolution is the implementation and integration of IoT devices into manufacturing processes and operations, as a way to achieve more efficient, adaptable and autonomous plants;

  - **Agriculture** – The correct integration of IoT systems allows for an increase in efficiency, productivity and a more optimized use of resources. In agriculture, processes like planting, monitoring, watering and harvesting crops can all be improved thanks to the analysis of parameters such as soil, moisture, weather and crop development, ultimately increasing yields and minimizing the effects of pests and unfavorable weather conditions;

  - **Public Service** – IoT systems have been integrated into urban services such as public transportation services, traffic monitoring, water and sewage systems and even education, contributing to the development of a city with a more efficient usage of resources and services – a “smart city”;

  - **Electronics** – New products show up on the consumer market every day, ranging from wearables, vehicles, electric and home appliances, micro controllers, etc. Allowing a user to access a large amount of data and functionalities from any computer or mobile device with an internet connection;

  - **Health** – Most IoT health applications are related to the monitoring of a persons health parameters and the subsequent analysis of this data, whether in an autonomous way by applying Artificial Intelligence (AI) or by allowing healthcare professionals to remotely examine and diagnose patients;

  - **Energy** – In the energy sector research has been conducted to optimize the distribution and usage of electricity. By having electricity meters connected to the internet the energy suppliers can optimize power distribution according to the needs of a certain area as well as due to usage/production peaks. Resulting in the concept of a "smart grid";

  - **Data mining** – Information obtained by sensors, in particular data regarding users, can be of value to companies. Thus, it is valuable for the companies that provide services and have access to that data to process it in order to develop and provide innovative products and services.

A term that has been and will be commonly used over the course of this dissertation is the term "IoT devices" or simply "devices". Given the many different domains of application and the heterogeneity of devices, according to [BFF+19], we can distinguish a set of capabilities which devices can possess:

- **Transducer capabilities** – Devices capable of interacting with the physical world, thus acting as connectors between the digital and physical worlds. There are two categories:
  - **Sensors** – Devices capable of measuring a characteristic of the physical world into some sort of measuring unit (i.e. a thermometer, pressure sensor, a camera or a microphone). Several different sensors can be interconnected in a network, creating what is called a Wireless sensor network (WSN);
  - **Actuators** – Devices capable of performing a change in the physical world, this can include a mechanical change (i.e. caused by a servo motor, mechanical arm, etc.), a change in temperature (caused by a heater/cooler), change in electric state or magnetic field, among others.
- **Interface capabilities** – Devices that can interface (communicate) with another device or some other entity. It is possible to split these into the following categories:
  - **Application interface**: The ability for computing devices to communicate with IoT devices via a device application. An example of this is a smartphone turning on a smart lamp using an API inside an application;
  - **Human user interface**: A device that can communicate/interact with a human user (i.e. touchscreens, cameras, speakers and microphones);
  - **Network interface**: The ability to communicate with a network and its devices in order to transmit or receive data. Every IoT device has at least one network interface available. Examples of network interfaces are Wi-Fi, Bluetooth, NFC, Z-Wave, among other communication protocols mentioned earlier in this chapter.
- **Supporting capabilities** – Devices that provide other IoT or computational capabilities, this includes security, privacy and device management.

2.2.2 Definition of Security in IoT

As a result of the fast growth and development in the many areas of IoT, the matters of security and data privacy have been fairly overlooked. In a commercial and business setting security is often not respected, being integrated in a poor or insufficient way somewhere along the development process, instead of being one of the key components that should be defined and integrated right from the architectural design stage.

From the analysis of [BFF$^+$19] we can conclude that there are 3 ways in which we can categorize security in the IoT context:

1. **Device security**: This includes all the security aspects related to the physical devices, meaning the device should be safe from things such as being physically tampered with or destroyed by a malicious entity. It also includes logical interactions and communications with other components or elements of the network, this means there should be ways to prevent the device from participating in Denial of Service (DoS) attacks, eavesdropping/sniffing traffic in a network, or in any way to compromise other devices;

2. **Data security**: Refers to all data that requires protection. It also refers to the concept of information security, that states that there are 4 basic data security premises:

    - **Data integrity** – consists in making sure data did not suffer any kind of unauthorized changes;

    - **Confidentiality** – data must not be made available or disclosed to unauthorized entities;

    - **Identification/authentication** – whoever receives the data must be able to determine its origins;

    - **Non-repudiation** – the assurance that an entity cannot deny the validity of authenticity of something.

3. **Individual privacy**: Personally Identifiable Information (PII) refers to data that can be used to uniquely identify an individual, this could be anything from biometrics data, a passport/ID number, date of birth, among other examples.

From each one of these categories we can identify and assess what vulnerabilities a system or device will be exposed to. This way risk mitigating actions can be implemented according to the systems requirements. Also, according to the previous author, security in IoT involves some slightly different considerations and risks compared to the regular IT sector, namely:

- **Some IoT devices can interact with the physical world in a direct way**. In an IoT network, devices such as servo motors or heating, ventilation and air conditioning systems can cause a real change in a physical property. If used in a malicious way these can cause more damage than what a regular IT system could ever do. This also means that operational requirements for performance, reliability and safety of the devices might have to be taken into consideration;

- **IoT devices often have different maintenance and management necessities.** Sometimes devices can not be remotely accessed and configured due to their location or specifications. Therefore, maintenance could be a factor to consider;

- **The security tools used in the regular IT sector can be different from those available to IoT devices**. Due to the limited specifications of IoT devices and networks, these often do not have the computational resources or bandwidth to use the same tools and cryptography algorithms used in regular IT systems. So it is necessary to adapt the system to the available security tools.

When discussing security it is also necessary to introduce the concept of Trust. There is not a consensus in security literature regarding the definition of Trust, this is due to the many different contexts in which trust can acquire a specific and different meaning. Trust is thus subjective and hard to quantify and evaluate, it can only be satisfied according to the needs for security, privacy and authentication of each individual system.

For example, if a user logs into his home banking account using a simple username/password authentication method, the bank trusts that the person performing that action is the owner of the account, because he is the only one that *should* have access to those credentials. However, the bank has no way to actually confirm if that is true and if the person logging in is not someone else who had somehow obtained the credentials and accessed the account.

2.2.3 Trust

In the context of IoT, devices have to trust the data they receive from users and each other across a network. This is why it is so important to establish device and data security measures, as well as assuring data privacy. Without sufficient trust, a system has no way to verify if it is working with data that represents the reality it is inserted into.

According to [ACH15] trust in an IoT system should be assured in these 3 levels:

- **Trust between each IoT layer**: Assurance that the communication across nodes and layers is done correctly. This means that the data received by a node from another node is correct, valid and has not been tampered with;

- **Trust regarding security and privacy in each layer**: The security and privacy protocols implemented in each layer are being followed regardless of the circumstances, so the data generated and shared is still following the reliability, integrity and confidentiality concepts;

- **Trust between the user and the IoT system**: Users need to interact with the system, therefore it is necessary for both the system to trust the users data and the user to trust the data provided by the system. Without this level of trust the implemented IoT system has failed at a conceptual level. The system also needs to account for potentially malicious or incorrect user input and actions, even if they are caused by negligence.

In order to handle and manage the issues related with trust there are Trust Management Systems, which employ and ensure that all the proper rules and protocols regarding the integrity, availability and privacy of data are being followed.

2.3 Security Threats, Attacks and Countermeasures

IoT systems, like any other computational systems, are vulnerable to malicious attacks and other actions aimed at disrupting its normal functioning or stealing data. However, in the IoT sector there are particular challenges to overcome regarding security. According to [HFH15] these are:

- **Computational restraints** – A large portion of IoT devices have very limited processing power and bandwidth due to the usage of low power CPUs and weak transmitters, and often run on proprietary systems and software. This means that commonly used security, cryptographic and communication protocols might not be compatible or even be able to run without disrupting the normal functioning of the device. Memory management is also a difficult subject due to the usually small amounts of memory available, which makes these devices susceptible to potential Distributed Denial of Service (DDoS) attacks;

- **Power restraints** – IoT devices often only have access to a limited amount of power, like that from a battery or the surrounding environment. This means that CPU and communication intensive tasks need to be minimized and extremely optimized in order to conserve power, as devices are often in hard to reach or distant locations so recharging them or replacing batteries is a highly unpractical task that should be done as less often as possible;

- **Packaging and physical access** – Due to the many different and often public and easy to access places where IoT devices can be placed and integrated, this brings up the problem that a malicious entity could potentially hijack the device and possibly steal data such as cryptography keys, change and possibly install malicious software, among other scenarios. Proper packaging and tampering prevention mechanisms should be utilized;

- **Patching and updates** – Deploying patches and updates might be impossible in some devices as they might not allow for remote reprogramming or even have the ability to integrate new libraries and code. Many devices are even produced with fixed security software and protocols so users are not able to modify them, therefore it is important for devices to be secure by design;

- **Multiplicity of devices and network protocols** – Due to the many different ranges of specifications, functionalities, network protocols and interfaces that the Things may have, it is difficult to create a security standard or protocol common to the majority of devices. This also has an impact on the scalability factor of a network, because the bigger the number of different systems that need to be connected, the more complex the network becomes;

- **Dynamic network topologies** – A basic feature of IoT is its versatility and the ability to quickly assimilate a new device into the network. This represents a problem from a security point of view, as modern security schemes can not handle fast changes in network topology. Communication from inside to outside the network can also be an issue in some circumstances, for example, if devices in a network use a non-IP network protocol and want to connect to the internet, there needs to be an interface between the two protocols (protocol translation), which represents another potential point of failure.

With this in mind, there are a number of considerations regarding data and device security that need to be taken into account when designing security systems for IoT. According to [HFH15] and [ACH15] these are:

- **Software and Data integrity** – This ensures that no malicious or accidental changes to the devices software or data have been made. However, in the case of ultra low processing power devices, the commonly used hash algorithms are often computationally infeasible due the low processing power and the energy/time cost associated with running that algorithm. Creating secure and efficient hashing algorithms suitable for these constrained devices is a work in progress, as can be seen on [Kap]. The National Institute of Standards and Technology (NIST) recently started the process of evaluating and standardizing lightweight cryptographic algorithms, as can be seen on [TM□+19];

- **Data confidentiality** – Since anyone can potentially listen to information transmitted in wireless networks, it is necessary that sensitive data is not made available to unauthorized devices. This is done using encryption, as this way only devices with the right key can access the encrypted data. Encryption is also used in a number of cryptographic processes such as keyed message digests (Hash Message Authentication Code (HMAC)) and digital signatures. Due to hardware limitations the regularly used strong encryption algorithms such as Advanced Encryption Standard (AES) cannot be used by many devices. Like mentioned in the previous point, NIST is still in the process of evaluating and standardizing lightweight cryptographic algorithms [TM□+19];

- **Anonymity** – In certain circumstances it is usefull to hide the origin of data in order to further preserve individual data privacy and confidentiality. This is particularly useful if a certain node is responsible for handling sensitive data or data from other nodes;

- **Device authentication and authorization** – Authentication refers to the ability of keeping malicious or unknown devices out of the system. All devices connecting to a network or another device should go through an authentication protocol before sharing any data. Authorization (or access control) means that only users and devices with sufficient permissions are allowed access to data or resources;

- **Redundancy** – Identifying potential points of failure in the network and implementing measures to mitigate effects in case something unexpected happens. The system should be able to handle exceptions in the expected behavior in order to maintain the availability and resiliency of the system, as well as being able to self organize up until a certain amount of failures. These measures not only allow the system to be robust and handle unexpected events (such as a node failure/high-jacking/corruption, software bugs, hardware failures, etc.) but will also allow the existing security scheme to keep functioning as it should;

- **Cloud security** – An IoT systems backbone infrastructure – databases, networking devices, cloud services, etc – should also respect the best practices regarding security and privacy, as this could represent another security point of failure with very serious implications, such as data theft.

For the purpose of this dissertation however, several other properties were considered, these being:

- **Availability and Reliability** – Availability refers to the notion that networks and devices should be resilient and tolerant to flaws, allowing them to always be available when an authorized user or device needs access to it. Reliability is a similar concept, as it requires devices and data to function consistently according to the systems design and specifications;

- **Data Freshness and Origin** – Aims to ensure that data received or being handled at the time is recent and not a copy or re-transmission of previously received data. Additionally, devices should be able to verify the origin of data being received, as a way to ensure it is not being sent by an impostor;

- **Forgery Resistance** – Is somewhat related to the previous property, the main objective is to prevent falsified data from being accepted in the system. This aims to prevent a system entity from accepting a malicious payload that is designed to resemble a payload from a legitimate device;

- **Confinement** – The goal of this property is to reduce and mitigate any effects that might be produced during an attack. This way it is possible to prevent disruptions to the normal functioning of healthy nodes or even to avoid the spread of the attack across the network;

- **Interoperability** – Allows for software, data or hardware to be usable across different devices. By standardizing data and software/hardware it is possible to more easily perform security development and auditing, contributing to a more secure system.

In an IoT network there are a number of attacks that can be done to steal data or disrupt the normal functioning of the network. From the analysis of [SM13] and [ACH15] attacks can be categorized according to several parameters. These parameters can be divided into two groups, in the first group the attacks are split either by the attacker location relative to the system, his activity or his computational power. In the second group the attacks are categorized according to the network layer they affect.

Regarding the first group mentioned, we can split attacks into the following categories:

- According to the location of the attacker relative to the target system:
  - **Outsider** – The attack is performed by a device from outside network;
  - **Insider** – The attack is done by a node or device inside the network.
- According to the degree of activity of the attacker:
  - **Passive** – In a passive attack the antagonist simply listens or eavesdrops to the communication channels used in the network. He does not attempt in any way to interfere with the functioning of the system;

    - **Active** – In an active attack the malicious entity actively modifies/transmits data, or in some other way attempts to interfere with the system and its devices.

- Considering the computational capabilities of the attacker:

    - **Mote class** – This sort of attack refers to attacks conducted from devices with computational capabilities similar to those of the devices from the targeted network, meaning the attack can be performed by a sensor, actuator, smart phone, among other examples;

    - **Laptop Class** – These are attacks performed from devices which have computational capabilities vastly superior to those of the devices in the IoT network, for instance, from a laptop/desktop computer or a server.

Regarding the possible attacks themselves, these can target the following layers of the system:

- **Physical attacks** – These are the attacks that either physically damage or modify devices and their components or directly interfere with their functioning. These attacks generally require a certain proximity to devices in the system;

    - **Node damage/destruction** – An attacker physically damages or even destroys a device belonging to an IoT network. This directly aims at disrupting or taking down the network or a portion of it;

    - **Node tampering** – An adversary physically accesses the device but instead of directly causing damage in order to destroy it, he uses it for his own gain. This includes actions such as replacing components, extracting sensitive data like cryptographic keys, routing tables or even installing software that allows him to have backdoor access to the device and its data;

    - **Signal interference or jamming** – This can be considered a type of physical DoS attack in which the adversary uses a transmitter to broadcast noise signals in the same radio frequencies used by the IoT devices, corrupting communications and thus causing them to be unreadable. Depending on the power of the transmitter used and the devices affected, this kind of attack can disturb or even completely disable the network for the duration of the attack;

    - **Node injection** – The attacker can physically deploy a malicious node on the network, gaining access or controlling the data circulating between nodes. Some network attacks that will be analyzed in the next section (namely man in the middle and spoofing/cloning) could also be considered as variants of node injection attacks, in cases where the malicious node was not originally part of the network;

    - **Sleep Deprivation/Denial of Sleep Attack** – Like it was mentioned earlier, a limiting factor in IoT networks is the energy restraints in devices. Most devices are usually in some sort of power saving or sleep mode when they are not actively functioning, only turning on to perform a certain task such as taking a measurement or communicating with another node. In a sleep deprivation attack actions that cause the device to stay awake are made in a seemingly innocent and regular way, but preventing or disturbing it from entering its regular sleep or idle mode scheduling. These actions may be disguised as legitimate or unauthenticated communications or just spammed at a certain interval for example. The purpose of this attack is to speed up the battery drainage of the device, eventually disrupting the normal behavior of the node and in certain cases the network. Due to the fact that the malicious actions or communications can be so easily concealed as legitimate ones this attack is particularly difficult to detect and prevent entirely, as even receiving an obviously fake or replayed packet that will get dropped will cause energy to be expended to receive it and read the header.

- **Network attacks** – These attacks are carried out through the IoT system network layers and do not necessarily require proximity by the malicious entity. Some types of devices such as RFID devices are particularly vulnerable to certain attacks, however, due to the short range in which RFID tags work the attackers need to be close to the tag;

  - **Traffic/Network Analysis Attacks and Eavesdropping** – A malicious entity can potentially analyze the network traffic in order to find unencrypted data or certain patterns that allow the attacker to extrapolate relevant information regarding the system or an entity. This process can also be known as eavesdropping;

  - **Spoofing/cloning** – Spoofing consists in stealing or concealing a device address in order to be able to send data from another device that is not the original or receive data the device should not have access to. A common example is the case of Address Resolution Protocol (ARP) spoofing, in which attackers attempt to update the ARP tables of devices, so that an Internet Protocol (IP) address gets associated with the wrong Media Access Control (MAC) address, causing data to be sent to the wrong device. Cloning consists in copying the data of the original device into another one, potentially copying unique device IDs or addresses. This sort of attack allows for malicious entities to send data into the system while pretending to be a legitimate device. RFID tags are usually the devices most targeted and affected by cloning and spoofing, as they are typically used in user and object authentication systems, making them more valuable targets;

  - **Sinkhole/Black hole attack** – A sinkhole attack is an attack in which a malicious node attempts to draw in network traffic. This is done by means such as advertising fake routing updates to make it appear a shorter path to certain nodes or using an antenna with better transmission power, making it a more attractive node to use for network communications. This way legitimate nodes send data through the compromised node, which then has access to all that data, potentially violating confidentiality. The attacker can also simply disrupt the network functioning by dropping all the packets that go through him or redirecting them to a certain location. In a blackhole attack the compromised node simply drops all the packets he receives instead of forwarding them to their destination. A variation of this attack (called a greyhole or selective forwarding attack) also exists, in which instead of dropping all the packets, the node allows for certain packets to go through, only dropping those for a certain device for example, making it harder to detect the disruption;

    - **Man in the middle attack** – An adversary manages to interfere with the communications between two unknowing nodes, establishing cryptographic keys to each legitimate node and becoming a sort of middleman between them. The legitimate node thinks that he is communicating with the correct node, unaware that his communications are going through a malicious node, who can decrypt and possibly modify the data which he then re-encrypts and sends to the other legitimate node. There are two types of man in the middle attacks: active and passive. In a passive attack the adversary simply has access to the communications and does not interfere or change anything in the data contained in them. In an active attack the adversary actively controls the communications between the two nodes, meaning that he purposefully alters data, drops or adds new packets, etc.;

    - **Network DoS** – In a Denial of Service attack the opponent causes a device or network to receive more traffic than it can handle, effectively causing resource exhaustion and rendering a particular node, cluster or even the network unusable for the duration of the attack. There are many kinds of DoS and flooding attacks, but they are all based on the same principle, which is abusing networking protocols in order to provoke resource exhaustion. An example of a network based DoS attack is the case of HELLO flood attacks, in which an attacker with more transmission power than the target nodes broadcasts a large amount of HELLO packets to nodes in a network, making them believe they have a new neighboring node. These nodes then reply and attempt to communicate with this node, but since it is outside their communications range this is just a waste of the bandwidth and computational resources of those nodes. Another example is the case of SYN flooding attacks, in which the attackers abuse the Transmission Control Protocol (TCP) three way handshake by sending several SYN (synchronize) requests, receiving the SYN-ACK response from the device, but then not replying with the ACK (acknowledge) response leaving the device waiting for those replies. This can potentially result in legitimate devices failing to connect to the flooded node;

    - **Sybil attack** – In this type of attack a fake or malicious node tries to impersonate or claim the identity of multiple nodes. This allows the attacker to modify the routing table of the target node(s), disrupting the regular network topology and potentially gaining access to data he should not have access to or even injecting false data, leading to situations of unfairness towards legitimate nodes;

- **Application attacks** – Attacks on the application layer target the software used by the devices, usually by exploiting some vulnerability or flaw in the software. Also, user misconduct or error are often factors in the success of these attacks;

    - **Phishing attacks** – The adversary targets users in order to obtain valuable sensitive data, such as user credentials and login information. This is achieved by tricking the user into using his private data in fake websites, devices or apps;

    - **Malware and Software Vulnerabilities** – If an attacker manages to infect/inject nodes with malicious software, he can then perform a number of tasks such as stealing data, executing malicious commands/requests over a network, perform denial of service attacks, among other malicious actions. Meanwhile, flaws and vulnerabilities in the firmware or software of the devices are another security hazard that malicious entities can exploit in order to compromise devices, violate data privacy and authenticity, install malware, perform unwanted actions, etc.;

- **Application DoS** – There are specific types of Denial of Service attacks that typically happen in this layer. These are called path based DoS (in which freshly created and possibly forged or replayed packets are sent to a distant node in an end-to-end network, causing nodes along that path to spend resources processing and transmitting it), and Sensor Overwhelming attacks (in which the attacker tries to cause sensors to trigger additional actions or measurements for example, expending power and more computational resources). The purpose of these attacks is to exhaust computational resources and network bandwidth, disturbing the normal functioning of the network or devices.

Fortunately, there is a number of preventive measures that can be implemented in order to secure systems against these threats, however, for a majority of these cases there is not a “one size fits all” solution, as there can be different variations and ways to perform a particular attack. The countermeasures implemented in a system should be adequate to the security requirements defined for the system.

- **Physical attack defensive measures**:

  - **Node Damage/Destruction** – Devices should be built with strong, hard to access packaging in order to prevent, delay and discourage physical damage. However, since with enough time and motivation all physical packaging breaks, the best preventive measures consist of preventing the attacker from gaining physical access to the device in the first place. This can involve a mix of things, from hiding or placing the devices in a secure or inaccessible location to adding surveillance and frequent maintenance checks. Other vulnerabilities specific to the device should be considered, for example, devices connected to the electric grid might be vulnerable to power outages or surges, certain sensors and cameras can be damaged using lasers, etc.;

  - **Node Tampering** – In cases of node tampering the attacker usually attempts to manipulate the hardware, software or sensitive data in the device. Measures to prevent physical access to the device like adding surveillance or placing the device in a secure location should still be taken nonetheless, and in the case an attacker can get access to the device anyway, steps to prevent access to the device circuitry and communication ports should be taken. This can involve the use of tampering-resistant hardware and packaging, disabling debugging and test access ports, as well as taking measures to secure the bootstrap loader and to prevent and detect unauthorized software and operating system changes. This mostly aims at preventing access to cryptographic keys, routing tables, or any other sensitive information, as well as to prevent the installation of malicious software;

  - **Signal Interference/Jamming** – Traditional defensive measures for this sort of attacks usually involve some process of channel surfing or frequency-hopping. Meaning that devices would switch communication frequencies following a set interval and pattern determined by them, this is the process used by the Frequency-Hopping Spread Spectrum technique. The Ultra-wide band technology is inherently resistant to jamming attacks as it operates in a wide range of frequencies, however, despite being around for many years only now it seems to be gaining traction in consumer products, so it is not quite available as a very usable alternative yet. Some experimental methodologies (i.e. Jammed-Area Mapping) propose a way to identify the nodes in the area being jammed, isolating them from the rest of the network, allowing the network and remaining devices to re-organize and not spend resources with the disabled nodes;

  - **Node Injection** – Since many IoT networks have a fairly large range and are placed across public or commercial spaces, it can be impossible to completely deny someone from being in the wireless communications range of the devices. Thus, in order to prevent a malicious individual from setting up an illegitimate node in the network, defensive measures typically revolve around having an authentication scheme to secure access to the network (meaning a device without the proper credentials does not have access to the network) and having routing rules that deny or ignore communications with unknown or unauthorized devices. Data encryption should also be applied to the communications of legitimate nodes, should a malicious node be used to spy on communications being relayed through it;

  - **Sleep Deprivation** – Due to the nature of this attack, detection and prevention is difficult and very much depends on the regular behavior of the node and the identification of abuse cases. Some pattern and intrusion detection systems can be effective in identifying malicious nodes, however the computational costs associated might not be suitable for all devices and networks.

- **Network attacks defensive measures**:

  - **Eavesdropping/Traffic Analysis** – Since the vast majority of IoT devices communicate wirelessly it is nearly impossible to detect or keep a malicious entity from potentially eavesdropping and listening to the communications. So, countermeasures against this sort of passive attack typically consist in applying padding and encryption to make sure an adversary is not able to make sense of the data being transmitted. This cryptographic process requires secret keys to be established first. The cryptographic algorithms and key distribution schemes used should also be adequate according to the device specifications and security requirements. Most communication protocols in use in IoT already support these operations as well as several lightweight cryptographic algorithms;

  - **Spoofing/Cloning** – Spoofing prevention usually relies on making sure devices and servers are indeed communicating with the real device and one of the best ways to achieve this is by using device authentication. For attacks such as ARP spoofing, countermeasures typically involve having static ARP entries, that do not allow for address updates on certain (trusted) addresses. Some software solutions also exist and can be implemented at different levels (i.e. in routers, servers, switches and certain devices), these typically involve monitoring ARP responses and known or authorized IP addresses and rejecting malicious responses. To solve problems with cloning, authentication is also the answer. It is generally impossible or very hard to clone private cryptographic key information if it is properly secured. The usage of Physical Unclonable Function (PUF)s is another viable solution, as a unique key is derived from physical properties and imperfections in the hardware of the devices;

- **Sinkhole/blackhole** – Defensive measures against these attacks usually consist in network analysis in order to detect and then avoid the anomalous areas, or in routing protocols that allow for multi-path routing or that ensure end-to-end communications and verify that traffic is bidirectional. Research has been done on different types of analysis methods however they all generally analyze network flow information in order to detect nodes with abusive behaviors, some research also suggests that mesh topology also might be more resilient to these attacks than a star or tree topology;

- **Man In the Middle** – One of the ways to begin this attack is to perform an ARP spoofing attack as this allows for the attacker to force the devices to communicate with the malicious node whilst they believe to be communicating with the correct, legitimate node. With this in mind we can consider ARP spoofing preventive measures to also be valid countermeasures for this sort of attack. Another basis for the defense against this type of attacks, is to use device authentication mechanisms that ensure the device is communicating with the real one. An example of such mechanism is the usage of a Public Key Infrastructure (PKI), as this would allow for the validation of signatures, confirming the identity of devices. The problem with this solution is that certificate management can be problematic due to the computational and bandwidth restraints in typical IoT devices;

- **Network DoS** – Solutions for this group of attacks depend specifically on the kind of DoS attack being executed, for the case of the common attacks mentioned, the countermeasures for the HELLO flood attack mainly consist in verifying the bidirectionality of traffic (as the attacker often uses a powerful transmitter in order to stay out of range of the target nodes) and in authenticating devices. For the case of SYN floods, countermeasures can involve filtering (removing the ability for the attacker to send requests from spoofed addresses), increasing the size of the ACK backlog, decreasing the time the device will wait for an ACK message, using a First In First Out model when the backlog is full (recycling the oldest entry waiting on the ACK message), the usage of SYN cache or cookies or finally making the client solve a puzzle on each connection;

- **Sybil** – This kind of attack can also be solved by implementing a good device authentication scheme in a network or by validating communications and devices trying to pair with each other, this should preferably be done via cryptographic methods. However, due to the computational requirements involved in these authentication schemes, some alternative methods based on network and device behavioral analysis have been designed in order to identify fake nodes.

- **Application attacks defensive measures**:

  - **Phishing** – The best countermeasure for this sort of attack is to educate users to be able to identify and report phishing emails/apps/websites. This process can be easier in a commercial company, as the usage of phishing email and content filters is also a valid defensive method. Additionally, workers can be expected to have some degree of training in order to prevent falling for phishing scams. Regarding clients and end-users, awareness campaigns should exist to alert them to the potential hazards of phishing, as well as providing tips on how to identify and avoid phishing attempts;

  - **Malware and Software Vulnerabilities** – The best way to prevent malware and the exploitation of vulnerabilities is to apply patches and update software in a timely manner. Depending on the device this might be a complicated task, as some devices may not even allow for wireless updates to happen and might be located in a hard-to-reach location. However, failure to keep devices updated can very well lead to abusive behavior by users or malicious entities, which depending on the severity of the vulnerability can lead to data breaches, shortening of machinery lifespan, theft, etc. Another defensive measure that can be useful is having adequate user and device access control lists, as this would limit them from having access to data or to perform system changes that they should not be able to;

  - **Application DoS** – Countermeasures for path-based DoS attacks typically consist in detecting and stopping the propagation of illegitimate and replayed packets. Several of these detection methods have been proposed and analyzed but due to the memory, processing power and bandwidth constraints in IoT devices as well as the high device heterogeneity and routing path updates, there is not an optimal solution that can be applied across most networks. There is a lightweight and fairly resilient model that has been proposed which uses one-way hash chains to verify if a packet is legitimate and has not been replayed. Having a good authentication scheme is also a good preventive measure as nodes can simply drop traffic from unknown nodes. Regarding sensor overwhelming attacks, these can be prevented or at least mitigated by adjusting sensor sensitivity, limiting the rate of readings or actions possible as well as applying limits for bandwidth expenditure or communications to other nodes.

2.4 Related Works

This section contains a review of the state of the art regarding threat modeling methodologies and related research works.

2.4.1 Attack and Threat Modeling

First, it is important to distinguish a threat from an attack. A threat to a system refers to a hypothesis of something bad/harmful happening. An attack is the actual action(s) associated with the realization of a threat.

Threat modeling is a security engineering task that refers to the practice of analyzing a system, its components and interactions, in order to identify all the security threats and vulnerabilities that the system might have. A threat being an external **event** that could potentially cause harm to a system, its nodes, or data. While a vulnerability refers to a **weakness or flaw** in the system that could potentially be exploited.

An attack model is very similar to a threat model, the main difference being that instead of focusing on identifying problems and the respective defensive measures, it focuses on how to accomplish a threat. This means that an attack model focuses on defining the steps required for an attacker to reach its goal.

From this identification of threats and vulnerabilities, the system designers can decide which security mechanisms are to be implemented, depending on the level of security the system requires and the security mechanisms the system can handle. At the same time, a risk management strategy should also be created, in order to serve as a basis of procedures in case of an attack. This would define what actions to take in case of attack, what parts of the system are critical and more likely to be targeted by the attacker next, and how to safeguard data. This analysis should be preferably done in the design stages of the system, as it typically is more costly to implement security mechanisms or address security risks in already implemented systems.

The modeling process can typically be decomposed into the following steps [MLY05]:

1. **System categorization** – To begin the modeling process the security analysts needs to have a clear understanding of the system at hand. This means that all the devices, protocols used, and how the devices behave and interact with each other within the system needs to be analyzed. In order to accurately represent the system modeling languages or diagrams are typically used, the most common being DFDs which can map all the system nodes and data shared between them;

2. **Identification of assets and access points** – An asset can be defined as any resource (whether it is physical or digital) which can be of value to an attacker. An access point can be anything from open ports, software or hardware with vulnerabilities, wireless networks without authentication, among other possibilities, and it represents the way through which the attacker is able to enter the system and eventually gain access to the assets. With these definitions in mind, the analysts can come up with the answers for security related questions, such as:

    - Who might be the potential adversaries to the system;

    - What do the attackers want to achieve;

    - What can be done to stop or mitigate an attack;

    - What sort of information regarding the security measures of the system do the attackers know.

    These are some basic questions that will provide the analysts a good starting point regarding the direction they should take when specifying the security necessities of the system;

3. **Identification of threats** – From the knowledge obtained from these steps the actual threats to the system can be defined. Threats and assets are always closely related, as each threat always targets one or more assets. This step analyzes which of the attacks defined in the previous section can be of concern and how can they be executed in the system, in order for an attacker to compromise an asset. By correctly identifying how to stop these threats, security requirements can be defined.

The first two parts of this process are generally consistent for all threat models, as a good knowledge base of how the system works, its vulnerabilities and what an attacker might try to do in order to achieve a goal are universal objectives. However, the part regarding the identification of threats is where the threat modeling itself takes place.

When discussing threat modeling it is also important to have methods to properly evaluate risk. Some threats pose a higher risk to the system than others, so it is necessary to prioritize threats according to several factors such as: ease of execution, potential damage, amount of affected devices/users/services, quality of mitigation measures, among others. High risk threats should obviously be treated with more concern than lower risk ones.

There are several methods to analyze and define threats however, for the purpose of this dissertation only 4 were analyzed, as they represent the most commonly used, practical and complete methodologies.

2.4.1.1 Attack Trees

Attack trees are an excellent way to analyze threats and the steps required for an attacker to be successful. It follows a tree/graph like structure in which the root node represents the threat or the final objective of an attacker. The branches and sub-branch nodes then sequentially represent the many steps an attacker has to take in order to be successful. In these trees logical conditions can also be represented, by default it is assumed that different branches at the same level are OR conditions. AND conditions can also be used, generally by using an arrow over the branches that are required, as can be seen on figure 2.1 represented by the arrow below the root node. Attack trees can be customized to better fit the necessities of analysts, for example by adding the necessary defense measures next to a branch, making a horizontal tree, adding or color coding branches according to certain conditions, associating costs with nodes, adding logic gates, etc.

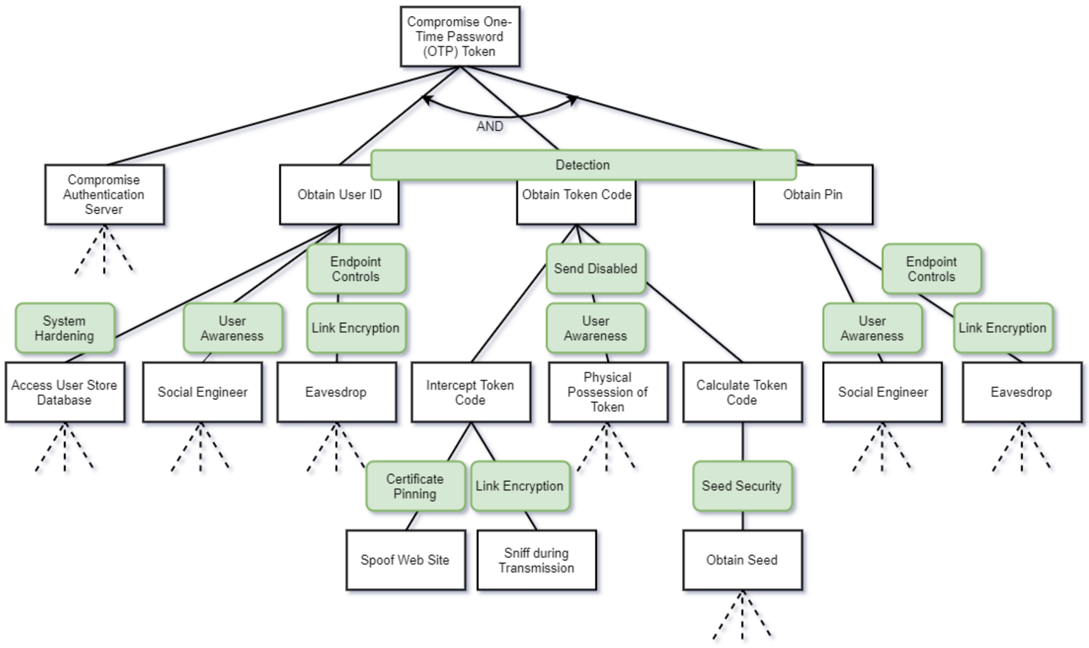


Figure 2.1: Example of an attack tree with logical conditions and color coding to differentiate steps from counter measures. Adapted from [MF19].

It is also possible to represent attack trees in a more simple, text based manner, which is in outline form as seen in listing 1. Much like in graph form, there can exist variations in the appearance and features of the tree, however the basic principle still applies: a root node that represents a threat, followed by branches that represent the necessary actions for a threat to be fulfilled. Branches and sub-branches can also be re-utilized across different trees, allowing for the reuse of elements that have already been analyzed.

Listing 1 Example of an attack tree in outline form. Adapted from [BFM04].

```
Goal: Gain unauthorized access to account
OR
1. Obtain the password
   OR
   1.1. Listen during the transmission in an insecure channel
   1.2. Brute force it
   1.3. Social Engineering
2. Bypass the access control
   OR
   2.1. SQL injection
   2.2. Session high-jack
   2.3. Insecure dependency
3. Modify the credentials directly
   3.1. Get access to the database to change them directly
      OR
      3.1.1. Gain internal access to the Database
   3.2. Social Engineering
```

2.4.1.2 Diamond Model

Diamond models were originally created in order to characterize network intrusions; however, they have been used for other security purposes in Information Technology security. This type of model describes intrusion events by taking the form of a 4-pointed graph in which each corner describes a key component of the attack. These components (also called core features) are the adversary, capability, the victim and the infrastructure. This relationship logic can be translated as an adversary that deploys a capability over an infrastructure against a victim [CPB]. The author describes these components as:

- **Adversary** – The entity responsible for using a capability against a victim in order to achieve a goal. For most events this component cannot be defined *a priori* because the attacker and his capabilities are usually unknown to the victim until the attacker manifests himself;

- **Capability** – Defined as the tools and techniques used by the adversary, this usually refers to exploits or flaws that can be abused in order for the adversary to achieve a goal (stealing data, disrupting the service, eavesdropping, etc.);

- **Victim** – The target of the adversary. Vulnerabilities or flaws in the victims' infrastructure will be the targets for exploits in order to achieve a goal;

- **Infrastructure** – Physical and logical communication structures used by the adversary in order to deliver capabilities, maintain control of them, extract data, among other nefarious possibilities. This infrastructure can be either fully owned by the adversary (called type 1 infrastructure), controlled by a knowing or unknowing intermediary (type 2 infrastructure) or a service provider.

The same author also defines a set of secondary features (called meta-features) which can be used to order events and represent more data about actions in the model. These meta-features are timestamps, phases, results, directions, methodologies and resources. In this modeling scheme there are edges between the corners that represent the natural relationships between the components. This overall structure can be seen on figure 2.2.

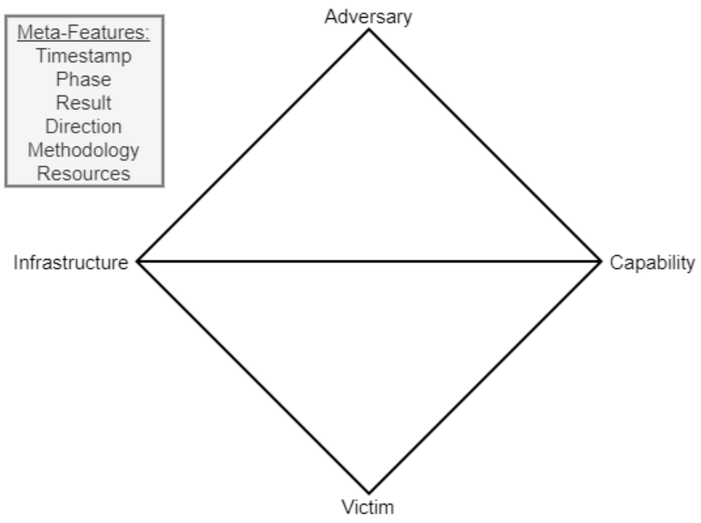


Figure 2.2: Structure and components of a diamond model. Adapted from [CPB].

For this type of model there are also events which, according to the author, can be described as *"an adversary taking a step towards an intended goal by using a capability over infrastructure against a victim to produce a result"*.

When formally using this methodology events are defined as an n-tuple of both the core and meta-features, as well as any other additional features the analysts find relevant (i.e. a host name, IP address or software exploited), this however, is also valid for the rest of the features, as the amount of characteristics and values associated to each core and meta feature can vary according to what the analyst needs, allowing for a more complete description of the system.

Also, relevant for this type of scheme is the definition of analytic pivoting, it consists in analyzing an element of a core feature of the model in order to find other related threats and vulnerabilities. For this to be successful it is important for the analyst to have a clear understanding of how the system works, how the elements interact with each other, their limitations, what can an external entity deduce about the system, among other factors more specific to that system in particular. Over in figure 2.3 it is possible to analyze an example of an attack analysis using this methodology.

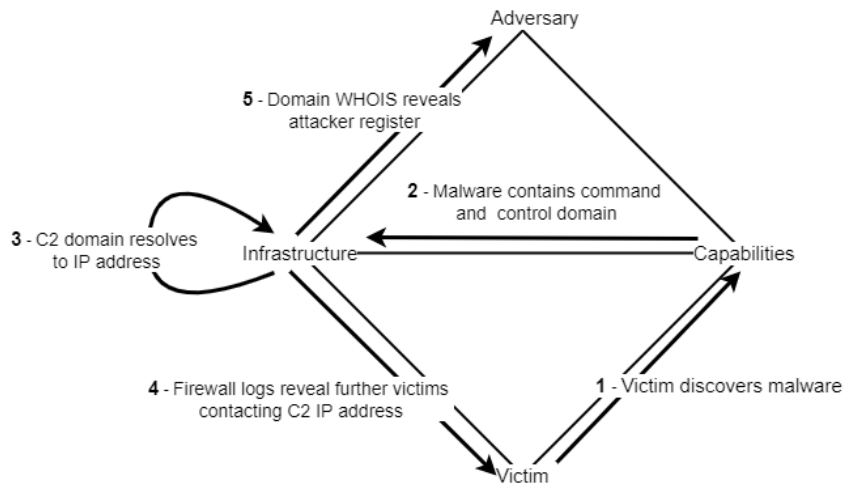

Adversary
5 - Domain WHOIS reveals attacker register
2 - Malware contains command and control domain
3 - C2 domain resolves to IP address
Infrastructure
Capabilities
4 - Firewall logs reveal further victims contacting C2 IP address
1 - Victim discovers malware
Victim

Figure 2.3: Practical example of the usage of a Diamond Model. Adapted from [CPB].

2.4.1.3 Kill Chain

The kill chain or cyber kill chain model is threat model published by Lockheed Martin that describes a successful attack as a series of seven phases chained together. If one of those phases fails or is stopped then the rest of the attack fails, this is akin to an actual chain, where if a link fails then the links after that point of failure will be useless.

The seven stages that make up this modeling process are:

1. **Reconnaissance** – The attacker gathers information about the system that is going to be attacked. This includes actions such as studying the devices employed in a system, their vulnerabilities and limitations or identifying and gathering information about administrators and users with special permissions. Phishing schemes and social engineering are also commonly used here;

2. **Weaponization** – The attacker develops some sort of malicious payload or device (i.e. malware or a malicious node) capable of performing some action in the targeted system. These payloads often exploit old or newly discovered (zero-day) vulnerabilities in devices or software used by them in order to disable security measures, avoid detection, escalate privileges, or any other malicious and unauthorized action;

3. **Delivery** – The payload is sent and delivered to the system. This can be done in several ways, such as sending an executable file disguised as something a user would use (i.e. a PDF, a text or a configuration file), physically modifying a node and introducing it into the network, injecting malicious network traffic, among other creative methods;

4. **Exploitation** – This step consists in the execution of the malicious code in a device. This step often includes human action such as a user opening a malicious email attachment file. However, it can also be done without human interaction via exploit kits which are toolkits that scan the device for possible exploits, and then abuse them;

5. **Installation** – The attacker installs something such as a backdoor, a remote access trojan or some other way to be able to remotely gain access to the device;

6. **Command and Control (C2)** – In this step the necessary networking tasks have been completed, allowing the adversary to manipulate and interact with the device as well as executing actions. A communication channel is also typically set up, allowing for the adversary to perform new commands;

7. **Action on Objectives** – The attacker is finally able to fully achieve his goal, whether it is stealing data or violating data privacy, disrupting the regular functioning of the system, gaining access to a back-end service, use the compromised system as a hop point in order to compromise other devices, etc.

Attacks should preferably be stopped before reaching the exploitation stage, as they are substantially easier to stop before that stage. Once an adversary manages to manifest himself in the system after the exploitation stage, it becomes much harder to stop and control the damages.

The authors of this methodology [HCA] also introduce the idea of the *Course of Action Matrix*, that consists in a table in which countermeasures are displayed and organized according to 6 categories and what step of the chain they apply to. Table 2.1 represents a generic course of action matrix with some basic countermeasures for a certain attack on a system. In a real world setting these tables and measures are adjusted according to the actual kill chain threat being analyzed.

Phase	Detect	Deny	Disrupt	Degrade	Deceive	Destroy
Reconnaissance	Web analytics	Firewall, ACL				
Weaponization	NIDS	NIPS				
Delivery	Vigilant users	Proxy filter	In-line AV	Queuing		
Exploitation	HIDS	Patches	DEP			
Installation	HIDS	chroot jail	AV			
C2	NIDS	Firewall, ACL	NIPS	Tarpit	DNS redirect	
Actions on Objectives	Audit log			Quality of service	Honeypot	

Table 2.1: Example of a course action matrix with some basic counter measures. Based on [HCA].

2.4.1.4 STRIDE

STRIDE is a modeling process originally developed and used by Microsoft, it categorizes threats into six categories, getting its name from their initials. The six categories are the following:

- **Spoofing** – Refers to an adversary attempting to impersonate some user or node, attempting to violate the property of authentication;

- **Tempering** – The attacker tries to breach the integrity or access control properties. This is done by performing modifications to the software or hardware of a node;

- **Repudiation** – It allows a malicious user or node to deny having performed an action;

- **Information disclosure** – This refers to the fact that data should not be exposed to unauthorized entities. It violates the data confidentiality property;

- **Denial of service** – The attacker attempts to disrupt the normal functioning of the system by overloading one or more nodes with traffic or computations, exhausting its resources and causing the nodes to become unavailable. This targets the availability property;

- **Elevation of privilege** – A user or node attempts to gain access to data, services or devices to which he lacks permission to access. Targets the authorization property.

In order to perform threat analysis with STRIDE several steps have to be taken[SWJ13]:

1. **Create a DFD representative of the system**, decomposing it into logical and structural components. DFDs use a standard set of four symbols, however for the context of threat modeling there exists an additional symbol for trust boundaries. These symbols can be seen on 2.4 along with some alternative representations. This model should represent the overall flow of data and component interactions of the system, however, for some systems a more refined model needs to be developed. In these cases the analyst simply needs to repeat the process for that particular area of the system, until the model is representative enough.

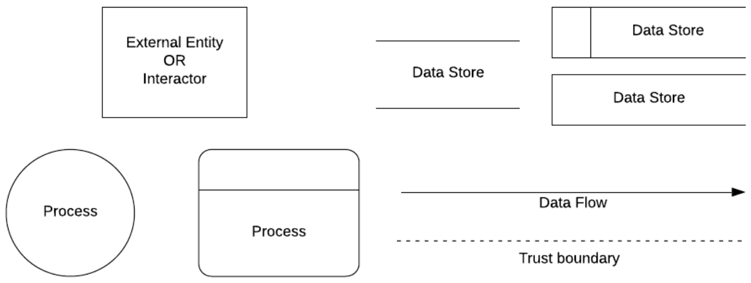


Figure 2.4: Elements and alternative models of a DFD.

2. **Mapping the DFD elements to threat categories**. This means that the analyst should build a table mapping the elements that were defined in the DFD with the respective threat categories that are relevant to it. As an example, this can be seen on the table 2.2. Do note that this process should be done for all elements of the system;

Element	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privilege
Data Flows	X				X	
Data Stores	X	X		X		
Processes			X	X	X	X
Interactors	X	X	X			X

Table 2.2: Simple example of a generic table mapping elements to threat categories.

3. **Extract, deduce and document threats**. From the analysis of the threat table, the possible attacks and vulnerabilities that each element is exposed to are deduced, inspected and documented. Although proper documentation of the vulnerabilities and threats found is not a mandatory step, it is a good practice, as this will make future security reviews or incidents easier to analyze. This documentation process is often done using abuse cases, reports, diagrams or any other tool the analyst finds relevant.

2.4.2 Related Investigation and Commercial Products

Extensive research has been conducted regarding threat and attack modeling but the vast majority was in the context of regular IT systems. Due to the significant overlap between a regular IT system and an IoT system many of the principles apply to both, however, due to the constant evolution of IoT devices, protocols and software this is quickly not becoming the norm. This required new research that focuses only on attack and threat modeling in the IoT ecosystem to be conducted, some examples being:

- **Bugeja et al.** [BVJV19] explore some currently available security models and methodologies and introduce their own, which is based on the Software Assurance Maturity Model (SAMM), that helps organizations develop and implement the most adequate security strategy according to the type of risks it faces. Their model is based on 4 components, namely governance (software should be designed with security in mind), construction (risk and threat modeling requirements should be implemented), verification (there should be proper code, device and network testing) and operations (how products are released commercially and the kind of support they get);

- **Frustaci et al.** [FPAF18] survey the most prominent threats in IoT, analyze trust and compare the concepts of security in IoT compared to a traditional IT system. A more in depth analysis of threats and solutions to communication protocols used in IoT is also done, providing a methodology for evaluating security risks. This is a very important subject as there is not a set of standards for communication protocols. This high heterogeneity in protocols, software and devices results in yet another issue when dealing with security because it is impossible to create a “one size fits all” solution;

- **Mariona et al.** [RMHK$^+$16] focus on supporting security decision-making for IoT Supervisory Control And Data Acquisition (SCADA) systems used in Industrial Control System (ICS), namely to control critical infrastructures and automated processes. It analyzes a tool called “C-SEC” which is based of 3 parts: Evaluation (analyze available technologies in order to find one whose functionalities better suit the target system), metrics (criteria to evaluate these technologies) and framework (component to discuss tools and new or past evaluations and metrics);

- **George and Thampi** [GT18] developed a 3-step graph based framework: in the 1st step occurs the mapping of the network topology and configuration, analyzing attack initiation points, target nodes or objectives (called sink nodes) and device vulnerabilities and possible interactions between devices. In the second step device vulnerabilities are mapped as nodes in the graph, with the edges between them representing vulnerability dependencies. The third and final step consists in analyzing the graph, calculating the best routes an attacker would take in order successfully achieve an attack (the best paths to go from an initiation point to a sink node). For this process, the authors define a metric to calculate threat according to the vulnerabilities exploited and the number of hops (vulnerability nodes) an attacker would take. This way it is possible to determine what nodes are more problematic and susceptible to attacks, according to the amount of exploits in which they appear and how serious they are;

- **Hassija et al.** [HCS$^+$19] perform a survey on current critical areas of IoT, an overview of the security issues associated with IoT technologies and the improvements and enhancements required to overcome many security issues in this field. The authors then analyze solutions and problems associated with the introduction of blockchain, fog computing, AI and edge computing in the field of IoT security;

- **Liu et al.** [LZL$^+$15] propose a dynamic defense architecture for IoT based on 6 components, these being: active defense – a component that implements traditional security measures, directly confronting and preventing the most common security threats. This component implements a defense strategy library that contains all the security measures and can be expanded to include new ones in order for the framework to be able to secure a system against new threats. A threat detection component, that finds new threats and updates a security threat library that aggregates possible system threats. A security warning component responsible for logging and warning the proper entity once new threats are found. A security response system, that implements measures to be taken in case of a security event, based of a response strategy library. A security recovery system to fix and mitigate disruptions or damages caused by an attack. And lastly a defense assessment component which is responsible for evaluating and reforming ineffective defense strategies, updating the defense strategy library;

- Machine learning has also been applied in the context of security in IoT, mainly as a way to detect intrusions. **Cañedo and Skejllum** [CS16] use machine learning to detect anomalies such as the injection of false data or man in the middle attacks at the IoT network gateway, with very good accuracy. Similarly, **Hodo et al.** [HBH^{+}16] implemented a denial of service attack detection system using artificial neural networks, which learns from the regular network traffic patterns in order to detect when the network is being abused.

Commercially, there are a few commercial as well as open source threat modeling tools available for general use in the information technology sector such as **Irius Risk**, **Microsoft Threat Modeling Tool**, **PyTM** and **OWASP Threat Dragon**. In general they all implement the same basic functionalities, such as system modeling support, automatic analysis of threats and countermeasures according to security policies, security requirements or even the tracking of threats and countermeasures across all stages of the software development life cycle.

2.5 Prototypes Produced in the Scope of the Main Project

This dissertation was developed within the scope of the SECURIoTESIGN project. The main goal of this project consisted in developing a comprehensive framework of prototype tools in order to assist in the identification and mapping of security requirements and technologies, attack and system modeling, as well as helping with test specifications and the generation of high quality documentation and auditing. The ultimate goal of this framework is to provide tools that allow for IoT systems to be developed with security as a consideration right from the system design stage all the way up to the testing stage.

However, from the several existing sub-projects, only the two discussed in the next subsections were relevant for the context of this dissertation.

2.5.1 SECURIoTEQUIREMENTS

The SECURIoTEQUIREMENTS tool was developed to assist in the identification of security requirements. It receives user data regarding his system architecture, generating a report with not only the necessary system requirements, but also with a taxonomy of attacks and vulnerabilities that the system is exposed to, should those requirements not be implemented adequately.

2.5.2 SECURIoTPRACTICES

The SECURIoTPRACTICES project produced another tool that receives system architecture information from the user, but then presents a report containing a set of good practices that respect and implement the necessary security measures to secure applications based of the Cloud/Mobile ecosystem. This outputted report is also obtained from a user questionnaire that contains questions regarding several aspects of the system architecture.

2.6 Conclusion

Over the course of this chapter, an analysis of the security context in IoT was performed in order to provide the reader with a good basic understanding of several factors such as: the current landscape of the IoT ecosystem and what security issues it currently faces, what security modeling methodologies exist and what research is being conducted in the field. Finally, an overview of the relevant sub-projects of the SECURIoTESIGN was presented, as these will play a role in the development of the prototype developed for this dissertation. The following chapter approaches the system design of this prototype, namely its requirements and how the design process was developed in order to adequately fulfill said requirements.

Chapter 3

System Proposal and Flow

3.1 Introduction

This chapter provides a more theoretical overview on how the tools of this dissertation were developed. The purpose of this chapter is to provide the user with a clear understanding of how the goals of the system are achieved, how was it designed, what are its inputs and outputs, among other considerations.

This chapter is organized in the following subsections:

1. Section 3.2 – **Requirements of the Tools** – this section presents an analysis of the expected functional and non-functional requirements of the tools;

2. Section 3.3 – **System Design and Purpose** – in order to fully understand the design principles for the prototype, this section provides a breakdown of the requirements, system flow and what are the expected usage paths by the user.

3. Section 3.4 – **Conclusion** – section summarizing the main points from this chapter.

3.2 Requirements of the Tools

The purpose of this chapter is to define the functional and non-functional requirements needed to fulfill this dissertation project.

3.2.1 Functional Requirements

This project must produce two security tools: an attack modeling tool and a security modeling tool. These tools do not necessarily have to be separate and can be integrated in the same platform.

The attack modeling tool should be able to receive as inputs information regarding the system specifications, architecture and security requirements. Some of these parameters should be preferably be processed using some inference mechanism in order to infer on possible attack vectors and threats, thus allowing for changes or additions to the security requirements as well as performing adaptations to the attack model.

The system modeling tool should receive as input the system specifications in order to obtain the system security requirements, and from those display a set of security practices that allow for that requirement to be fulfilled. This tool should create a model of the system architecture and its components, exposing what requirements need to be applied to what components of the system.

Regarding the inputs and outputs, both the attack modeling and the system modeling tool should receive as inputs the outputs provided by the SECURIoTEQUIREMENTS and SECURiotPRACTICES tools. Excerpts of these files can be seen on listing 2 and listing 3 respectively.

Listing 2 Abridged example of a requirements file produced by the SECURIoTEQUIREMENTS tool.

```
#  SECURIoTEQUIREMENTS
   The  **SECURIoTEQUIREMENTS** is a custom made program. It helps in the
   creation of IoT tools with security by providing the security requirements
   before the creation of the tool It is part of the outputs of project
   Project SECURIoTESIGN, with funding  by FCT-Fundação para a Ciência e
   Tecnologia (Portugal) through the  research grant SFRH/BD/133838/2017.
## License
   Developed by Carolina Lopes and Pedro R. M. Inácio Department of Computer
   Science Universidade da Beira Interior Copyright 2019 Carolina Lopes and
   Pedro R. M. Inácio  SPDX-License-Identifier: Apache-2.0

# The Requirements to develop your tool are:

---

## Confidentiality

The property that ensures that information is not disclosed or made available
to any unauthorized entity. In other words,personal information cannot be
accessed by an unauthorized third party.
This requirement is applied were the information is stored.

---

## Integrity

Is the property of safeguarding the correctness and completeness of assets in
an IoT system. In other words it involves maintaining the data consistent,
trustworthy and accurate during it life-cycle.
This requirement is applied in the Cloud.

Not addressing this requirement may lead to vulnerabilities explored by
attacks such as:

  *Sinkhole Attack:
   In this attack some nods are made more attractive than others by tampering
   with the routing information, when arriving to the sinkhole node the
   messages may be dropped or altered.

  *Malicious  code  injection:
   In  this  attack  the  perpetrator injects  malicious code in the system
   to gain access to information or even to gain control of the entire
   system.
(...)
```

Listing 3 Abridged example of a practices file produced by the SECURiotPRACTICES tool.

```
# Final Report

| Architecture -----------------| Web Application ; Web Service ; Desktop Application ;
| Has DB -----------------------| No
| Type of data storage ------| N/A
| Which DB--------------------| N/A
| Type of data stored -------| N/A
| Authentication--------------| Username and Password
| User Registration ----------| Yes
| Type of Registration ------| The users will register themselves
| Programming Languages -| Python ; Java ;
| Input Forms -----------------| Yes
| Upload Files -----------------| Yes
| The system has logs -------| No

## Authentication

**Authentication** is the process of verification that an individual, entity or website is who it
↪   claims to be. Authentication in the context of web applications is commonly performed by
↪   submitting a user name or ID and one or more items of private information that only a
↪   given user should know.

**User ID's**
Make sure your usernames/userids are case insensitive.
(...)

**Password Length**
Longer passwords provide a greater combination of characters and consequently make it more
↪   difficult for an attacker to guess.^^IMinimum length of the passwords should be enforced by
↪   the application.
(...)
```

Since these outputs are saved as simple text files in the machine where the tools are executed, the prototype produced in the context of this dissertation will have to be able to directly import the files in order to process them. However, in the near future, a platform in the scope of the SECURIoTESIGN project is being produced, that aggregates all the other tools into a single platform. Thus, in order to facilitate the future integration of this tool with the rest of the platform, the tools produced in the context of this dissertation should also be able to import or read these results via the internet (by accessing a database or a web service, for example).

No outputs are expected to be produced besides the tools or platform that performs and displays the results of the analysis, however, should any be produced they should be standardized in order to ensure they are easily convertible and portable, meaning that portable file types such as Markdown, JSON or Extensible Markup Language (XML) should be preferred.

3.2.2 Non-functional Requirements

Regarding the non-functional requirements, there are no limitations regarding the programming languages, libraries or other software from which the system must be developed. There are also no limitations or requirements regarding minimum hardware requirements or the platforms in which it has to be supported. However, there is preference in the creation of a web based application as this will allow for the prototype to be more easily accessible by the user and across devices, eliminating the need for an installation or any other set-up process.

Despite not existing any performance limitations or requirements *per se*, the tool is expected to be able to execute and perform its tasks in a timely manner, meaning that it should operate within a reasonable time frame.

Since this project is supposed to be integrated with the rest of the SECURIoTESIGN project, some care should be taken with the design and implementation of the system, in order to facilitate future integration or development with the rest of the projects. Therefore, following a modular approach when developing the prototype is highly recommended.

These tools should be straightforward and simple to use, as well as fairly explanatory of security terminology in order to be easily usable by any developer, even those with a more limited knowledge of computer security mechanisms and concepts. Documentation regarding the development and inner workings of the tools should also be developed and it must be complete, following good documentation practices.

3.3 System Design and Purpose

In order to achieve the main goals of this dissertation, a web application was developed to process the outputs produced by the SECURIoTEQUIREMENTS and SECURIoTPRACTICES tools, displaying all the relevant system security requirements and threat information to the end user. This web application can be split into two main components: the system modeling component and threat modeling component. This approach allows for the implementation and development of the main features in a modular way.

In the SECURIoTEQUIREMENTS project, a tool was developed that asked the developers a number of questions regarding the architecture of the system. From these answers, the security requirements for the system are produced as an output.

Meanwhile, the SECURiotPRACTICES project produced a similar tool, that questioned the user regarding the architecture of the system but instead outputted a report with guidelines for the developers to follow, in order to correctly implement the security requirements of the IoT system in question.

For the context of this project it was necessary to implement the results obtained from both these tools in order to be able to extract more information regarding the system architecture and its requirements, as well as information regarding the attacks that might affect the system and directly breach its security requirements. Thus, it was necessary to map the outputs of both these tools, matching the security requirements and vulnerabilities with the proper security guidelines. It was also necessary to provide information as to what parts of the system each requirement would need to be applied to, this means it was required to extrapolate security requirements from the system architecture analysis provided by the tools.

To summarize, the user starts by filling the questionnaires in the SECURIoTEQUIREMENTS and SECURiotPRACTICES tools. The outputs of these tools are then uploaded by the user into the back-end of the platform using the web application. In the back-end the files are parsed and if correct, the parsed output containing the requirements and guides are sent back to the web application, where they are mapped to a requirements and attacks JSON file and all the relevant information is displayed to the user, which can finally make full use of the platform.

Having performed this review of the overall system functioning, we can create and analyze the DFDs regarding the main system and user flows. The main system flow (see figure 3.1) essentially represents what was said over this section, making a high level representation of how the system functions from start to finish.

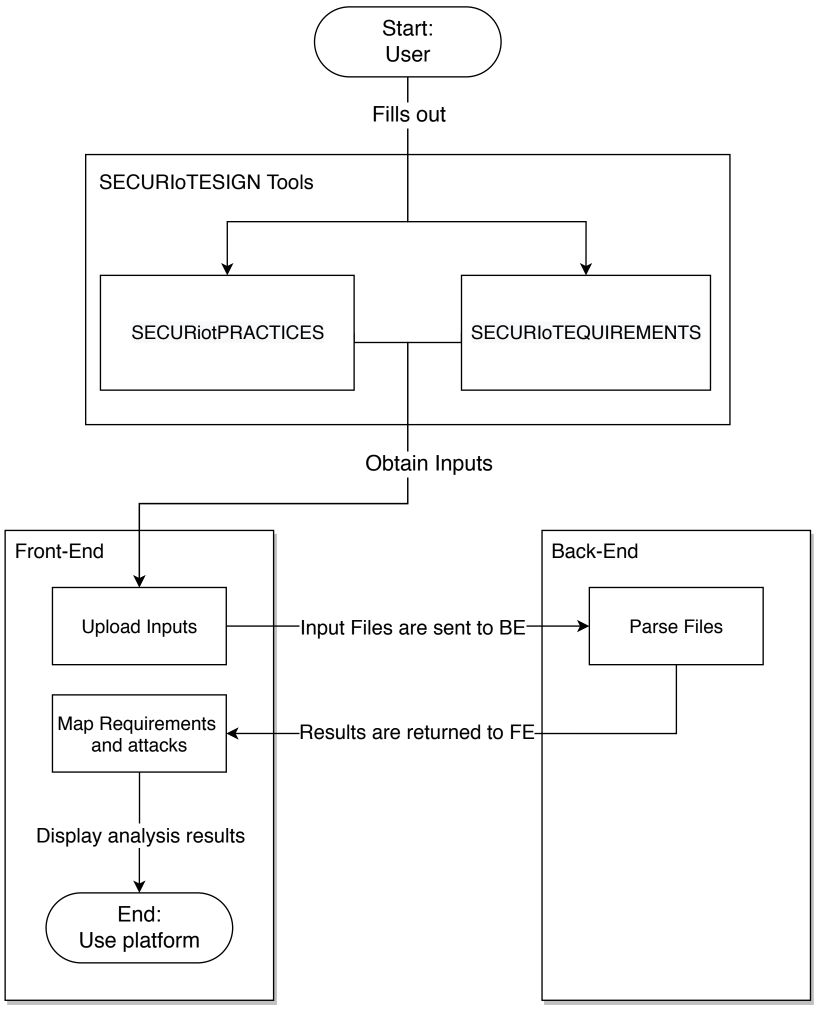


Figure 3.1: DFD representing the main flow of the system.

Another relevant analysis is the user flow analysis, which displays the interfaces/views and possible decisions along the usage flow that a typical user follows when interacting with the web application.

In this DFD (represented in figure 3.2) we can see that from a users perspective, he starts out in the index page, where he makes a decision to either select his input files from his device, or chooses to select one of the available example scenarios. Upon making a decision, there is an intermediate step where the data from the selection is processed and analyzed. Once concluded, the user is now capable of freely interacting and navigating between the web application sections, namely the system modeling, threat modeling and guides pages.

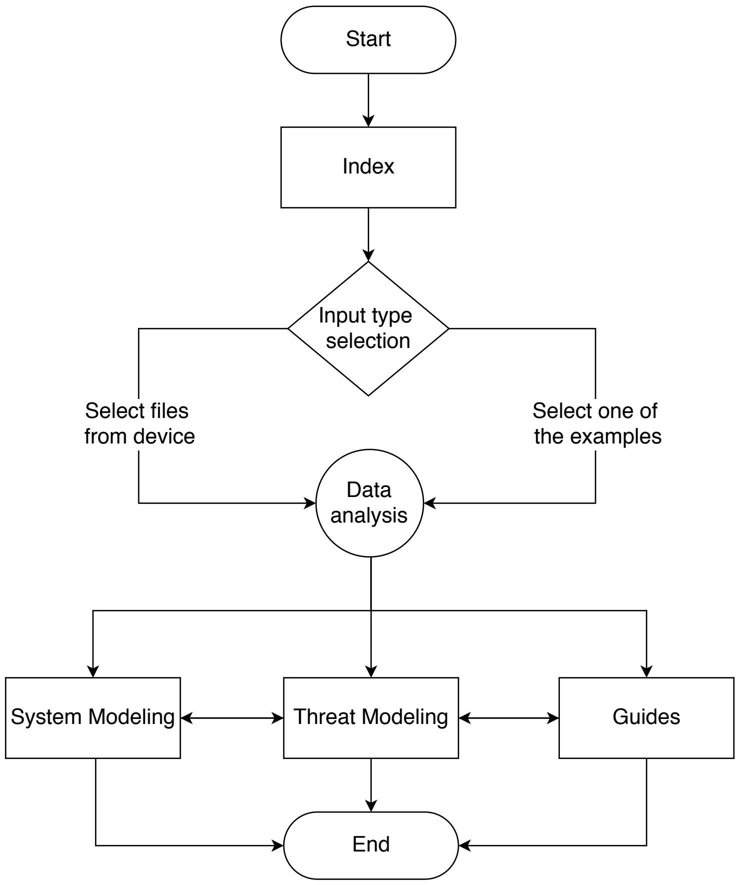


Figure 3.2: DFD representing the user flow of the system.

3.4 Conclusion

This chapter presents an overview of the requirements for the practical component of this project, as well as an analysis of the flows and the logic required for it to fulfill its purpose. This theoretical analysis and planning provides the reader with a better understanding of the project going into the next chapter, where its technical implementation will be analyzed in further detail.

Chapter 4

System Implementation

4.1 Introduction

This chapter will go into detail regarding the technical implementation of this project. Namely:

1. Section 4.2 – **Technologies and Libraries Used** – provides a review of the software and libraries used while also briefly explaining why they were chosen;

2. Section 4.3 – **Implementation Logic** – goes over the details of how each tool was implemented, describing the thought process behind said implementation;

3. Section 4.4 – **Technical Details and Code Samples** – this section contains code samples (and their explanations) of some of the most important parts of the system logic referenced over in the previous chapter;

4. Section 4.5 – **Technical Challenges** – provides an overview of some of the technical questions and difficulties that became obvious during the development of the platform, as well as providing a justification for the decisions made;

5. Section 5.4 – **Conclusion** – section that briefly summarizes the work detailed on this chapter.

4.2 Technologies and Libraries Used

This project can be seen split into two main components, namely front-end and back-end. The back-end component is where the Representational State Transfer (REST) APIs are implemented, they provide functionalities related with file processing and providing examples to the front-end. Meanwhile, the front-end component constitutes a web application, which is the part of the platform that graphically displays in the browser the results of the analysis of the data provided by the user questionnaires (the inputs of the system);

For the back-end development the following technologies were used:

- The main programming language used was **Java**. This was mostly due to previous experience with the language as well as support for other tools (Maven) and frameworks (Spring, Swagger) that would be very useful for the development process;

- **Maven** [Mav] is a very commonly used open source project management tool, it heavily facilitates and helps to organize processes such as dependency management, module management and project compilation/testing/deployment;

- **Spring** [Spr] is an application framework for Java. It contains a module called Spring Model View Controller (MVC) which is very powerful for the development of the backbone services and APIs for web applications. In addition to this, Spring contains a very large and well maintained ecosystem, that allows for the quick incorporation of various modules, such as ones dedicated to security or database integration for example. This great ecosystem and variety of available modules also has the advantage that it would heavily facilitate the addition of extra features, should future work be conducted on the platform;

- **Swagger** [Swa] is an open source framework that in essence, assists with the documentation of REST APIs. Once configured it only requires some annotations to be added to each API, as well as details such as a description or the expected inputs and outputs for it. The documented APIs can then be consulted in a web page that is auto generated by the framework. This is helpful in situations where other projects and people might have to use an exposed API, as well as internally, should more work be conducted in the platform in the future.

For the front-end development the following technologies were used:

- The main tool for the development of the web application was the open-source web development framework **Angular** [Ang]. It is a JavaScript web framework with a vast collection of supported libraries, allowing for an easy implementation of essential functionalities for any web application, such as data binding, routing and animations. This choice was mostly due to previous experience with the technology, being easy to create dynamic content, as well as having good support for the libraries used to represent the information and graphs;

- **D3-graphviz** [d3-20] was used to render, manipulate and handle all the aspects regarding the graphs. This library is a port of an open source graph visualization software called Graphviz. This library uses a programming language called DOT, which is a graph description language, to render its content;

- **Bootstrap** [Boo] implements a set of template elements and classes that allow for a more efficient view development process as well as facilitating the development of a responsive web app;

- **PrimeNG** [Pri] is a library that contains User Interface (UI) components and widgets. Its main usage was to assist with the information presentation and layout.

4.3 Implementation Overview

Regarding the technical implementation of the system, this process can be decomposed into three steps, which will be explained in the following sub-sections: Preparations, Back-end implementation and Front-end Implementation.

4.3.1 Preparations

The first part of this preparation was mostly theoretical and consisted of performing an analysis of the relevant literature and state of the art, in order to define the main methodology in which to display and create the modeling scenarios. The final choice was to represent the threat modeling component using attack trees, as this would be the most user-friendly way to convey the threat information to an audience with a possibly limited security knowledge or background. As for the system modeling component, the choice was to follow a traditional UML diagram because the system modeling considerations are applied to a very basic and generic architectural model. This is very simple model and with a high degree of abstraction due to the limitations regarding the amount of information that can be obtained from the under-laying projects this tool was built of - namely the outputs from the SECURIoTEQUIREMENTS and SECURIoTPRACTICES tools.

The second part consisted in defining the relevant information to be displayed in each tool. For the threat modeling tool all the relevant attacks (along with their countermeasures) to the IoT ecosystem were gathered and compiled into a JSON file. A JSON data structure was the selected format in which to internally save the data to be displayed to the user because it streamlined the process of parsing the contents of the file in order for the data to be processed and rendered. In addition to this, in case of necessity this format is already very portable so it trivializes the conversion into any other format.

Listing 4 presents a truncated example of the JSON representation of the attacks (and their countermeasures) that can be displayed in the threat modeling tool. Although this representation is lengthy, it is considered to be important to the visual interpretation of this explanation due to the fact that the JSON format is very easy to read and interpret. This JSON object is split into three child objects, each representing one of the main types of attack categories (Physical, Network and Application based attacks). Inside each category the attacks are then decomposed into sub-steps, meaning, they are decomposed into the several countermeasures that can be taken or implemented in order to prevent or mitigate each attack. A description of the attack and its countermeasures is also provided, as it will be used in the tool to provide some extra context to each attack.

Listing 4 Example of the definition of an threat in the security threats JSON file.

```
{
    ”category”: [
        {
            ”name”: ”Physical Security”,
            ”description”: ”These are the attacks that either physically damage or (...)”,
            ”threats”: [
                {
                    ”name”: ”Node Damage/Destruction”,
                    ”description”: ”An attacker physically damages or even destroys a device (...)”,
                    ”stepsDescription”: ” Devices should be built with strong, hard to access (...)”,
                    ”step”: [
                        {
                            ”name”: ”use tampering resistant packaging”,
                            ”step”: [
                                {
                                    ”name”: ”Consider indirect ways the device can be damaged (i.e. lasers,
                                    ↪ power surges)”,
                                    ”step”: null
                                }
                            ]
                        },
                        {
                            ”name”: ”prevent access to the device”,
                            ”step”: [
                                {
                                    ”name”: ”Secure the device location”,
                                    ”step”: [
                                        {
                                            ”name”: ”Add Surveillance”,
                                            ”step”: null
                                        },
                                        {
                                            ”name”: ”Place the device in a safe location”,
                                            ”step”: [
                                                {
                                                    ”name”: ”Hide the device in a hard-to-reach location”,
                                                    ”step”: null
                                                },
                                                {
                                                    ”name”: ”Camouflage the device”,
                                                    ”step”: null
                                                }
                                        (...)
```

For the system modeling tool another JSON file was produced, containing several pieces of information regarding each possible security requirement - namely a description, where the requirement can be applied, vulnerabilities that jeopardize it and lastly what are the best security practices for it. Additionally, both the threats and practices (also known as security guides) contain a description explaining in what way that threat or guide is relevant to that security requirement. An abbreviated example of this structure can be seen in listing 5.

Listing 5 Example of the definition of a security requirement in the requirements JSON file.

```
{
  "iot security requirements": [
    {
      "requirement": {
        "name": "Confidentiality",
        "description": "The property that ensures that information is not disclosed or (...)",
        "application": "This requirement should be applied where the information is (...)",

        "applicationShort": [
          "database",
          "transmission"
        ],
        "vulnerabilities": [
          {
            "name": "Node tampering",
            "description": "An attacker can physically tamper with a device, (...)"
          },
          (...)
        ],
        "practices": [
          {
            "name": "API Security Guide",
            "measures": "Sensitive information in HTTP requests",
            "description": "In order to preserve confidentiality special care (...)"
          },
          (...)
```

The information required to build this file was obtained from manually analyzing all the outputs from the SECURIoTEQUIREMENTS and SECURiotPRACTICES projects, associating both the correct practices and the relevant attacks for each security requirement. The mapping was done by inspecting each requirement produced by the SECURIoTEQUIREMENTS tool and then associating it the attacks that threaten it, where that requirement needs to be applied and what are the good practice guides that can assist to secure it.

Since this mapping process was highly dependent on individually analyzing all the possible output requirements, deducing the security context in which they are applied and then finally associating the correct threats and guides, this process was not practical nor viable to automate by using any inference mechanism, therefore it had to be done in a manual manner.

4.3.2 Back-End Implementation

The back-end implementation of the project was made in Java using the Spring framework. Spring allows for quick development, implementation and deployment of APIs and other services. In order to achieve the goals of the prototype the main functionalities for the back-end were the following:

- **To perform all the file processing on the server side** - There are several reasons for this, for starters, by doing these steps in the server this saves resources on the client-side (the browser) as web frameworks are typically less efficient processing wise due to the larger overhead and the environment where that code is executed. Another important factor is that by actually having a back-end implementation and exposing REST APIs, other applications can make use of the services developed. This allows for future tools or expansions of the SECUR IoT ESIGN project to easily make use of them should it be necessary.

Another advantage is that in future work the services and APIs developed to handle the file processing can easily be adapted in order to perform additional actions, such as saving the results of the parsing on a database, on a file server, sending them to another platform, among other possibilities;

- **To provide examples of use cases** – The platform already possesses several use case examples implemented as proofs of concept. These examples are created and made available through the back-end and then provided to the front-end through an API. This way these examples can be quickly and easily updated, removed or added, as well as being usable in other projects;

As a matter of organization and in order to maintain a high degree of modularity, the back-end was structured into the following packages:

- **Rest** – Contains all the classes that deploy a REST API. In this project all of them were created in the class called "ModelingRest". These REST classes do not implement any logic, since logic is executed exclusively in services, these classes call functions in the services and return their outputs;

- **Service** – Aggregates all the main logic components for the back-end, implementing all the logic that is provided in the APIs. The service classes created for this platform include a "FileStorageService" and a "FileParsingService";

- **DTO** – This package contains all the object classes created and used in the back-end, despite the name not all the objects declared here are actually transferred via APIs and are only used internally;

- **Utils** – This package is used to aggregate other packages and miscellaneous classes that are required. For example, it contains an "exceptions" package to aggregate the exceptions created, an "enums" package to aggregate possible *enum* classes and other classes responsible for properties, such as a "FileStorageProperties" and a "SwaggerConfig" class.

4.3.3 Frond-End Implementation

The front-end implementation was made using Angular. One of the many advantages of using this framework is that it implements elements of the web application view as modular components. A component controls its own section of the available screen and acts independently of other parent or child components that might exist in the overall view. Each component has an HTML file that contains the content to be displayed in that view, a Cascading Style Sheets (CSS) file responsible for the stylesheets in this component, a typescript file which is the controller for the view and finally a spec file which can be used to perform unit tests for that component.

In addition to components, Angular allows the creation of services which are singleton objects that implement some kind of logic. These can be reused across components as well as allowing for data to be shared between them, contributing to the development of a modular platform. Services are made up by a typescript file that contains all of the logic that is implemented, as well as a spec file used for unit testing.

For this platform, several modules were created, however the most important ones were the system and threat modeling related components. The overall component structure of the web application is as follows:

- **Index** - Also known as homepage, is the first component of the system. It contains a brief description of the project, instructions regarding how to upload the inquiry result files obtained from the other tools, as well as presenting an option to use one of the pre-built examples available. The user can upload their files by using a file upload widget or select one example from the tab window;

- **System modeling component** – The system modeling component is responsible for matching the system requirements from the inquiry result files with the system modeling mapping file, in order to display the relevant information to the user;

  The information regarding the requirements is organized into five categories representing the main components of a typical IoT system, namely:

  - **System** – Refers to requirements that are relevant across all components and entities of the system;

  - **Device** – This category encompasses all the devices/things of the system, meaning that it refers to all sensors, actuators, user devices and other nodes that interact with each other;

  - **Transmission** – Refers to all the wireless and wired transmissions of data in any part of the system;

  - **Database** – The database component refers to all the locations where data is stored;

  - **Cloud** – Refers to all the cloud based services utilized by the system.

  Each category contains a list of all the security requirements that are needed to properly secure that part of the system. By selecting one of these requirements information regarding them is displayed. This additional information consists of a brief description of what the requirement is, the relevant attacks that can breach this requirement and lastly a list of guidelines with information relevant to secure that requirement;

- **System modeling graph component** – This component is inside the system modeling component and it is responsible for creating the UML representation of the system with the labeled requirements on each of the main components mentioned previously. The decision to have a dedicated component for the graph representation was to keep the requirement mapping logic separate from the d3-Graphviz graph drawing logic, in order to keep the project more organized and modular;

- **Threat modeling component** – The threat modeling component is responsible for displaying the information regarding the attacks that can affect the security requirements of the application.

  In this component the attacks are organized into three security categories (physical security, network security and application security), which represent the system layer where they are performed. Inside each one of these categories are the relevant attacks to the system, as specified in the system modeling component.

  Each attack item can be expanded in order to display information regarding the attack, such as a description and an attack tree containing countermeasures that can prevent or counter that attack, these can be displayed both in graph shape and in a text based shape. Each attack tree is also followed by a more detailed text explanation of the countermeasures displayed, as a method to further contextualize the user.

  The attack trees with the steps to perform the attack itself are not represented because these steps can be vastly different depending on the type of IoT system, the type and amount of devices, the network topology, software used, among many other factors. Therefore, due to the generic nature of the inquiry and the lack of specifications regarding a single particular system, a choice was made to simply represent the steps that the developers of the system should consider in order to generally prevent these attacks from happening or being successful;

- **Threat modeling graph component** – This is the component responsible for generating the attack trees with the countermeasures. Like in the system modeling graph component, the decision to have this as a dedicated component was to separate the attack selection and information logic from the attack tree representation logic. The trees can be either text based or graphs based using d3-graphviz;

- **Main guides** – This is the parent component that contains and renders the guides as child components, it also acts a menu for all the guides;

- **The guides themselves** - Each guide has its own separate component and contains information regarding the secure implementation of several key functionalities in an IoT system. Guides were adapted from the original SECURiotPRACTICES project, having suffered only minor corrections. There are guides for the following topics:

  - Authentication;

  - API development and implementation;

  - Access Control;

  - Cross Site Scripting;

    - Cryptography;
    - File Upload;
    - IoT Security ;
    - Input Validation;
    - Logging;
    - SQL Injection;
    - Session Management;
    - Web Service development and implementation.

- **Navbar and footer components** - These simple components are mostly decorative. They are responsible for the header and navigation bar at the top of the page, as well as the footer at the bottom.

4.4 Technical Details and Code Samples

The purpose of this section is to complement the more theoretical analysis of the previous chapter by performing a more detailed analysis of the implementation, providing code (or pseudo-code) samples.

4.4.1 Back-end Details

The implementation steps for the back-end consisted of the following:

1. Creation of the DTOs package containing the object classes that were used during the project. There were two main DTOs, which can also be seen in listing 6:

    - PartialAnalysisDTO - is used to obtain the results of the parsing of a single file. It is made up by two attributes: a String which is used to store the type of file and a list of Strings that contains the name of the requirements or recommendations in that file;

    - FinalAnalysisDTO - this is the object that is returned to the front-end after files are uploaded and parsed as well as when obtaining the examples from the server. It contains four attributes: a String that acts as a name/id for the object, a description and two PartialAnalysisDTO, one to store the requirements and another for the recommendations;

Listing 6 Brief example of the variables in the DTOs used in the back-end of the project.

```
public class PartialAnalysisDTO {
    private String type;
    private List<String> items;

    (...)
}

public class FinalAnalysisDTO {
    private String id;
    private String description;

    private PartialAnalysisDTO requirements;
    private PartialAnalysisDTO recommendations;

    (...)
}
```

2. Creation of the REST APIs for the clients to have access to the services provided by the server:

    - uploadMultipleFiles receives the user files from the front-end in order for them to be processed and the result returned to the client. This API receives as a request parameter an array of MultipartFile objects which are passed to the FileParsingService processUploadedFiles function to be processed, returning the mapped result in an FinalAnalysisDTO to the front-end;

    - getExamples returns to the client a list of FinalAnalysisDTO that contain the information regarding the examples that are available to the users. This list is obtained by calling the getExamplesFromDir method in FileParsingService.

3. A file parsing service (FileParsingService) which contains three main methods:

    - parseFile either receives a file or loads one from a file name. Since the outputs from the other tools do not follow the same standard (but do obey some markdown rules) the first step is to identify the kind of file currently being studied. This is done by reading the first line of the file and checking if that string matches what a requirements or recommendations file starts with. This identification is saved on an object (a PartialAnalysisDTO) that will be returned in the end.

      Next, the file is read line by line, checking if the line contains the characters "##", which are the characters used to mark the start of a requirement or recommendation. This line is then stripped of the "#" characters as well as any white space and the resulting string added to a list on the PartialAnalysisDTO object, which is then returned. This simplified version of this function can be seen in listing 7;

Listing 7 Simplified version of the parseFile function which handles the parsing of an analysis file

```
public PartialAnalysisDTO parseFile(File file, String filename){
    PartialAnalysisDTO analysis = new PartialAnalysisDTO();
    List<String> securityRequirements = new ArrayList<>();

    try (BufferedReader br = new BufferedReader(file, filename)) {
        String firstLine = br.readLine();

        if (firstLine.contains("# SECURIoTEQUIREMENTS"))
         ↪   analysis.setType(FileTypeEnum.REQUIREMENTS);

        else if (firstLine.contains("# Final Report"))
         ↪   analysis.setType(FileTypeEnum.RECOMMENDATION);

        for (String line; (line = br.readLine()) != null; ) {
            if (line.contains("##"))
                securityRequirements.add(line);
        }
    }
    analysis.setItems(securityRequirements);
    return analysis;
}
```

- processUploadedFiles receives an array of Multipartfiles and passes them over to the parseFile method one at the time, receiving a PartialAnalysisDTO for each file. This object is then verified if it refers to a requirement or recommendation, in order to add them to a FinalAnalysisDTO that is returned. A simplified overview of this method is available in listing 8;

Listing 8 Simplified example of the implementation of the function processUploadedFiles, responsible for handling the files uploaded by the user.

```
public FinalAnalysisDTO processUploadedFiles(MultipartFile[] files){
    FinalAnalysisDTO proccessedOutput = new FinalAnalysisDTO();
    proccessedOutput.setId("analysisResults");

    Arrays.stream(files).forEach(multipartFile -> {
        try {
            PartialAnalysisDTO result = parseFile(null, multipartFile.getOriginalFilename());

            if (result.getType().equals(FileTypeEnum.REQUIREMENTS))
                proccessedOutput.setRequirements(result);

            if (result.getType().equals(FileTypeEnum.RECOMMENDATION))
                proccessedOutput.setRecommendations(result);
        } catch (Exception) {}
    });

    if (proccessedOutput.getRecommendations() == null || proccessedOutput.getRequirements()
     ↪   == null)
        throw new Exception("Invalid files");

    return proccessedOutput;
}
```

- getExamplesFromDir is the method responsible for handling the example analysis provided to the front-end. The examples are stored in individual directories, each one containing their respective requirements and recommendation files as well as a JSON file called "details.json" with two attributes, a name and a description for the example. This method scans the main examples directory for sub-directories inside of it. Then, for each file inside it calls the parseFile method in order to fill out a FinalAnalysisDTO with the name, description, requirements and recommendations belonging to that example. Finally, this object is added to a list that and returned to the getExamples API. A simplified example of this method can be found on listing 9;

Listing 9 Abbreviated and simplified code of the getExamplesFromDir function.

```
public List<FinalAnalysisDTO> getExamplesFromDir(){
    List<FinalAnalysisDTO> listAnalysisExamples = new ArrayList<>();

    baseDirectory.forEach(subdirectory -> {
        FinalAnalysisDTO outputExample = new FinalAnalysisDTO();
        Arrays.stream(subdirectory.listFiles()).forEach(file -> {
                try {
                    if (file.getName().equals("details.json")) {
                        outputExample.setDescription(file.get("description");
                        outputExample.setId(file.get("name"));
                    } else {
                        PartialAnalysisDTO parsedFile = parseFile(file);
                        if (parsedFile.getType().equals(FileTypeEnum.REQUIREMENTS))
                            outputExample.setRequirements(parsedFile);
                        else if (parsedFile.getType().equals(FileTypeEnum.RECOMMENDATION))
                            outputExample.setRecommendations(parsedFile);
                    }
                } catch (Exception) {}
        });
        listAnalysisExamples.add(outputExample);

    });
    return listAnalysisExamples;
}
```

4. During early development additional DTOs, APIs and services were made to handle individual file upload and download to and from the server. Despite these functionalities not being used in the final version they will remain in the project for future proofing, in case future development is made in the platform that makes use of those functionalities.

4.4.2 Front-end Details

The front-end was responsible by taking the requirements and practices contained in the FinalAnalysisDTO and displaying the data associated with it to the user in each dedicated component. In order to better understand the functioning of these components it is necessary to keep the following in mind:

1. The files containing the security requirements (plus associated information) and the attack countermeasure information were stored in the "assets" directory under the names of "system_requirements.json" and "attacks_and_countermeasures.json" respectively. Re-

peated re-transmission of these files are not a concern, since browsers should automatically cache these files. Thus, when using the application and as long as the files are still cached and up to date, an Hypertext Transfer Protocol (HTTP) 304 Not Modified response code is received when requesting the files;

2. Despite services allowing for sharing data across components, they do not allow for data to persist across multiple sessions. Therefore, in order to persist information even if the application is closed and later re-opened or refreshed, the web browsers local storage was used. This also allows to set the initial state of the application if it has already been used previously and the local storage has not been wiped. The following objects were saved this way:

   - **"userInputAnalysis"** – if a user uploads analysis files then the resulting FinalAnalysisDTO is stored here;

   - **"listExamples"** – contains all the FinalAnalysisDTO objects obtained from the server when fetching the list of examples;

   - **"selectedAnalysis"** – stores the JSON object of the FinalAnalysisDTO from the analysis selected by the user, whether it is from an example or from a file upload;

   - **"relevantAttacks"** – is used to store the relevant attacks to the requirements defined in the "selectedAnalysis" object.

3. Two Angular services were created, one to handle fetching the examples by calling the getExamples API and another one to handle saving the relevant attacks from the analysis requirements:

   - **GetExamplesService** – simply calls the getExamples API and returns the output;

   - **RelevantAttacksService** – is the service responsible for mapping the relevant attacks to the "selectedAnalysis" object and saving them in the local storage (under the "relevantAttacks" entry). This service works by loading the "system_requirements.json" file in the extractAttacksFromRequirements function, which then passes it to the getAttacks method. Here, the "selectedAnalysis" object containing the name of all the security requirements for system is also loaded from local storage. In order to obtain the data relevant to the attacks, these objects are iterated one over the other and if the names of the requirements match (meaning that requirement is applicable to the system being analyzed), the current requirement of the "system_requirements.json" file is passed onto the getAttacks function to extract the attacks/vulnerabilities into a list. Finally, when this process is complete this list containing the relevant attacks (and attack information) for the selected system is saved to the local storage under the key name of "relevantAttacks". A simplified and abridged version of the code for this service is demonstrated in listing 10. In order to fully explain the logic it implements it wasn't possible to shorten it even more.

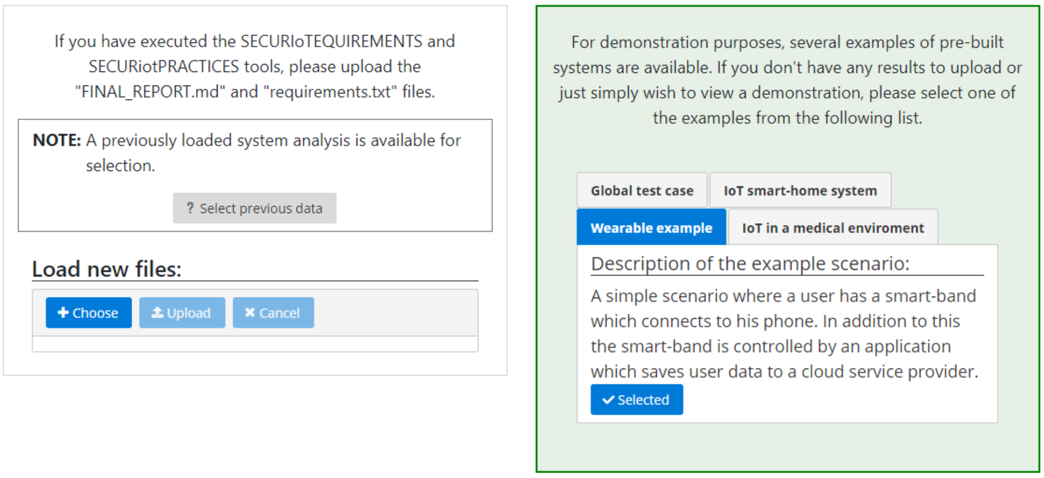
Listing 10 Simplified code of the Angular service RelevantAttacksService, which is responsible for extracting the relevant attacks from a group of requirements.

```
relevantAttacks: Object[] = [];

extractAttacksFromRequirements() {
    var requirementsFile = "./assets/system_requirements.json";
    this.getAttacks(requirementsFile));
}

getAttacks(requirementsFile) {
    let selectedRequirements = localStorage.getItem("selectedAnalysis")["requirements"]["items"];
    let allSecurityRequirements = requirementsFile["iot security requirements"];

    allSecurityRequirements.forEach(requirement => {
        selectedRequirements.forEach(selectedRequirement => {
            if ((requirement["requirement"]["name"]) == selectedRequirement)
                this.setRelevantAttacks(requirement["requirement"]["vulnerabilities"]);
        })
    })
}

setRelevantAttacks(receivedAttacks) {
    let exists = false;
    receivedAttacks.forEach(attack => {
        this.relevantAttacks.forEach(element => {
            if (attack["name"] == element["name"]) exists = true;
        });
        if (!exists) this.relevantAttacks.push(attack);
    });
    localStorage.setItem("relevantAttacks", this.relevantAttacks);
}

getRelevantAttacks() {
    return localStorage.getItem("relevantAttacks");
}
```

Having explained these aspects, the more important front-end components and their implementation can be explained:

- **Index** - The index component is the homepage for the platform and its main purpose is to allow the user to either upload their requirements and practices files or to select one from the available examples. In order to display the examples, this component made use of the available Angular life cycle hooks by calling the ngOnInit function, which is executed when Angular is finished with creating the component. Inside this function the list of examples is initialized by first calling the GetExamplesService to obtain the array of objects containing the examples and parsing them into another object array suitable to be loaded by the tab view widget, which dynamically creates a tab for each element in this array.

  To perform the file upload a widget from the PrimeNG library was used. The user selects the files and clicks upload, sending the files to the uploadMultipleFiles API. In addition to this, an onUpload function called handleFileProcessingResponse is executed, receiving the reply from the API call and storing the FinalAnalysisDTO in the local storage under the key name of “userInputAnalysis”, as well as automatically selecting it by setting the “selectedAnalysis” entry in the local storage with this data. A screenshot of this component can be seen in figure 4.1 and an abridged version of these functions can be seen on listing 11, which could not be made shorter or simpler in order to display the relevant logic for this component.


If you have executed the SECURIoTEQUIREMENTS and SECURiotPRACTICES tools, please upload the "FINAL_REPORT.md" and "requirements.txt" files.

NOTE: A previously loaded system analysis is available for selection.

Load new files:

For demonstration purposes, several examples of pre-built systems are available. If you don't have any results to upload or just simply wish to view a demonstration, please select one of the examples from the following list.

Description of the example scenario:

A simple scenario where a user has a smart-band which connects to his phone. In addition to this the smart-band is controlled by an application which saves user data to a cloud service provider.

Figure 4.1: Screenshot of the index component were the user can upload his own files or select one of the available examples.

Listing 11 Simplified version of the main functions in the index component controller.

```
ngOnInit() {
    this.listExamples = this.fillList();
}

fillList() {
    let tempList = [];

    this.examplesService.getExamples().subscribe((examples: Object[]) => {
        localStorage.setItem(”listExamples”, examples);

        examples.forEach(element => {
            if (element[”id”])
                tempList.push({ label: element[”id”], value: element[”description”] });
        })
    });
    return tempList;
}

handleFileProcessingResponse(uploadEvent) {
    let analysisTo = uploadEvent[”originalEvent”][”body”]
    if (analysisTo[”requirements”] != null && analysisTo[”recommendations”] != null) {
        localStorage.setItem(”selectedAnalysis”, analysisTo);
        localStorage.setItem(”userInputAnalysis”, analysisTo);
        this.messageService.add(summary: 'File content successfully loaded'});
    }
    else this.messageService.add( summary: 'Failed to process files'});
}
```

Finally, to assist with user usability and visual clarity, some auxiliary functions were also created in order to keep track if the currently selected analysis belongs to a user upload or to an example (and what example is selected) as well as to indicate the selected analysis when re-opening the web application;

- **system-modeling** – In the system modeling component the relevant security requirements are displayed, as well as a simple UML diagram representative of where said requirements need to be applied. The requirements are displayed on the left side of the page and split into five categories, organized in tabs within a tab view element. Like it was mentioned in the previous section, each category corresponds to a major architectural component in a typical IoT system.

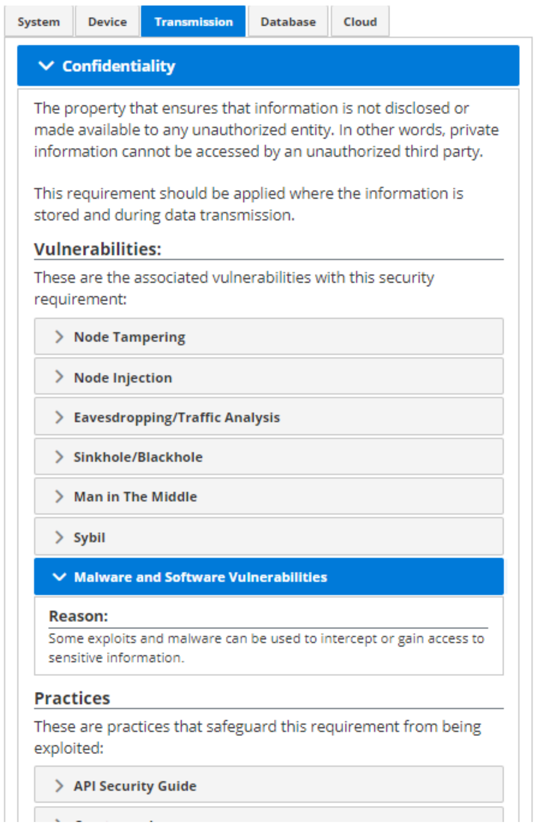
  Inside each one of these tabs the attacks are displayed in a type of graphical control element called an accordion, which expands in order to display the content within. This content consists of an explanation of why that requirement is relevant, the attacks that can breach it and finally what are the relevant guides to secure it.

  The static content in this page consists of the five base categories and all other text assets, which are mostly loaded from the security requirements file. The content displayed dynamically consists of the relevant requirements to each part of the system in the tab view element and in the graph. In order to render this content dynamically in the correct tabs the ngAfterViewInit life-cycle hook is called after the view is initially rendered, executing a function that opens the security requirements file and iterates the analysis selected by the user over it. If a necessary requirement is found, the locations in the system where it needs to be used are found and that requirement is added into an array of requirements for that part of the system. This array is used by the view in order to display all the relevant attacks in each category. An example of this controller implementation logic is displayed in listing 12.

Listing 12 Abridged code of the system modeling component controller that selects the necessary requirements.

```
applicationTypes: String[] = ["database", "cloud", "device", "system", "transmission"];
databaseApplicationInfo: object[] = new Array();
cloudApplicationInfo: object[] = new Array();
(...)
applicationInfoComplete: object[] = new Array();

ngAfterViewInit() {
    var filename = "./assets/system_requirements.json"
    this.parseJson(filename);
}

parseJson(json) {
    let storedRequirements = JSON.parse(localStorage.getItem("selectedAnalysis"));

    json.forEach(element => {
        storedRequirements.forEach(item => {
        if ((element["name"]) == item["name"]) {
            this.applicationTypes.forEach(type => {
                if (element["requirement"]["application"].includes(type)) {
                    switch (type) {
                        case "database":
                            this.databaseApplicationInfo.push(element["requirement"])
                            break;
                        case "cloud":
                            this.cloudApplicationInfo.push(element["requirement"]);
                            break;
                        (...)
                        default:
                            break;
                    }
            (...)
    });

    this.applicationInfoComplete.push(
        this.databaseApplicationInfo,
        this.cloudApplicationInfo,
        (...)
    );
}
```

For example, if a requirement needs to be applied in the database and in the cloud service of the system, the iteration of the “system_requirements.json” file goes over that requirement, matching it to the same requirement contained in the “selectedAnalysis” object. Then, the system category where it is applied is checked by verifying the “application” property of the current JSON requirement, and adding it into a database requirements array and a cloud requirements array. In the view of the component these arrays are used to display the requirements inside the tabs by using the Angular ngFor directive to render an accordion for each requirement and in the correct category (database and cloud in this example). Images of this component can be seen of figures 4.2 and 4.3 and in Appendix A there is listing 16, which demonstrates the logic on the view to render this component using the Angular ngFor directives, as was explained in the previous paragraph;


System | Device | Transmission | Database | Cloud

Confidentiality

The property that ensures that information is not disclosed or made available to any unauthorized entity. In other words, private information cannot be accessed by an unauthorized third party.

This requirement should be applied where the information is stored and during data transmission.

Vulnerabilities:

These are the associated vulnerabilities with this security requirement:

- Node Tampering
- Node Injection
- Eavesdropping/Traffic Analysis
- Sinkhole/Blackhole
- Man in The Middle
- Sybil
- Malware and Software Vulnerabilities

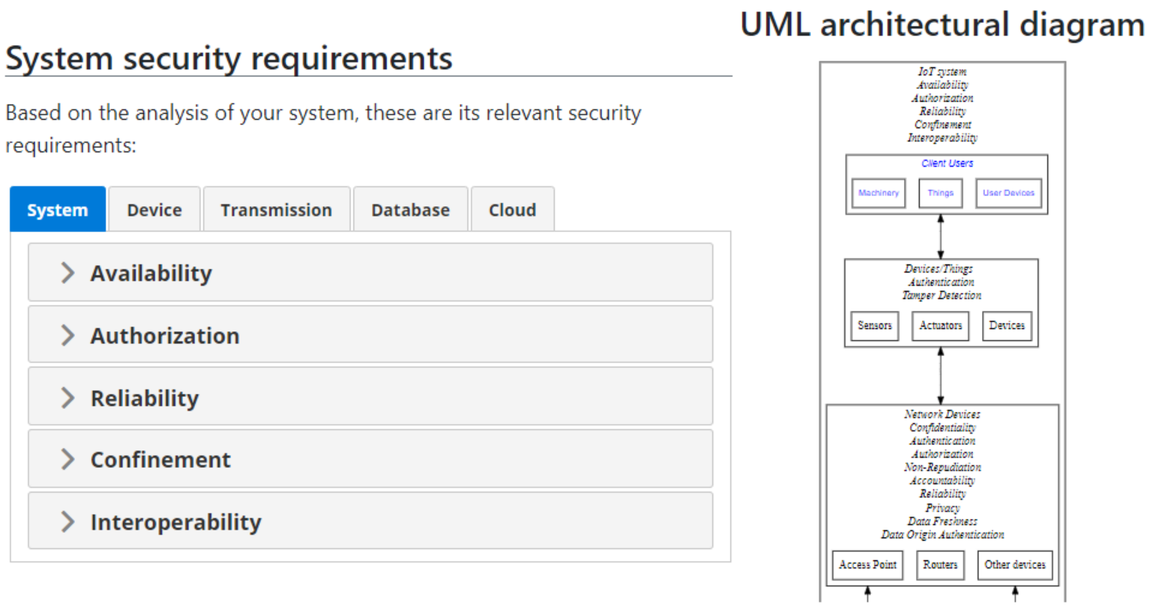
Reason:
Some exploits and malware can be used to intercept or gain access to sensitive information.

Practices

These are practices that safeguard this requirement from being exploited:

- API Security Guide

Figure 4.2: Example of the information and content inside a requirement accordion.


System security requirements

Based on the analysis of your system, these are its relevant security requirements:

System | Device | Transmission | Database | Cloud

- Availability
- Authorization
- Reliability
- Confinement
- Interoperability

Figure 4.3: On the left: the tab-view with the five categories and requirements in the accordions. On the right: the (cropped) system diagram with the requirements listed for each category.

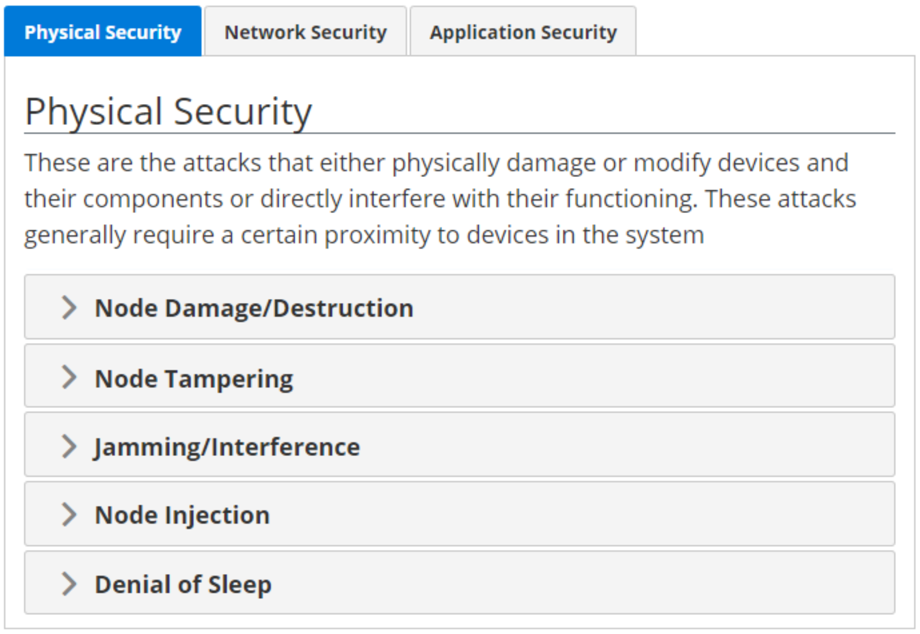
- **system-modeling-graph** – Within the system-modeling component, on the right side, a system diagram is rendered using the d3-graphviz library. This diagram is mostly a visual indicator of where and what requirements should be applied to each major subsystem in order to fulfill the correct requirements. There was an attempt to make this graph conform to a typical UML communications diagram but due to limitations of the library this was not fully possible.

This component is rendered into view when the system-modeling component finishes processing/selecting the correct requirements to display. When this process is complete it renders this component, passing into it an object containing the requirements for each category. Here in the ngOnInit life-cycle hook, the requirements are extracted and placed in a template readable by d3-graphivz to render the model. Graphviz renders the model by taking a string containing the code (written in the DOT language) and binding to a Document Object Model (DOM) element, rendering the graph inside. A minimalist version of this logic can be seen in listing 13 and a screenshot of part of the component and the diagram can be seen in the previously shown figure 4.3;

Listing 13 Simple example of how the system model is rendered with the correct requirements using d3-graphviz.

```
@Input() applicationObject: object;

ngOnInit(): void {
    let cloudRequirements: String = this.applicationObject["cloud"];
    let databaseRequirements: String = this.applicationObject["database"];
    (...)

    let dotGraph = `
        strict digraph {
            subgraph cluster_database {
                label="Storage requirements:  ${ databaseRequirements}"
                database [  shape=rectangle label="Database"]
            }
            subgraph cluster_cloud {
                label="Cloud Services: ${cloudRequirements}"
                cloud [ shape=rectangle label="Cloud"]
            }
        }
        (...)
        acessPoint -> cloud
        cloud->database
        database->cloud
        (...)
    }`;

    graphviz('#diagramDiv').fit().scale(1).zoom(true).renderDot(dotGraph);
  }
```

- **threat-modeling** - The threat modeling component displays all the relevant attacks to the system, split into three main categories depending on what major system layer (physical, network or application) is affected by the attack. The categories are organized into tabs and inside each tab the attacks are displayed using an accordion view, as can be seen in figure 4.4).

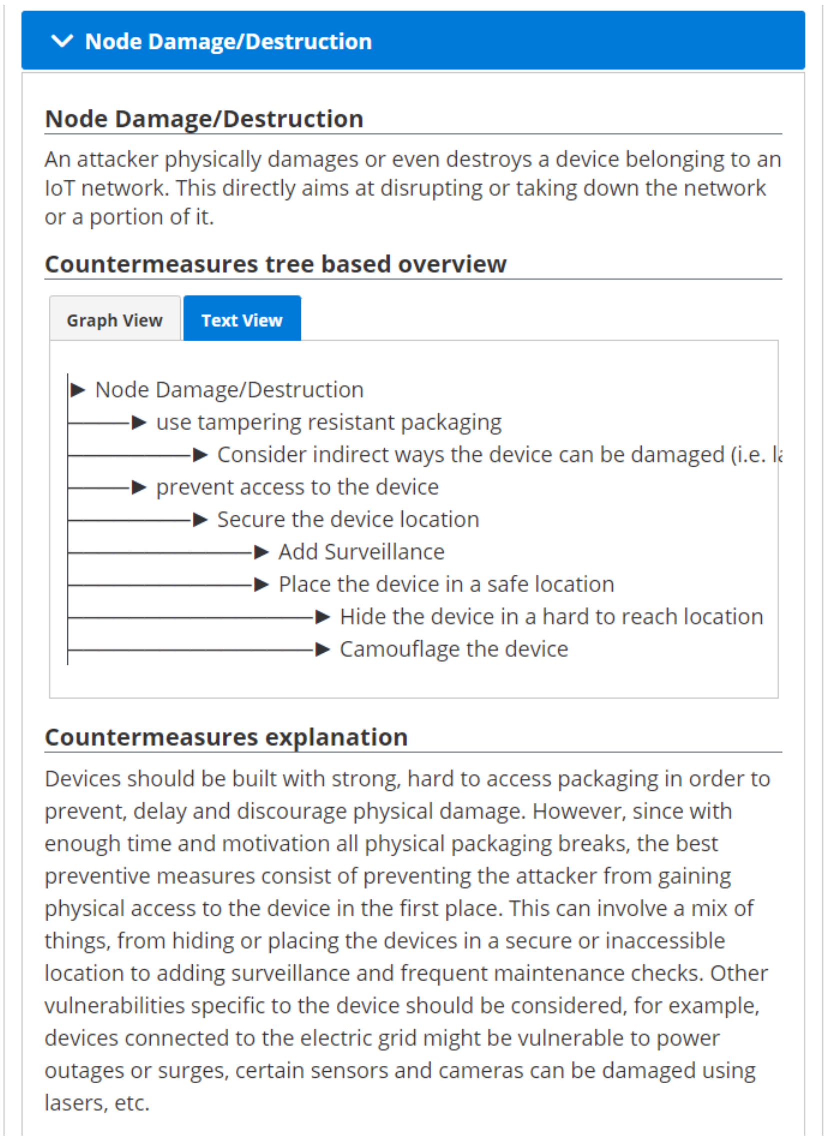


Physical Security

These are the attacks that either physically damage or modify devices and their components or directly interfere with their functioning. These attacks generally require a certain proximity to devices in the system

Figure 4.4: Screenshot of the tab view containing the types of layers in which attacks are performed, complete with all the relevant attacks in their own accordion.

Inside each accordion there is an attack description as well as an attack tree with countermeasures and an explanation of those measures. This tree essentially displays the defensive steps to prevent and defend against that particular kind of attack, with the smaller branches inside representing sub-steps of that countermeasure. The tree display can be toggled between a text based three, or a graph based three using D3-Graphviz. The contents of an accordion, complete with the attack tree can be seen in figure 4.5.



Node Damage/Destruction

Node Damage/Destruction

An attacker physically damages or even destroys a device belonging to an IoT network. This directly aims at disrupting or taking down the network or a portion of it.

Countermeasures tree based overview

```
▶ Node Damage/Destruction
  ──▶ use tampering resistant packaging
      ──▶ Consider indirect ways the device can be damaged (i.e. l
  ──▶ prevent access to the device
      ──▶ Secure the device location
          ──▶ Add Surveillance
          ──▶ Place the device in a safe location
              ──▶ Hide the device in a hard to reach location
              ──▶ Camouflage the device
```

Countermeasures explanation

Devices should be built with strong, hard to access packaging in order to prevent, delay and discourage physical damage. However, since with enough time and motivation all physical packaging breaks, the best preventive measures consist of preventing the attacker from gaining physical access to the device in the first place. This can involve a mix of things, from hiding or placing the devices in a secure or inaccessible location to adding surveillance and frequent maintenance checks. Other vulnerabilities specific to the device should be considered, for example, devices connected to the electric grid might be vulnerable to power outages or surges, certain sensors and cameras can be damaged using lasers, etc.

Figure 4.5: Example of the contents inside of the accordion view of an attack. The threat-modeling-graph component can be seen in the center.

The accordions with the attack information are all rendered dynamically according to what security requirements (and therefore relevant attacks) are selected. Since the list of relevant attacks is already stored in the local storage, what the component controller does to render the attacks consists in calling a function inside the ngAfterViewInit life-cycle hook, loading the ”attacks_and_countermeasures.json” file from the assets and the relevant system attacks from the local storage “relevantAttacks” entry. Then, these are iterated one over the other and if an attack from the requirements is matched with that attack in the file, that attack and all its information is added to an array. The abridged example of this controller can be seen in listing 14.

Listing 14 Simplified example of the code responsible for selecting the information relevant for the attack modeling component.

```
jsonAttacks: Object[] = null;

ngAfterViewInit() {
    var filename = "./assets/attacks_and_countermeasures.json";
    json_proccessor(filename);
}

json_proccessor(json) {
    this.jsonAttacks = json["category"];

    let usedAttacks = JSON.parse(attackTrackingService.getRelevantAttacks());
    let finalPhysicalSection = new Array();
    let finalNetworkSection = new Array();
    let finalApplicationSection = new Array();

    this.jsonAttacks.forEach(element => {
        if (element["name"] == "Physical Security") {
            element["threats"].forEach(attack => {
                usedAttacks.forEach(element => {
                    if (element["name"] == attack["name"])
                        finalPhysicalSection.push(attack);
                });
            });
            this.jsonAttacks[0]["threats"] = finalPhysicalSection;
        });

        if (element["name"] == "Network Security") {
            (...)
        }
        (...)
    });
}
```

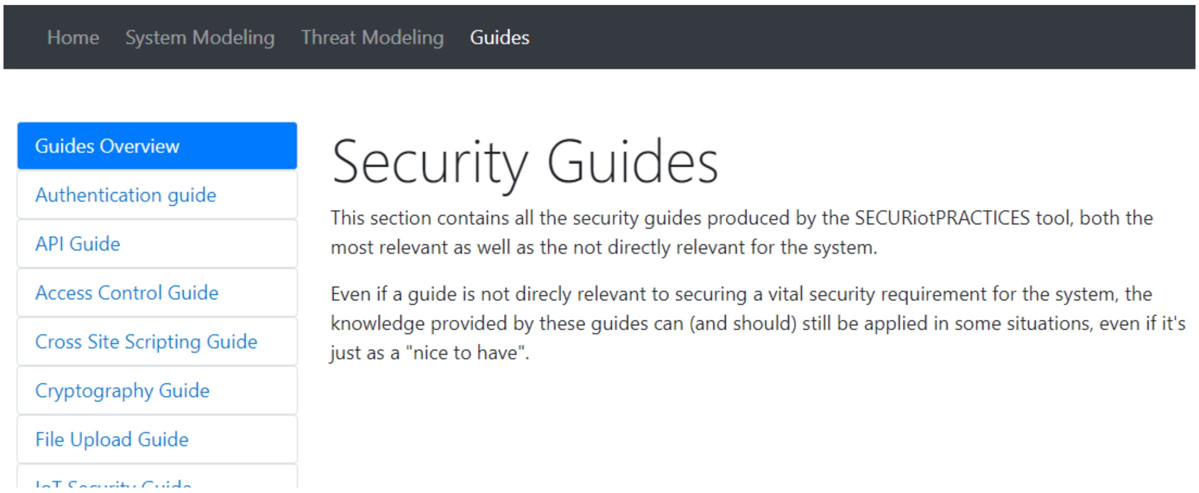
This array is used to display the accordions with the attacks, as well as the individual attack information inside. Also, inside each attack accordion, the corresponding threat-modeling-graph component is rendered, passing the current attack object into it as a parameter;

- **threat-modeling-graph** – For the graphic representation of the attack countermeasures two options are provided: a text based or a graph based tree. In order to display these correctly, a recursive function was used to go through all the branches of the attack countermeasures contained in the attack object received as a parameter. To render the graphs this recursive function saves all parent to child branch relations as strings, and passes this list of strings to d3-Graphviz, which automatically renders all the nodes and edges between them on the DOM element selected. For the text representation, a string containing the name of the node was padded based on the level of recursion and pushed into an array. Then, in the view of the component a ngFor cycle was used to render the array of strings, displaying the tree with the countermeasures. The (simplified) code behind these functions can be seen in listing 15;

Listing 15 Simplified example of the code responsible for processing the attack countermeasures in order for them to be displayed in an attack tree representation.

```
@Input() attackType: Object;
listEdges: String = "";
attackDotCode: String = "";
textAttackTree: Object[] = [];

ngOnInit() {
    this.attackDotCode = `digraph  {`  + this.initializeData()+ `}`;
  }

ngAfterViewInit() {
    graphviz('#' + this.attackName).renderDot(this.attackDotCode);
}

initializeData() {
    this.scanAttack(this.attackType, 0);
    return this.listEdges;
}

scanAttack(json, index) {
    var key;
    if (json instanceof Object) {
        for (key in json) {
            if (key == "step") {
                try {
                    json[key].forEach(element => {
                        let edge_string = json["name"] + " -> " + element["name"];
                        this.listEdges = this.listEdges.concat(edge_string);
                    });
                } catch (error) {}
            }
            if (json.hasOwnProperty(key))
                this.scanAttack(json[key], index+1);
        }
    } else {
        if (typeof(json) == "string" )
            this.textAttackTree.push({value: "--".repeat(index) + ">" + json });
    };
    return 0;
}
```

- **guides** - The guides component is used to display the information of all the relevant guides to the system. The information in the guides is the same that is present in the SECURIoT-PRACTICES good practice guides, they were simply organized into a web page for easier user accessibility. Guides were displayed in a vertical tab view using a PrimeNG widget where each tab loads an individual component containing a particular guide. These guide pages were statically created, so adding or modifying an existing one will require modifications to the HTML code itself. A screenshot of the guides component can be seen in figure 4.6



Security Guides

This section contains all the security guides produced by the SECURiotPRACTICES tool, both the most relevant as well as the not directly relevant for the system.

Even if a guide is not direcly relevant to securing a vital security requirement for the system, the knowledge provided by these guides can (and should) still be applied in some situations, even if it's just as a "nice to have".

Figure 4.6: Screenshot of the guides component, to the left is the tab view were the guides are selected, while the main content is displayed to the right.

4.5 Technical Challenges

Over the course of the implementation of system several design and implementation challenges became evident and a solution had to be thought up and implemented. These challenges consisted in:

- **How to perform the mapping between the SECURIoTEQUIREMENTS and SECURiotPRACTICES tools** – The first problem faced during the development of this project was deciding on how to go about combining the outputs of the requirements and practices tools. The output from the SECURIoTEQUIREMENTS tool contained three pieces of information: the requirements and their descriptions, where in the system they are applied, and the attacks that breach that requirement. The SECURiotPRACTICES tool produced a file containing the guides necessary to secure the system, given its architecture.

  The decision was to combine the two in a single file, mapping all the possible requirements (including relevant attacks) to the relevant practices. During this process a review of the existing requirements data was also performed, and information was added explaining why each attack or guide is necessary, or the dangers of not properly following the correct practices.

  From a technical standpoint it was then necessary to decide on how to save all this data. The final decision was to organize and store the contents in a JSON file, as this format has several benefits, such as:

  - It can be self explanatory and easy to read/interpret, meaning that someone else could easily maintain and update it in the future;

  - It is compact, even more so compared to other standardized formats such as XML, so it does not create unnecessarily large traffic during loading;

- It is very easy to parse in any modern programming language, so manipulation in the front/back-end is trivial.

This file is kept inside the assets folder of the Angular project. When the application loads for the first time it is downloaded from the server and cached, being automatically re-downloaded if it gets updated;

- **In what way should the analysis results be displayed to the user whilst having the mapped data in consideration** – Presenting the analysis results in a simple manner was also an important topic, as usability by even developers with limited security knowledge was also an important requirement. Therefore, and from the analysis of the various modeling methodologies (as reviewed over in chapter 2.4.1) the conclusion was that the simpler the model the better, so a traditional UML diagram for the system model and an attack tree for the threat modeling component would be the best solutions. Also, it is a good practice to label or provide relevant text to the images, graphs, or any other visual elements presented to the user;

- **Since this project was to be integrated with the rest of the SECURIoTESIGN platform, how should it be developed in a manner that facilitates this integration, as well as any possible future work and development** – In order to fulfill this requirement the choice was to separate the platform, developing a back-end and a front-end service instead of performing all the processing and file analysis on the web application (client-side). Having a dedicated back-end has the following advantages:

    - Allows for more operations to be done to the data, for example, if a persistence service were to be added in the future using a database, it would be simple to implement the required modifications in the services where the files are processed, and without making any alterations to the API input/output, making the process transparent to the browser or any other service that uses that API;

    - A dedicated back-end allows for the implementation of public APIs, meaning that other applications, services and projects could potentially make use of the work developed in the context of this project. In addition to this, adequate documentation of the internal services and available APIs would also be an advantage should other future projects require functionalities developed by this project;

    - Saves computational resources on the client, meaning that the user should not feel any stuttering or spikes due to processing, overall contributing to a more responsive platform, especially in mobile devices.

Additionally, both the back-end and front-end implementations were made having modularity in mind. In the case of the back-end, each package and their classes are responsible for a single task, allowing for the logic inside them to be expanded without altering the rest of the back-end implementation.

In the front-end, by taking advantage of Angular and implementing different components and services for each major functionality, it is possible to rework, add, remove or even move a component without it affecting the logic of the rest of the application or the view where it is inserted. For example, if a new major component were to be added in the future, this could be done without any sort of impact to the remaining structure, and all the existing ones could even be re-used and added inside it.

Lastly, by using a JSON file to store the system requirement and attack mappings, as long as its structure is not modified it is possible to easily update them and add or remove data without having to change anything in the front-end. This greatly benefits the projects ability to be updated and expanded in the future.

4.6 Conclusion

This chapter provided detailed information regarding the technical implementation of the platform, including an overview of the technologies utilized, details regarding the logic and implementation of the platform and some of the technical issues that were encountered and the approach to solve them. The evaluation of this implementation will be presented and discussed in the following chapters.

Chapter 5

Testing and System Validation

5.1 Introduction

This chapter will present an analysis of the testing methodology and execution, as well as the results obtained and conclusions deduced from them.

The chapter is organized into the following sections:

1. Section 5.2 – **Methodology** – explains the testing methodology used in order to evaluate if the platform fulfills the necessary requirements;

2. Section 5.3 – **Results and Conclusions** – analyses the results and draws their conclusions;

3. Section 5.4 – **Conclusion** – section that briefly summarizes the work conducted over this chapter.

5.2 Methodology

In order to fully test the system and evaluate if all the requirements were met, the testing methodology utilized consisted in the following steps:

1. Design several hypothetical IoT systems representative of real world scenarios and run them through the SECURIoTEQUIREMENTS and SECURiotPRACTICES tools, obtaining their requirements and practices files;

2. Run the files of those scenarios through the platform and analyze if there are no problems with the platform, check if all the information and diagrams are present and correct, if there are no display issues or any other unexpected problems. Fix any issues found;

3. Select one of those scenarios and use it to verify if the analysis produced by the system is correct when compared to one performed by an expert in security in IoT. This was performed in the following steps:

   (a) Develop a document for the scenario with information regarding the devices in the system, how they are connected and what relationship is there between them, the system users, and any accounts or applications used;

   (b) Provide that document to an expert and ask him to manually develop a system requirement and threat analysis;

(c) Provide that same information to a group of volunteers and have them fill out the SECURIoTEQUIREMENTS and SECURiotPRACTICES tools and then upload to the prototype the requirements and practices files obtained;

4. Compare the analysis made by the expert with the analysis obtained by the users, evaluating how the analysis produced by the prototype compares to the one made by the security expert.

Regarding the (hypothetical) test scenario declaration mentioned in step 1, three small scenarios were defined to test if the prototype was functioning as expected and there were no major issues. These scenarios were also deployed in the examples section in the index component.

A fourth, more detailed scenario was then developed for the system validation tests. This scenario was created in order to provide all the involved volunteers enough information to make valid and reasonable assumptions regarding the system. The information provided to both the test users and the expert was the same and consisted of the following:

- **Devices** – 5 smart lamps, 1 Amazon Echo (the Alexa home assistant), 3 smart plugs, 3 security cameras, 1 smart AC unit, 1 soundbar, 1 smart cooker, 1 robot vacuum, 2 routers, 1 Network-Attached Storage (NAS) unit, 1 desktop computer and 2 phones;

- **Users** – 2 Home users. Neighbors and other passers-by can come to the vicinity of the house;

- **Context** – These devices are part of a smart home environment within one Wi-Fi network. All the devices except the cameras are located inside the house. The smart-lamps, smart plugs, AC unit and cooker work of Wi-Fi and can be all be controlled and programmed by Alexa, using a total of 3 phone applications or via three web portals. Most of these applications require an authenticated account to work. The robot vacuum can be controlled by a phone application via Wi-Fi. The soundbar can be connected via Bluetooth and controlled by a phone or Alexa. The security cameras record footage to a NAS server. 2 cameras are connected via Wi-Fi and 1 via an Ethernet cable. The users can access the NAS and its contents by using a computer or any mobile device via the internet. A password is required for this. Alexa and the NAS server have access to a cloud service account for backups. This requires an authenticated user account.

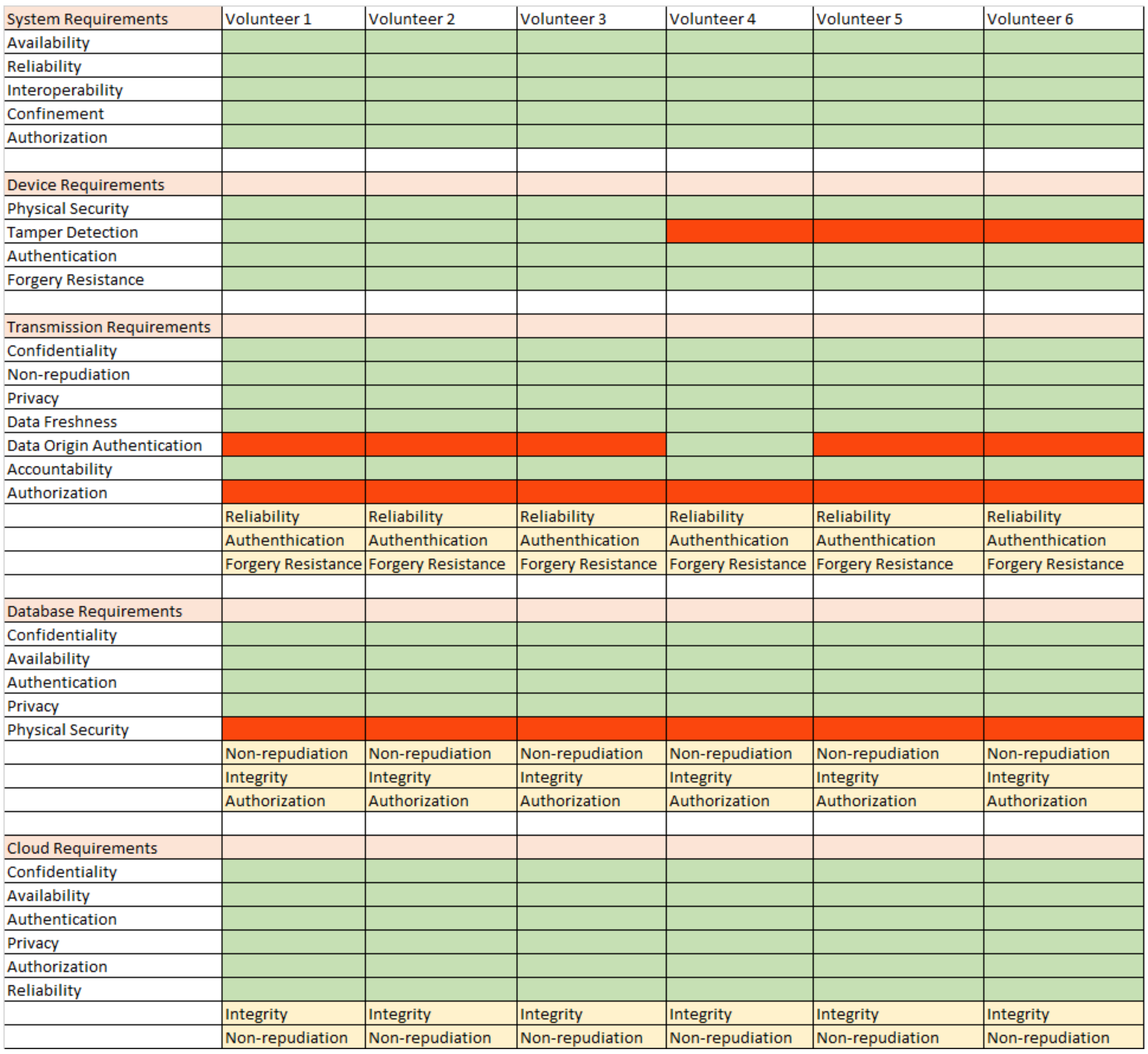
5.3 Results and Conclusions

As it could be seen from the analysis of the proposed methodology there are two testing steps that are important to analyze.

The first was step 2, which allowed to identify a number of bugs with the display of certain visual elements of the platform, as well as some correct requirements and threats not being displayed at all. Additionally, some changes and corrections were also performed to the requirements mapping file, namely updating the location where some requirements were applied in the system.

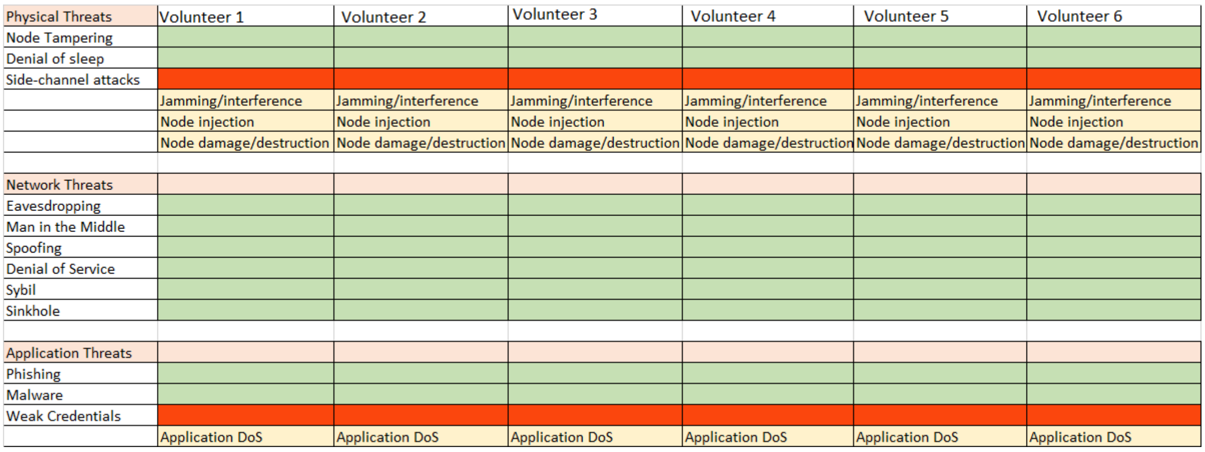
However, the most important testing step was step 4. A number of six volunteers were used to perform this testing stage and the results from these tests in comparison to the results obtained by the expert can be seen in tables 5.1 and 5.2.

The first figure represents a mapping of what were the requirements defined by the expert, and what were the results obtained by the volunteers using the prototype. The system requirements or threats defined by the expert are on the left-most column, while the other columns represent the results obtained by the volunteers. The cells in green mean that the requirement or threat were correctly identified, while red means that the requirement/threat was not identified. Additionally, in the yellow cells, there are the requirements and threats that were identified by the tool, but were not identified by the expert.



System Requirements	Volunteer 1	Volunteer 2	Volunteer 3	Volunteer 4	Volunteer 5	Volunteer 6
Availability	🟩	🟩	🟩	🟩	🟩	🟩
Reliability	🟩	🟩	🟩	🟩	🟩	🟩
Interoperability	🟩	🟩	🟩	🟩	🟩	🟩
Confinement	🟩	🟩	🟩	🟩	🟩	🟩
Authorization	🟩	🟩	🟩	🟩	🟩	🟩
Device Requirements						
Physical Security	🟩	🟩	🟩	🟩	🟩	🟩
Tamper Detection	🟩	🟩	🟩	🟥	🟥	🟥
Authentication	🟩	🟩	🟩	🟩	🟩	🟩
Forgery Resistance	🟩	🟩	🟩	🟩	🟩	🟩
Transmission Requirements						
Confidentiality	🟩	🟩	🟩	🟩	🟩	🟩
Non-repudiation	🟩	🟩	🟩	🟩	🟩	🟩
Privacy	🟩	🟩	🟩	🟩	🟩	🟩
Data Freshness	🟩	🟩	🟩	🟩	🟩	🟩
Data Origin Authentication	🟥	🟥	🟥	🟩	🟥	🟥
Accountability	🟩	🟩	🟩	🟩	🟩	🟩
Authorization	🟥	🟥	🟥	🟥	🟥	🟥
	Reliability	Reliability	Reliability	Reliability	Reliability	Reliability
	Authenthication	Authenthication	Authenthication	Authenthication	Authenthication	Authenthication
	Forgery Resistance	Forgery Resistance	Forgery Resistance	Forgery Resistance	Forgery Resistance	Forgery Resistance
Database Requirements						
Confidentiality	🟩	🟩	🟩	🟩	🟩	🟩
Availability	🟩	🟩	🟩	🟩	🟩	🟩
Authentication	🟩	🟩	🟩	🟩	🟩	🟩
Privacy	🟩	🟩	🟩	🟩	🟩	🟩
Physical Security	🟥	🟥	🟥	🟥	🟥	🟥
	Non-repudiation	Non-repudiation	Non-repudiation	Non-repudiation	Non-repudiation	Non-repudiation
	Integrity	Integrity	Integrity	Integrity	Integrity	Integrity
	Authorization	Authorization	Authorization	Authorization	Authorization	Authorization
Cloud Requirements						
Confidentiality	🟩	🟩	🟩	🟩	🟩	🟩
Availability	🟩	🟩	🟩	🟩	🟩	🟩
Authentication	🟩	🟩	🟩	🟩	🟩	🟩
Privacy	🟩	🟩	🟩	🟩	🟩	🟩
Authorization	🟩	🟩	🟩	🟩	🟩	🟩
Reliability	🟩	🟩	🟩	🟩	🟩	🟩
	Integrity	Integrity	Integrity	Integrity	Integrity	Integrity
	Non-repudiation	Non-repudiation	Non-repudiation	Non-repudiation	Non-repudiation	Non-repudiation

Figure 5.1: Comparison of the requirements obtained by the expert (on the left-most column) and the volunteers. Green cells mean it was successfully identified, red cells represent a failure in the identification and yellow cells contain requirements identified by the tool but not the expert.



Physical Threats	Volunteer 1	Volunteer 2	Volunteer 3	Volunteer 4	Volunteer 5	Volunteer 6
Node Tampering	🟩	🟩	🟩	🟩	🟩	🟩
Denial of sleep	🟩	🟩	🟩	🟩	🟩	🟩
Side-channel attacks	🟥	🟥	🟥	🟥	🟥	🟥
	Jamming/interference	Jamming/interference	Jamming/interference	Jamming/interference	Jamming/interference	Jamming/interference
	Node injection	Node injection	Node injection	Node injection	Node injection	Node injection
	Node damage/destruction	Node damage/destruction	Node damage/destruction	Node damage/destruction	Node damage/destruction	Node damage/destruction
Network Threats						
Eavesdropping	🟩	🟩	🟩	🟩	🟩	🟩
Man in the Middle	🟩	🟩	🟩	🟩	🟩	🟩
Spoofing	🟩	🟩	🟩	🟩	🟩	🟩
Denial of Service	🟩	🟩	🟩	🟩	🟩	🟩
Sybil	🟩	🟩	🟩	🟩	🟩	🟩
Sinkhole	🟩	🟩	🟩	🟩	🟩	🟩
Application Threats						
Phishing	🟩	🟩	🟩	🟩	🟩	🟩
Malware	🟩	🟩	🟩	🟩	🟩	🟩
Weak Credentials	🟥	🟥	🟥	🟥	🟥	🟥
	Application DoS	Application DoS	Application DoS	Application DoS	Application DoS	Application DoS

Figure 5.2: Comparison of the threats obtained by the expert (on the left-most column) and the volunteers. The colored cells have the same meaning as in the previous comparison.

From the analysis of these results we can verify that overall, most of the requirements and attacks were successfully identified.

Most failures in the requirement analysis can be attributed to mistakes in the requirement mapping file, as some requirements might have an incorrect definition of where they are applied. In order to correct this a more extensive review and validation of the requirements file needs to be performed. This would preferably involve validation by several security experts, making adjustments and the corrections deemed necessary and then re-validate the results by repeating this testing methodology.

The requirements that were identified by the tool, but not the expert are not necessarily incorrect, but are not necessarily right either, as arguments can be made both against them or defending them and ultimately they would have to be validated on a case by case basis. Additionally, from a security point of view it is preferable to over-estimate security requirements rather than under-estimate, as this contributes to the development of a stronger, more secure system anyway.

Regarding the attack tool results, similar conclusions can be drawn regarding the yellow cells. As for the failures in identification, these can be attributed to the fact that this project does not directly include the "Side channel" threat, but it is included inside the "Jamming/interference" attack. However, the "Weak Credentials" threat is not contemplated by this dissertation and tools, so a review could be performed to add this as a new threat, on include it as part of an existing one.

Additionally, informal feedback regarding the usability of the application was also collected and none of the volunteers had any major problems using and understanding the content provided the platform. However, due to the small number of volunteers and lack of formal usability testing and feedback methodology, a more extensive usability study should be conducted as future work.

5.4 Conclusion

This chapter provided an overview of the testing methodology used to evaluate if the prototype developed implemented all the necessary requirements but most importantly, if the system and threat analysis results obtained from it are comparable to ones produced by an expert.

After specifying the testing methodology the results were analyzed and duly evaluated, concluding that overall the results obtained by the prototype were very positive. The next and final chapter will conclude this dissertation, assessing if the overall objectives were successfully achieved, and debating what future work could still be performed in the prototype and in the theme of security modeling in IoT.

Chapter 6

Conclusions and Future Work

Closing this dissertation, this chapter will discuss two main aspects, namely what are the conclusions that can be drawn from the work presented and also what future work can still be developed regarding this prototype and the field of security modeling in IoT.

6.1 Main Conclusions

Given the fast speed at which IoT was adopted and IoT based systems started becoming a common thing in our daily lives, security was mostly seen as an afterthought. This was not only due to the special limitations of IoT devices and their communication protocols, but also due to lack of knowledge and planning by the people in charge of designing and developing these systems.

This dissertation attempted to tackle this issue by developing a prototype tool to assist with the system and threat modeling processes, allowing users with varying levels of skill in security in the context of IoT to successfully and easily define the security requirements for their projects, with focus on the system design stage.

The main objectives for this dissertation consisted of:

1. Performing an analysis of security in the IoT area, defining how system and threat modeling can be used to secure a system;

2. Create a tool to assist with the system modeling process, having as basis the outputs produced by the SECURIoTEQUIREMENTS and SECURiotPRACTICES tools. This tool should output the correct requirements and guidelines to adequately secure the system;

3. Develop another tool, this time to assist with the threat modeling process. This tool would also receive system information from the SECURIoTEQUIREMENTS and SECURiotPRACTICES tools, and from that information deduce the correct attacks and attack vectors relevant to the system, producing information regarding the relevant attacks and steps to prevent them;

4. Design the tools to be easily understandable by users with a limited security knowledge. Therefore, it should be simple to use and provide the necessary information in a simple and direct manner.

The first objective was achieved during chapter 2, where an extensive review of the current state of IoT and its security and modeling methodologies was performed. This chapter also presented many key security aspects relevant to the understanding of the project and this dissertation.

Objectives two and three were also successfully achieved. Their design and requirements exploration was developed during chapter 3, while details regarding the implementation and demonstration of the system were provided over the course of chapter 4. Finally, in order to evaluate if the system was successful in producing a correct system analysis for an IoT system, the prototype was tested by comparing a hand made system analysis by an expert in the field of security in IoT with the analysis created by half a dozen volunteer testers with basic knowledge of security in IoT.

The result obtained was fairly good, as most of the requirements and attacks defined by the tool were also present in the analysis created by the expert. The minor differences can be attributed to flaws in the requirements and threat mappings, which can be easily fixed with more extensive validation and testing.

Objective 4 can also be considered to be successful, as the volunteers provided basic feedback regarding their opinion of the prototype and this feedback was overall positive. However, more extensive usability testing should be performed, as the number of testers was very low and a proper standardized usability and feedback inquiry was not performed. This was partly due to the current Pandemic situation, as these tests would have to be performed in person and it was difficult to find a reasonable amount of volunteers.

6.2 Future Work

As can be deduced from the analysis of this dissertation there is still room for improvement of the prototype.

Despite the main functionalities of the system being mostly successfully implemented, there are still improvements that can be made, mainly with regard to the requirement mappings. This would require further testing, preferably with real world scenarios, in order to more adequately evaluate if the analysis produced by this tool is fully complete and correct - meaning that the tool did not leave any important requirements behind and that all their information is valid and correct for the system in a real world scenario.

Usability testing and improvements could also be performed, perhaps by performing usability questionnaires to evaluate user feedback and suggestions.

Another direct improvement to the prototype would be to directly provide technical information regarding each attack and countermeasure proposed. However, it would be better if this was optional information, as it could potentially confuse users with less technical knowledge.

Additionally, in the future, all the SECURIoTESIGN tools are to be implemented within a single platform as a way to streamline the overall security modeling process. Despite this already being a work in progress there is still work to be done and some modifications to the existing tools might have to be performed in order to successfully combine them in a single environment.

It would also be useful to centralize all the inquiry data produced by the SECURIoTEQUIREMENTS and SECURiotPRACTICES tools. This could be done by saving the results to a database and then directly access it from this prototype or any other tool. This would give the option for users to save their results on something such as a personal account, allowing them to access their system specifications and modeling results across different devices and without having to carry and re-introduce the requirements and practices files across all devices;

Bibliography

[ACH15] Ioannis Andrea, Chrysostomos Chrysostomou, and George Hadjichristofi. Internet of Things: Security vulnerabilities and challenges. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 180–187, July 2015. 11, 13, 14

[Ang] Angular [online]. Developer: Google. Accessed: 2020-09-01. Available from: https://angular.io/. 40

[ASAAA17] Shadi Al-Sarawi, Mohammed Anbar, Kamal Alieyan, and Mahmood Alzubaidi. Internet of Things (IoT) communication protocols: Review. In *2017 8th International Conference on Information Technology (ICIT)*, pages 685–690, May 2017. 7

[BFF$^+$19] Katie Boeckl, Michael Fagan, William Fisher, Naomi Lefkovitz, Katerina N Megas, Ellen Nadeau, Danna Gabel O'Rourke, Ben Piccarreta, and Karen Scarfone. Considerations for managing Internet of Things (IoT) cybersecurity and privacy risks. Technical Report NIST IR 8228, National Institute of Standards and Technology, Gaithersburg, MD, June 2019. Available from: https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8228.pdf. 9, 10

[BFM04] Eric Byres, Matthew Franz, and Darrin Miller. The use of attack trees in assessing vulnerabilities in SCADA systems. January 2004. xxi, 24

[Boo] Bootstrap [online]. Developers: Mark Otto, Jacob Thornton. Accessed: 2020-09-01. Available from: https://getbootstrap.com/. 40

[BVJV19] Joseph Bugeja, Bahtijar Vogel, Andreas Jacobsson, and Rimpu Varshney. IoTSM: An End-to-end Security Model for IoT Ecosystems. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 267–272, March 2019. 29

[CPB] Sergio Caltagirone, Andrew Pendergast, and Christopher Betz. The Diamond Model of Intrusion Analysis. page 61. xvii, 25, 26

[CS16] Janice Cañedo and Anthony Skjellum. Using machine learning to secure IoT systems. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 219–222, December 2016. ISSN: null. 31

[d3-20] magjac/d3-graphviz [online], 2020. Developer: Gordon Smith. Accessed: 2020-09-01. Available from: https://github.com/magjac/d3-graphviz. 40

[DZS19] M. Dachyar, Teuku Yuri M. Zagloel, and L. Ranjaliba Saragih. Knowledge growth and development: internet of things (IoT) research, 2006–2018. *Heliyon*, 5(8):e02264, August 2019. Available from: http://www.sciencedirect.com/science/article/pii/S2405844019359249. 8

[FPAF18] Mario Frustaci, Pasquale Pace, Gianluca Aloi, and Giancarlo Fortino. Evaluating Critical Security Issues of the IoT World: Present and Future Challenges. *IEEE Internet of Things Journal*, 5(4):2483–2495, August 2018. 30

[GT18] Gemini George and Sabu M. Thampi. A Graph-Based Security Framework for Securing Industrial IoT Networks From Vulnerability Exploitations. *IEEE Access*, 6:43586–43601, 2018. 30

[HBH$^+$16] Elike Hodo, Xavier Bellekens, Andrew Hamilton, Pierre-Louis Dubouilh, Ephraim Iorkyase, Christos Tachtatzis, and Robert Atkinson. Threat analysis of IoT networks using artificial neural network intrusion detection system. In *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6, May 2016. ISSN: null. 31

[HCA] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. page 14. xix, 27, 28

[HCS$^+$19] Vikas Hassija, Vinay Chamola, Vikas Saxena, Divyansh Jain, Pranav Goyal, and Biplab Sikdar. A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures. *IEEE Access*, 7:82721–82743, 2019. 30

[HFH15] Md. Mahmud Hossain, Maziar Fotouhi, and Ragib Hasan. Towards an Analysis of Security Issues, Challenges, and Open Problems in the Internet of Things. In *2015 IEEE World Congress on Services*, pages 21–28, June 2015. ISSN: 2378-3818. 12, 13

[Kap] Jens-Peter Kaps. Cryptography for Ultra-Low Power Devices. page 185. 13

[LZL$^+$15] Caiming Liu, Yan Zhang, Zhonghua Li, Jiandong Zhang, Hongying Qin, and Jinquan Zeng. Dynamic Defense Architecture for the Security of the Internet of Things. In *2015 11th International Conference on Computational Intelligence and Security (CIS)*, pages 390–393, December 2015. ISSN: null. 30

[Mav] Maven – Apache Maven [online]. Developer: Apache Software Foundation. Accessed: 2020-09-01. Available from: https://maven.apache.org/. 39

[MF19] Michael Muckin and Scott C Fitch. A Threat-Driven Approach to Cyber Security. page 45, 2019. xvii, 24

[MLY05] Suvda Myagmar, Adam Lee, and William Yurcik. Threat Modeling as a Basis for Security Requirements. August 2005. 22

[Pri] PrimeNG | Angular UI Component Library [online]. Developer: PrimeTek Informatics. Accessed: 2020-09-01. Available from: https://www.primefaces.org/primeng/. 40

[RMHK$^+$16] Jose Romero-Mariona, Roger Hallman, Megan Kline, John San Miguel, Maxine Major, and Lawrence Kerr. Security in the Industrial Internet of Things - The C-SEC

Approach. In *IoTBD*, 2016. 30

[SM13] Ulya Sabeel and Saima Maqbool. Categorized Security Threats in the Wireless Sensor Networks: Countermeasures and Security Management Schemes. 2013. 14

[Spr] Java spring [online]. Developer: Pivotal Software. Accessed: 2020-09-01. Available from: https://spring.io/. 40

[Swa] API Documentation & Design Tools for Teams | Swagger [online]. Developer: SmartBear Software. Accessed: 2020-09-01. Available from: https://swagger.io/. 40

[SWJ13] Riccardo Scandariato, Kim Wuyts, and Wouter Joosen. A descriptive study of Microsoft's threat modeling technique. *Requirements Engineering*, 20, June 2013. 28

[TM¤+19] Meltem Sönmez Turan, Kerry A McKay, Çağdaş Çalık, Donghoon Chang, and Larry Bassham. Status report on the first round of the NIST lightweight cryptography standardization process. Technical Report NIST IR 8268, National Institute of Standards and Technology, Gaithersburg, MD, October 2019. Available from: https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8268.pdf. 13

Appendix A

Annexes

This annex presents the larger code snippets that were mentioned over the course of the implementation chapter. This listing in particular represents the (abridged) HTML code responsible for rendering the system modeling component mentioned in sub-section 4.4.2, using the angular ngIf and ngFor view functions.

Listing 16 Simplified snippet of the HTML code responsible for rendering the system modeling component.

```
<p-tabView>
  <p-tabPanel [header]="type" *ngFor="let type of applicationTypes; let i = index"
   ↪  [selected]="i">
    <p-accordion>
      <p-accordionTab *ngFor="let item of applicationInfoComplete[i]"
       ↪  header={{item.name}}>
        <p> {{item.description}} </p>
        <div>
          <h5> Vulnerabilities: </h5>
          <div *ngIf="item.vulnerabilities.length != 0">
            <p> The associated vulnerabilities for this security requirement: </p>
          </div>
          <div *ngFor="let vulnerabilities of item.vulnerabilities">
            <p-accordion>
              <p-accordionTab header="{{vulnerabilities.name}}">
                <p> Reason: </p>
                <p>{{vulnerabilities.description}}</p>
              </p-accordionTab>
            </p-accordion>
          (...)
          <h5> Practices </h5>
          <div *ngIf="item.practices.length != 0">
            These are practices that safeguard this requirement from being exploited:
          </div>
          <div *ngIf="item.practices.length == 0">
            No particular practices associated with this type of requirement;
          </div>
          <div *ngFor="let practices of item.practices">
            <p-accordion>
              <p-accordionTab header="{{practices.name}}" >
                <div >
                  <p> Description: </p>
                  <p> {{practices.description}}</p>
                  <p> Practices: </p>
                  <p> {{practices.measures}} </p>
                </div>
              </p-accordionTab>
            </p-accordion>
        (...)
```