UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

# Threat Modeling Solution for Internet of Things in a Web-based Security Framework

**Joana Cabral Amaral Nunes da Costa**

Dissertação para obtenção do Grau de Mestre em
**Engenharia Informática**
(2º Ciclo de Estudos)

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

**Covilhã, julho 2021**

# Acknowledgements

I would like to thank my supervisors, Professor Doutor Tiago Miguel Carrola Simões and Professor Doutor Pedro Ricardo Morais Inácio, for their guidance and availability when discussing ideas and doubts.

I am also grateful for the support and suggestions provided by Bernardo Sequeiros and Carolina Lopes.

Finally, I would like to thank my family and friends for all the motivation and encouragement in completing this stage of my academic life.

The work described in this dissertation was carried out at the Instituto de Telecomunicações, Multimedia Signal Processing - Cv Laboratory, in Universidade da Beira Interior, at Covilhã, Portugal. This research work was funded by the SECUR IoT ESIGN Project through FCT/COMPETE/FEDER under Reference Number POCI-01-0145-FEDER-030657 and by Fundação para Ciência e Tecnologia (FCT) research grant with reference BIL/ Nº12/2019-B00702.

# Resumo

A Internet das Coisas (do inglês *Internet of Things*, IoT) é um paradigma em acentuado crescimento com benefícios inegáveis para o dia a dia dos utilizadores, com uma elevada aplicação dos dispositivos da IoT em cenários sensíveis. No entanto, atualmente os dispositivos da IoT não garantem corretamente as propriedades de segurança, o que pode levar a toda uma panóplia de problemas, muitos com impacto no utilizador. Este trabalho propõe o desenvolvimento de uma ferramenta que auxilie os programadores a criar dispositivos da IoT seguros. A ferramenta é um módulo de uma *framework* denominada *Security Advising Modules* (SAM), e procura atingir o referido objetivo através da identificação de fraquezas que possam existir no software ou hardware dos dispositivos IoT.

Com o objetivo de delinear as fraquezas, consultou-se ao longo deste projeto um conjunto de bases de dados que contêm informações sobre vulnerabilidades e fraquezas encontradas em sistemas, do qual se escolheram um conjunto restrito de fraquezas a apresentar. A escolha deste conjunto deve-se a algumas das bases de dados consultadas conterem centenas de milhares de vulnerabilidades, pelo que não é exequível nem pertinente a sua completa apresentação na nossa ferramenta. Complementarmente, identificaram-se neste trabalho as questões que permitem obter informações sobre o sistema em desenvolvimento que depois nos permitem mapear as fraquezas em função das respostas do programador.

A ferramenta desenvolvida foi devidamente testada através da execução de testes automáticos, com a *framework* Selenium, e também validada por especialistas de segurança e avaliada por um conjunto de 18 utilizadores. Por fim, com base no *feedback* dos utilizadores, concluiu-se que a ferramenta desenvolvida era útil, de utilização simples e direta, e que 89% dos inquiridos nunca tinham interagido com uma ferramenta similar (nesse sentido inovadora).

# Palavras-chave

Ferramenta de Segurança, Internet das Coisas, Modelação de Ameaças, Segurança por *Design*

# Resumo alargado

## Introdução

O presente capítulo detalha a motivação e o âmbito desta dissertação, apresentando o enquadramento e descrição do problema encontrado na literatura, justificando a necessidade do desenvolvimento desta dissertação, os objetivos e as principais contribuições. Refere também sumariamente os resultados e o conhecimento provenientes desta dissertação.

## Enquadramento, Descrição do Problema e Objetivos

Atualmente, observa-se um crescimento significativo do número de dispositivos da Internet das Coisas (do inglês *Internet of Things*, IoT) existentes na rede, com implicações na vida quotidiana dos seus utilizadores. No entanto, a segurança destes dispositivos não foi totalmente assegurada pelas empresas, levantando problemas de privacidade e confidencialidade. Posto isto, é crítico para a aceitação das empresas no mercado, que estas comecem a aplicar mecanismos de segurança, devidamente identificados, nos seus dispositivos.

A exclusão da segurança dos dispositivos da IoT deve-se a dois fatores principais: 1) o consumo de energia e a capacidade de processamento necessários para aplicar mecanismos de segurança; e 2) a falta de conhecimento, por parte dos programadores de sistemas da IoT, na área de segurança. Assim, é necessário implementar uma solução que auxilie os utilizadores a produzir sistemas seguros, considerando a segurança desde a fase de *design*.

De forma a tentar solucionar o problema apresentado posteriormente, esta dissertação propõe o desenvolvimento de uma solução que auxilie na identificação de ameaças ao sistema da IoT em desenvolvimento. Nesta dissertação definem-se como objetivos: 1) a identificação de ameaças em diferentes domínios da IoT e com implementações variadas; 2) o planeamento e implementação de uma ferramenta Web que permita a identificação destas ameaças com base nas especificações de um dispositivo; e 3) a apresentação de uma divisão clara entre modelação de ataques e modelação de ameaças.

## Principais Contribuições

De acordo com os objetivos propostos, as principais contribuições desta dissertação consistem em: 1) um protótipo de uma ferramenta que apresenta um conjunto de fraquezas habitualmente encontradas no *Software* e no *Hardware* de um dispositivo; 2) uma análise extensa de ataques e vulnerabilidades, que permitiu esclarecer as diferenças entre modelação de ataques e modelação de ameaças.

## Estado da Arte

O capítulo do Estado da Arte aborda quatro tópicos principais e inclui uma comparação das ferramentas reportadas na literatura com a solução proposta. É feita a descrição do estado da IoT, onde se apresenta a sua estrutura, dos ataques (que se encontram divididos em três categorias), dos modelos de ataque e ameaça (representando modelos utilizados nesse contexto), e das vulnerabilidades e fraquezas (apresentando o seu registo e importância). Finalmente, apresentam-se um conjunto de quatro ferramentas similares à solução proposta e uma tabela sumária das caraterísticas de cada uma.

Segundo a literatura, a arquitetura dos dispositivos IoT é composta por três camadas principais: perceção, transporte e aplicação (*perception*, *transportation* e *application*, do inglês). A primeira camada engloba os dispositivos ou componentes que comunicam diretamente com o mundo físico; a camada de transporte é responsável pelo transporte de dados e/ou informação entre a camada de perceção e aplicação; por fim, a camada de aplicação processa todos os dados e produz informação útil para o utilizador. Analogamente, cada uma destas camadas está sujeita a ataques, pelo que estes são divididos em ataques aos sistemas da IoT, às comunicações e à criptografia. Adicionalmente, também se apresenta um conjunto de modelos habitualmente utilizados na modelação de ataques e ameaças, como *Attack Trees*, *Diamond Model* e *Cyber Kill Chain*. Estes modelos pretendem delinear passos que um atacante pode seguir, perceber quais os valores associados ao atacante e compreender como é que um ataque pode ser intercetado.

Por fim, apresenta-se um conjunto de vulnerabilidades e fraquezas encontradas em sistemas proprietários e/ou *open source*. Expõe-se um conjunto de bases de dados que contêm informações sobre as vulnerabilidades encontradas em sistemas proprietários, e qual a versão em que estas estão presentes, e ainda uma base de dados que retrata o conjunto de fraquezas habitualmente encontradas no desenvolvimento de sistemas, quer no *Software* quer no *Hardware*. Finaliza-se com a apresentação de quatro ferramentas cujo objetivo principal é realizar a modelação de ameaças de um determinado sistema.

## Sistema Proposto

Neste capítulo apresenta-se o planeamento da solução proposta, onde se definem os requisitos funcionais e não funcionais, a integração da solução numa plataforma de segurança já existente, denominada *Security Advising Modules* (SAM), a seleção e formulação das questões e das ameaças que podem existir num sistema, bem como a sua interligação.

No que diz respeito aos requisitos, os principais são: a solução apresenta um conjunto de ameaças com base em respostas dadas por um utilizador; a solução é modular e consegue lidar com a inserção e/ou remoção de outros módulos existentes na plataforma; e a solução está implementada na linguagem de programação necessária para executar na plataforma. Na fase de *design* do sistema, apresentam-se as relações entre a solução proposta e os módulos já existentes, ou em desenvolvimento, bem como a troca de dados entre estes. Adicionalmente, justificam-se as escolhas realizadas relativamente ao sistema proposto, nomeadamente, dos conteúdos a exibir na solução proposta, da seleção de um conjunto finito de fraquezas e da ordenação destas. Finalmente, apresenta-se a escolha de questões previamente existentes na plataforma e a formulação de novas, específicas para a solução proposta, bem como a associação entre as fraquezas e as questões escolhidas e desenvolvidas.

## Implementações do Módulo e da Plataforma

Dado que a solução proposta foi incluída numa plataforma de segurança já existente, as contribuições estão expostas em duas vertentes: incorporação da solução proposta e melhorias à plataforma.

No que diz respeito à incorporação da solução proposta, foram desenvolvidas algumas funções para obter as respostas dos módulos já existentes na plataforma, as recomendações registadas na base de dados e a seleção e ordenação das fraquezas a apresentar ao utilizador. Paralelamente, foram implementadas melhorias à plataforma que se encontram divididas em *Back-end* e *Front-end*. Especificamente, foram corrigidos diversos erros e incorporadas diversas melhorias. Por exemplo, foram implementadas novas funcionalidades que facilitam a interação com o utilizador e novas funcionalidades compreendidas na *Application Programming Interface* (API) de desenvolvimento da SAM que permitem a criação, implementação e instalação de novos módulos.

## Documentação e Avaliação

No presente capítulo, e com o intuito de avaliar a sua pertinência, definiram-se e realizaram-se um conjunto de testes à ferramenta desenvolvida e, inevitavelmente, à plataforma. O processo de testagem divide-se em três fases principais: planeamento, desenvolvimento de um guia do utilizador e de cenários a aplicar na ferramenta, e apresentação dos resultados obtidos.

De modo a avaliar a ferramenta desenvolvida, este processo foi dividido em três etapas: 1) avaliar se a ferramenta se comporta de acordo com o esperado; 2) obter a validação da ferramenta por parte de especialistas de segurança; e 3) avaliar o *feedback* dos utilizadores relativamente à utilidade e inovação da ferramenta. Adicionalmente, e com o intuito de facilitar a avaliação da ferramenta desenvolvida, produziu-se um guia de utilizador que apresenta uma breve descrição do funcionamento da plataforma e uma explicação das questões apresentadas pela ferramenta desenvolvida.

Através da análise dos resultados, pode-se concluir que: 1) a ferramenta se comportou conforme o desejado, produzindo os resultados esperados para cada conjunto de questões; 2) os especialistas de segurança confirmaram a utilidade da ferramenta, afirmando que esta produzia resultados diretos; e 3) os utilizadores finais, em geral, acharam a ferramenta fácil de utilizar e compreender, e também reforçaram o carácter inovador desta.

## Conclusões e Trabalho Futuro

Ao longo do trabalho desenvolvido foi possível chegar a conclusões secundárias. A área da IoT está ainda em expansão e não está completamente consolidada, produzindo incoerências nos trabalhos apresentados na literatura. Adicionalmente, verificou-se que o uso alternado, na literatura, das terminologias "*attack modeling*" e "*threat modeling*" para referenciar o processo de identificação do conjunto de passos necessários para um ataque bem sucedido dá azo a incoerências. Deste modo, e com o intuito de reduzir o entrave de compreensão destes dois processos, propõe-se uma possível distinção entre estas terminologias. Por fim, descobriu-se que a priorização de vulnerabilidades e fraquezas é uma tarefa manual custosa em termos temporais e que exige um elevado grau de experiência. Esta situação apresenta um obstáculo à determinação de como priorizar vulnerabilidades recentemente descobertas. Todas estas conclusões contribuem para que a principal contribuição desta dissertação seja aplicável na literatura para ajudar utilizadores inexperientes na àrea de segurança.

A solução desenvolvida encontra-se alojada numa plataforma de segurança denominada SAM, que é uma ferramenta Web que aborda diferentes tópicos de segurança. Esta solução debita um conjunto de fraquezas passíveis de serem encontradas no sistema que o utilizador está a desenvolver, obtendo informação sobre este sistema através de um conjunto de questões. Por fim, a solução desenvolvida foi devidamente testada, usando a *framework* Selenium, validada por especialistas de segurança e avaliada por utilizadores que afirmam a sua utilidade e simplicidade.

O trabalho futuro foca-se principalmente na ferramenta desenvolvida, nomeadamente na melhoria da disposição da informação aos utilizadores finais. Sugere-se o agrupamento das fraquezas apresentadas em categorias que representam os requisitos de segurança. Também se propõe a adição de uma descrição extensa, opcional, para que o utilizador final tenha acesso a mais detalhes sobre as recomendações dadas. Adicionalmente, sugere-se a criação de um conjunto de questões que permitam moldar as fraquezas aos domínios da IoT, através de uma investigação extensa destes. Por fim, propõe-se a aplicação de técnicas de Inteligência Artificial para moldar automaticamente a lógica da solução proposta. O trabalho apontado em último já está, na verdade, em desenvolvimento, e já deu azo a um artigo científico aceite.

# Abstract

The Internet of Things (IoT) is a growing paradigm that provides daily life benefits for its users, motivating a fast paced deployment of IoT devices in sensitive scenarios. However, current IoT devices do not correctly apply or integrate security controls or technology, potentially leading to a wide panoply of problems, most of them with harmful impact to the user. Thus, this work proposes the development of a tool that helps developers create properly secure IoT devices by identifying possible weaknesses in the system. This tool consists of a module of a framework, denominated *Security Advising Modules* (SAM) in the scope of this work, and achieves the referred objective by identifying possible weaknesses found in the software and hardware of IoT devices.

To define the weaknesses, a set of databases containing information about vulnerabilities and weaknesses found in a system were investigated throughout this project, and a restricted set of weaknesses to be presented was chosen. Since some databases contain hundreds of thousands of vulnerabilities, it was neither feasible nor pertinent to present them completely in the developed tool. Additionally, the questions to retrieve system information were identified in this work, allowing us to map the chosen weaknesses to the answers given by the developer to those questions.

The tool developed was properly tested by running automated tests, with the Selenium framework, and also validated by security experts and evaluated by a set of 18 users. Finally, based on user feedback, it was concluded that the developed tool was useful, simple and straightforward to use, and that 89% of respondents had never interacted with a similar tool (adding, in this way, to the innovative character).

## Keywords

Internet of Things, Security by Design, Security Framework, Threat Modeling

# Contents

**List of Figures** **xvi**

**List of Tables** **xvii**

**Acronyms** **xx**

**1 Introduction** **1**

1.1 Motivation and Scope . . . . . 1

1.2 Problem Statement and Objectives . . . . . 2

1.3 Proposed Approach and Main Contributions . . . . . 2

1.4 Document Organization . . . . . 3

**2 Background and Related Work** **5**

2.1 Introduction . . . . . 5

2.2 State of IoT . . . . . 5

2.2.1 Perception Layer . . . . . 5

2.2.2 Transportation Layer . . . . . 6

2.2.3 Application Layer . . . . . 7

2.3 Attack Types . . . . . 7

2.3.1 Cryptography . . . . . 8

2.3.2 Communications . . . . . 8

2.3.3 IoT Systems . . . . . 9

2.4 Attack and Threat Models . . . . . 10

2.4.1 Attack Tree . . . . . 10

2.4.2 Diamond Model . . . . . 11

2.4.3 Cyber Kill Chain . . . . . 13

2.5 Vulnerabilities and Weaknesses . . . . . 15

2.5.1 Databases . . . . . 15

2.5.2 Scores . . . . . 16

2.6 Related Work . . . . . 19

2.6.1 Main Literature Contributions . . . . . 19

2.6.2 Framework Comparison . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
2.7 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

**3 System Proposal** **23**
3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
3.2 Scope and Requirements Analysis . . . . . . . . . . . . . . . . . . . . . . . 23
3.3 System Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
3.4 Threat and Question Selection . . . . . . . . . . . . . . . . . . . . . . . . 25
3.4.1 Vulnerability and Weakness Selection . . . . . . . . . . . . . . . . . . 25
3.4.2 Question Selection and Formulation . . . . . . . . . . . . . . . . . . . 27
3.5 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

**4 Module and Framework Implementations** **35**
4.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
4.2 Module Incorporation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
4.3 Framework Enhancement . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
4.3.1 Back-end Improvements Overview . . . . . . . . . . . . . . . . . . . . 37
4.3.2 Front-end Improvements Overview . . . . . . . . . . . . . . . . . . . 39
4.4 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40

**5 Documentation, Demonstration, and Evaluation** **41**
5.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
5.2 Methodology . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
5.3 User Guide and IoT Scenarios . . . . . . . . . . . . . . . . . . . . . . . . 42
5.3.1 User Guide . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
5.3.2 IoT Scenarios . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
5.4 Results Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
5.5 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53

**6 Conclusions and Future Work** **55**
6.1 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
6.2 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56

**Bibliography** **57**

# List of Figures

2.1 AT example, with the primary goal of opening a safe, adapted from [Sch99]. 11

2.2 ADT example, with the root node being ensuring data confidentiality, adapted from [KMRS10]. The green bordered rectangles are defense nodes and red bordered circles are attack nodes. The dotted lines represent countermeasures. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.3 Diamond model usage example, adapted from [CPB13]. . . . . . . . . . . . 13

2.4 CKC application example, adapted from [HCA11]. . . . . . . . . . . . . . . 14

2.5 CVSS version 2 metrics for each group, adapted from [oIRT21a]. . . . . . . 17

2.6 CVSS version 3 metrics for each group, adapted from [oIRT21b]. . . . . . . 17

2.7 CWSS metrics for each group, used to prioritize weaknesses in a system. Adapted from [Cor18]. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

3.1 Overview of finished and under development modules at the time of writing of this dissertation. The *blue* rectangle represents the SAM platform, *white* rectangles represent the concluded modules, the *orange* rectangle denotes the proposed module, and the *red* rectangle depicts a module being developed concurrently. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

3.2 Framework front-end, back-end, and proposed module data flow diagram. The *orange* rectangle represents the proposed module. . . . . . . . . . . 25

3.3 Summary of the correlation between the questions gathered from the SRE module and weaknesses. Blue rectangles represent the questions, orange rectangles depict the answer given to that question, and the white rectangles denote the CWE identifier. . . . . . . . . . . . . . . . . . . . . . . . 28

3.4 Summary of the correlation between the questions gathered from the SBPG module and weaknesses. Blue rectangles represent the questions, orange rectangles depict the answer given to that question, and the white rectangles denote the CWE identifier. . . . . . . . . . . . . . . . . . . . . . . . 29

3.5 Summary of the correlation between the questions for the design phase and weaknesses. Blue rectangles represent the questions, orange rectangles depict the answer given to that question, and the white rectangles denote the CWE identifier. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

3.6 Summary of the correlation between the questions for post-design phases and weaknesses. Blue rectangles represent the questions, orange rectangles depict the answer given to that question, and the white rectangles denote the CWE identifier. . . . . . . . . . . . . . . . . . . . . . . . . . . . 32

4.1 Demonstration of the SAM platform workflow when selecting a module, answering questions, and viewing the outputs and their correlation with the back-end. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

5.1 High-level design of the Smart Transportation Monitoring scenario. . . . . 46

5.2 High-level scheme of the Smart Electrical Vehicle Home Charging scenario. 47

5.3 High-level design of the Smart Car scenario. . . . . . . . . . . . . . . . . . 48

5.4 High-level scheme of the Smart Pacemaker scenario. . . . . . . . . . . . . 49

5.5 High-level design of the Smart Irrigation System scenario. . . . . . . . . . 49

5.6 High-level scheme of the Smart Watch scenario. . . . . . . . . . . . . . . . 50

5.7 High-level design of the Smart Factory scenario. . . . . . . . . . . . . . . . 51

5.8 Graph summarizing the answers given to the evaluation questionnaire of the proposed module according to the scale from 1 to 7, where 1 is the most negative evaluation, and 7 is the most positive. *Green*, *Light Blue*, and *Dark Blue* represent options 2, 3, and 4, respectively. *Orange*, *Pink*, and *Cream* represent options 5, 6, and 7, respectively. . . . . . . . . . . . . . . . . . . . 52

# List of Tables

2.1 Proposed framework comparison with related work. Empty cells correspond to component unavailability. . . . . . . . . . . . . . . . . . . . . . . . . 21

# Acronyms

| | |
|---|---|
| **4G** | Fourth Generation |
| **5G** | Fifth Generation |
| **6LoWPAN** | IPv6 over Low-Power Wireless Personal Area Network |
| **ACISM** | Assessment of the Correct Integration of Security Mechanisms |
| **ACM** | Association for Computing Machinery |
| **ACT** | Attack Countermeasure Tree |
| **ADT** | Attack Defense Tree |
| **AG** | Attack Graph |
| **API** | Application Programming Interface |
| **AT** | Attack Tree |
| **CCS** | Computing Classification System |
| **CKC** | Cyber Kill Chain |
| **CPU** | Central Processing Unit |
| **CSRE** | Cloud Security Requirements Elicitation |
| **CVE** | Common Vulnerabilities and Exposures |
| **CVSS** | Common Vulnerability Scoring System |
| **CWE** | Common Weakness Enumeration |
| **CWSS** | Common Weakness Scoring System |
| **DBMS** | Database Management System |
| **DDoS** | Distributed Denial of Service |
| **DoS** | Denial of Service |
| **HARM** | Hierarchical Attack Representation Model |
| **HDL** | Hardware Description Language |
| **HTTP** | HyperText Transfer Protocol |
| **IEEE** | Institute of Electrical and Electronics Engineers |

| | |
|---|---|
| **IoT** | Internet of Things |
| **IP** | Internet Protocol |
| **IPv6** | Internet Protocol version 6 |
| **IT** | Information Technology |
| **JSON** | JavaScript Object Notation |
| **LWCAR** | LightWeight Cryptographic Algorithm Recommendation |
| **MitM** | Man-in-the-Middle |
| **MQTT** | Message Queuing Telemetry Transport |
| **NIAC** | National Infrastructure Assurance Council |
| **NIST** | National Institute of Standards and Technology |
| **NVD** | National Vulnerability Database |
| **OSAT** | Outsourced Semiconductor Assembly and Test |
| **OpenSSF** | Open Source Software Foundation |
| **OWASP** | Open Web Application Security Project |
| **PDF** | Portable Document Format |
| **PHP** | Hypertext Preprocessor |
| **RFID** | Radio-Frequency Identification |
| **SAM** | Security Advising Modules |
| **SBPG** | Security Best Pratice Guidelines |
| **SQL** | Structured Query Language |
| **SRE** | Security Requirements Elicitation |
| **TLS** | Transport Layer Security |
| **TSP** | Trust, Security and Privacy |
| **WPAN** | Wireless Personal Area Network |
| **WSN** | Wireless Sensor Networks |
| **XML** | Extensible Markup Language |

# Chapter 1

# Introduction

## 1.1 Motivation and Scope

The combination of quotidian physical objects used in different domains with the Internet degenerates into the new (though rapidly becoming commonplace) Internet of Things (IoT) paradigm [ZMKd17], a set of heterogeneous devices that collect data, intercommunicate and interact with the ideal purpose of improving quotidian tasks performance [SOBG19]. Therefore, IoT is ubiquitous and pervasive, interacting with the physical world by gathering information or actuating, and possibly harming human lives if not correctly secured.

Since IoT is a new paradigm that gathers private information about users or animals, about locations or complex conditions, etc., it is crucial to ensure the triad of security related properties known as data confidentiality, integrity and availability. Nonetheless, it is common knowledge that securing a device has performance and energy costs which manufacturers can not neglect, and which sometimes even hinders the adoption of security related technologies. Furthermore, considering security while designing IoT devices (since many security flaws come from a poor design [JDSTN19]) helps to surpass the previously presented problem.

IoT is very heterogeneous, meaning also that there is no unique solution to secure all the devices, presenting the problem of identifying the device-specific threats. Therefore, it is essential to develop an automated way of predicting security measures based on the device application domain and sourcing from system specifications and conditions, tackling the aforementioned issues. This provides the basic motivation of the work described herein.

The scope of this work falls in the intersection of the areas of information security, cyber-physical systems, and system modeling. Under the 2012 version of the Association for Computing Machinery (ACM) Computing Classification System (CCS), a de facto standard for computer science, the scope of the master's project, reflected in this dissertation, can be defined by the categories named:

- **Security and privacy ~ Systems security**;
- *Security and privacy ~ Human and societal aspects of security and privacy*;
- *Security and privacy ~ Security services.*

## 1.2 Problem Statement and Objectives

Nowadays, many IoT devices are being deployed and introduced into the market by different companies that actually take security into consideration [CBA17], thought still with different degrees of adequacy. Furthermore, security mechanisms tend to reduce the device performance and energy, which are crucial points in IoT. However, the demand kept growing, and the industry produced many insecure devices to meet this demand.

IoT devices are present in out everyday life, gathering sensitive information about the user and, sometimes, these devices play a crucial role in the life of the users or in society in general. Therefore, it it necessary to assure security of these devices in order to prevent major catastrophic events. Moreover, skilled planning is critical to ensure the security of the device, consisting of specification reporting, threats evaluation, and security testing. Thus, developing an IoT-based threat modeling tool that considers characteristics from the device and its implementation is necessary. This dissertation tackles this open issue via its main contribution: providing an IoT-specific framework responsible for modeling the threats to an individual IoT device or software. Therefore, the main objectives of this dissertation are presented as follows:

- Identify threats perceived in different IoT domains and for varying implementations;
- Plan, implement, and incorporate into a security platform a Web-based tool that allows threat identification based on user and system specifications;
- Provide a clear division between attack modeling and threat modeling, since they are often used interchangeably but used for different purposes.

## 1.3 Proposed Approach and Main Contributions

The main contribution of this dissertation is an automated web-based solution that outputs possible threats that can be found in IoT systems and software, which does not require security expertise and provides prioritization of these threats. This solution asks the end-user questions regarding an IoT system, processes the given answers using Python, and provides possible threats to that system. The development of this solution requires the analysis and review of currently used IoT technologies, attack and threat modeling techniques, and threat databases and scoring systems. In conjunction with examining the security framework, this thorough research allows selecting the information to be displayed and the implementation and integration of the proposed solution. Finally, the proposed solution will be duly tested, validated, and evaluated by security specialists and end-users. These were precisely the steps taken along the project and thus comprise the proposed approach to solve the problem.

During the development of this dissertation, particularly considering the analysis and review of the state-of-the-art, other open issues were found. Standardization of IoT methodologies and cryptographic mechanisms is still an open issue in literature and hinders applying these techniques in IoT systems. Additionally, there is a lack of standardization between the enterprise and research environments regarding the attack and threat modeling, limiting the comprehension of distinctive characteristics between these techniques. Finally, the prioritization of vulnerabilities and weaknesses is manually performed by organizations and can be automatized using machine learning techniques. This issue led to the development of a Common Vulnerability Scoring System (CVSS) calculator with embedded artificial intelligence, which will be the subject of a paper. Though this contribution is not described herein because it was not initially planned, it can be considered as a collateral of this work. Moreover, initial steps towards the construction of tools with embedded artificial intelligence was taken during the development of the project, in collaboration with other researchers, leading to the paper [LCS+21], submitted and accepted for publication in a scientific peer-reviewed conference.

## 1.4 Document Organization

The structure of this document follows closely the way the work was performed in accordance with the initially delineated in the work plan:

- Chapter 1 – **Introduction** – presents the scope and motivation of the project behind this dissertation, as well as open issues found in literature and objectives to solve them, ending up with the document organization;
- Chapter 2 – **Background and Related Work** – reviews available IoT systems, IoT targeted attacks, attack and threat models commonly used in security assessment and finally, it describes a set of databases built to map security vulnerabilities and weaknesses into a multi-numeric score;
- Chapter 3 – **System Proposal** – exposes the elicited requirements for the tool that materializes part of the proposed solution, as well as system design and integration in a greater security platform, and threat selection, prioritization, and question association;
- Chapter 4 – **Module and Framework Implementations** – discusses how the developed tool was incorporated into a major security framework, and describes the enhancements made to the back-end and front-end of that platform;
- Chapter 5 – **Documentation, Demonstration, and Evaluation** – contains the user guide, including a brief demonstration of the (entire) platform with focus on the developed tool, IoT scenarios, the planning for the tool evaluation, and the obtained results and feedback;
- Chapter 6 – **Conclusions and Future Work** – focuses on the main conclusions drawn from this dissertation and presents future improvements.

# Chapter 2

# Background and Related Work

## 2.1 Introduction

This chapter (briefly) covers the current state of the art of IoT, explaining the systems and communication protocols involved in section 2.2. Additionally, it discusses some attacks regarding cryptographic functions, communication protocols, and IoT systems, in section 2.3, illustrating different attack and threat modeling techniques in section 2.4. Moreover, it presents vulnerability and weakness databases and their scoring systems in section 2.5, concluding with the description and comparison of work related to the proposed solution 2.6.

## 2.2 State of IoT

The architecture of IoT systems is divided into three different layers [FPAF18]: perception, transportation, and application (the layer designations may vary, but the underlying idea remains). ISO 20924 defines IoT as the *"infrastructure of interconnected entities, people, systems, and information resources and services that processes and reacts to information from the physical and virtual world."* Additionally, this standard defines an IoT device as an *''entity of the IoT system that interacts and communicates with the physical world through sensing or actuating."* [fS18].

### 2.2.1 Perception Layer

A vast amount of heterogeneous devices with limited capabilities characterize the perception layer. This layer is in direct contact with the physical environment, being also called the edge layer, and can monitor and change its physical aspects. Therefore, pointing out a clear division between IoT devices like sensors and actuators.

Sensors can monitor a set of physical conditions, such as temperature, humidity, pressure, among others. The transportation layer sends this information to the application layer, which is responsible for all the processing. Afterward, a decision is sent to the edge devices, either by storing the data or sending an action to an actuator. An actuator has a physical impact on the environment, changing its conditions. For example, it can raise the temperature, reduce humidity, among others. Thus, the information that reaches the actuators must be the correct one. Otherwise, they have the wrong impact on the environment, creating risky or life-threatening situations.

IoT edge devices have memory and processing constraints (small amounts of memory and low capacity Central Processing Unit (CPU)) and are mostly power restricted. These characteristics suggest that it is unreasonable to run the currently used cryptographic algorithms, which rely on a heavy workload.

### 2.2.2 Transportation Layer

Device heterogeneity, lossy connections, and a massive amount of devices characterize IoT networks. Some devices perform two different functionalities, such as data acquisition and transportation, making it unclear which devices will permanently stay in IoT networks. Additionally, the connection established between two nodes is not point-to-point but point-to-multiple or multiple-to-point [WTB+12].

Communication protocols in IoT need to handle lossy connections to avoid packet loss and shorten their frame size to support device heterogeneity. Most of the Internet transport protocols rely on a set of devices that are permanently on the network and are strictly routing-oriented. This condition does not relate with IoT paradigm, leading to the creation of new protocols (described below) and the modification of older ones.

**Message Queuing Telemetry Transport (MQTT)** is a messaging protocol initially developed in 1999 to monitor oil and gas pipe remotely [Tea20]. MQTT is a publish/subscribe lightweight protocol, requiring minimal resources and bandwidth. Devices that desire to obtain information about a sensor on the network need to subscribe to the service provided by it. Every time the sensor informs about a physical event (*e.g.*, temperature), it sends this information to its subscribers. MQTT provides reliable message delivery, ensuring that the user does not miss any crucial data, and supports unreliable networks. Finally, this protocol uses Transport Layer Security (TLS) to encrypt messages and authenticate clients using OAuth [MQT20].

**Zigbee** is a Wireless Personal Area Network (WPAN) that works under low-power and processing conditions. This technology mainly focuses on reducing the power consumption associated with other protocols working under WPAN. Despite being developed to work with proprietary devices, it also functions with different communication protocols, providing interoperability. Zigbee networking protocol offers a set of security features ensuring message authentication and encryption [Far08].

**Bluetooth** is a wireless technology initially developed in 1989, being present in almost all personal devices. This technology allows data transfer at a short-range but was not designed with the IoT paradigm in mind. In July 2017, Bluetooth launched Bluetooth Mesh, which supports IoT connections in a simple, efficient, and flexible manner. Bluetooth Mesh was designed with security in mind, providing message authentication and encryption with two different keys [DSLA17].

**Wireless Sensor Networks (WSN)** defines as a network of nodes that interact with the physical environment, either by sensing or actuating on it. Many sensors exist in the network, meaning that a considerable amount of information will be collected and transmitted in these networks. It is mandatory to have Trust, Security and Privacy (TSP) to secure IoT networks. TSP technology supports message authentication, encryption, access control, and identity authentication. Thus, being a preferred technology to use in the IoT context [pt14].

**Radio-Frequency Identification (RFID)** is a technology that works with a set of tags and readers, using radio-frequency waves to identify an object. A tag is a passive element, meaning that it does not have power source. Meanwhile, a reader is an active element, responsible for providing enough power to the tag to read the information it stores. This technology has been around for approximately 50 years [DOL07]. If a user has a reader close to a tag, he can read the stored data and possibly have access to sensitive information. Tag processing and memory capabilities are very reduced, meaning it can not be secure by itself [Tew20].

**IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN)** is a protocol focused on saving energy and works with resource-constrained devices. It works on Internet Protocol version 6 (IPv6) with a reduced frame size to minimize power consumption via header compression [PTC16]. Since it uses IPv6 as its transport protocol, 6LoWPAN will face the same security and privacy issues as IPv6.

### 2.2.3 Application Layer

The application layer offers all the processing and analyzing capabilities of the IoT systems. It supplies the necessary amount of resources to create the intelligence (therefore easiness) provided by the perception layer. Existing cloud services or any device capable of processing a considerable amount of information can be considered as part of the application layer. Since this layer is known for high processing capacity and no power failures, current cryptographic algorithms can be applied, and most of its vulnerabilities are known.

Cloud services are delivered via many computing units with huge processing capacities and almost no power constraints (can handle power failures for long periods). Cloud is a technology that already exists for years and is now being leveraged to handle the application layer of IoT applications in many cases. Attackers have been exploiting its vulnerabilities for long, meaning that these systems can be considered the most secure among IoT.

## 2.3 Attack Types

When attacking a system, the attacker must know its vulnerabilities, chosen from different categories. The attacker either chooses to attack cryptographic implementations, communications, or system vulnerabilities at a specific period.

### 2.3.1 Cryptography

Cryptographic attacks are limited and mathematically defined, so it is straightforward to list them all, unless something was missed by the scheme designers. Firstly, cryptanalysts presented four attack models [SP18], and later two other models emerged with a modification of two of the existing models. The remaining part of this section briefly describes these models.

**Ciphertext-Only Attack** happens when the attacker attempts to acquire the plaintext or cipher key, only knowing the ciphertext. The attacker only has the passive capability of viewing ciphertext. Thus, being the hardest to be performed and the most realistic one [Bir11c].

**Known Plaintext Attack** occurs when an attacker has access to known-plaintext and corresponding ciphertext pairs, which are limited. This attack scenario increases the capabilities of the attacker, raising its practicability [Bir11d].

**Chosen Ciphertext Attack** is a scenario in which the attacker intends to obtain a specific plaintext by previously having access to multiple ciphertexts and their corresponding plaintexts. This attack model assumes that the attacker has access to the decryption function to decrypt the chosen ciphertexts. This scenario is suitable for public-key encryption because the encryption key is publicly available, and the attacker can produce ciphertexts, but not vice-versa [Bir05a].

**Chosen Plaintext Attack** is similar to Chosen Ciphertext Attack, but in this turn, the attacker has access to plaintext and the corresponding ciphertext. Therefore, the attacker needs to have access to the encryption function to produce the ciphertexts. This attack scenario is less practicable than Known Plaintext Attack [Bir05b].

**Adaptive Chosen Ciphertext Attack** operates as a Chosen Ciphertext Attack scenario in which the attacker can choose the inputs for the decryption function based on previously obtained ciphertexts. This scenario is more unrealistic than Chosen Ciphertext Attack because it raises the capabilities associated with the attacker [Bir11a].

**Adaptive Chosen Plaintext Attack** performance is similar to Adaptive Chosen Ciphertext Attack, but the attacker has access to the previously chosen plaintext and corresponding ciphertext. Therefore, this attack is more powerful, making it less realistic [Bir11b].

### 2.3.2 Communications

Independently of the communication protocol used, these are always susceptible to Man-in-the-Middle (MitM) attacks, either *passive* or active. The first one, also known as eavesdropping, happens when the attacker has a passive role and records the messages exchanged through that channel. On the other hand, *active* implies the possibility of communication modification between two devices. MitM attacks happen when an attacker can intercept the communication between two entities and identify itself as legitimate [PIS+17].

Like any other layer in the IoT network, communications are susceptible to Denial of Service (DoS) or Distributed Denial of Service (DDoS) attacks, which occur when the user does not receive or transmit information, or receives incomplete messages, due to the overflow or deletion of network packets or due to inflicted malfunction of network services or forwarding technologies. E.g., the attacker sends a massive amount of packets throughout the network or installs a device in that same network to receive all the packets preventing them from reaching their destination [Yu14].

A Sybil attack is another typical attack performed under IoT networks when system authentication is weak. In this case, the attacker introduces a malicious node or controls a legitimate node in the network and impersonates other nodes or generates fake identities [MAL$^+$19]. If no cryptography is applied to the communications, this type of attack makes the network insecure.

It is not feasible to describe all the communication layer attacks due to space limitations. However, it should be mentioned that these attacks are typically executed by a third person over the communication channel. Thus, most of the attacks described throughout this section suit under the category of attacks that involve a third person/device that needs to be legitimately identified in the network.

### 2.3.3 IoT Systems

Inevitably, IoT systems inherit the properties of the existing electronic devices, meaning that they are also susceptible to electronic devices vulnerabilities. Additionally, many IoT devices are resource and power-constrained, meaning that they are exposed to even a more significant number of attacks. Some of the IoT systems are also positioned in locations where they are not easily accessible and have seldom physical maintenance.

Computational resources are the basis of cryptographic security, meaning that the usage of cryptographic algorithms implies that they have a considerable amount of resources. Lack of capabilities in IoT devices leads to the impossibility of using the widely known cryptographic algorithms. Therefore, research and development of new cryptographic algorithms that offer the same strength but work under constrained conditions is needed.

Placing IoT devices in isolated locations makes these systems vulnerable to physical tampering, due to the lack of human presence to prevent the attacker from acquiring hardware information. It is essential to prevent the attacker from either reaching the device or tampering with its hardware.

IoT manufacturers are responsible for maintaining their systems. Nonetheless, to ease the maintenance process, manufacturers usually discard security, thus leaving another open entry for attackers to exploit and gain access to the device.

Another vulnerable characteristic is the power restriction of most of the IoT systems, making these devices susceptible to battery exhaustion under specific conditions. The attacker can simulate legitimate requests and continually send them to a power-constrained device, leading to battery exhaustion. As said before, some of the devices are located in isolated places, meaning that exchanging batteries is an expensive and time costly process.

IoT systems are as vulnerable to social engineering attacks as any other user-controlled device. The credentials to access the device are under the possession of a user who lacks the knowledge to dodge well-performed social engineering attacks. Unintentionally, the user gives the attacker access to the system.

## 2.4 Attack and Threat Models

Along the time, different ways of representing (modeling) susceptibility to attacks or the way that attacks are performed have been proposed. Different widely used attack and threat models will be detailed throughout this section to help defining the modeling technique for the proposed solution.

### 2.4.1 Attack Tree

One of the most widely adopted models to represent attacks and vulnerabilities to a specific system is named Attack Tree (AT) due to its significant advantage of easily being replaced through a textual representation. The attacker goal is the root node of such tree, while the remaining nodes are the steps required to reach this goal. Leaf nodes are atomic steps the attacker may accomplish to achieve his main goal. The connections between each node represent a logic operator, being `AND` or `OR` [Sch99, Sch01]. Figure 2.1 shows a simple example of an AT, whose main goal is to open a safe.

AT models are still frequently used by system administrators to perform risk assessment. It is possible to add the financial cost to complete each step on the AT, identifying expensive attacks that require higher monetary resources, thus narrowing the number of users that can perform a specific attack. The AT model can also contain information regarding the need for special equipment to achieve a particular step.

AT models are very dynamic and allow the user to perceive which will be the most attacked components, which units will need to be more thoroughly secured, and if the attacker needs special equipment. This analysis process makes it possible for system administrators to secure their systems correctly without overextending their budget.

The attacker viewpoint is the main focus of the presented model. As researchers felt the need to add the system defenses to represent the real dynamic of the systems, this model was updated, leading to the development of Attack Defense Tree (ADT). In such model, there are two types of nodes which represent either an attack or a defense. Furthermore, the connection between two nodes is either a refinement or countermeasure [KMRS10]. ADT is an improvement to the existing AT; thus, ADT models have the same interpretation as attack tree models. Figure 2.2 shows a truncated example of an ADT, whose root node

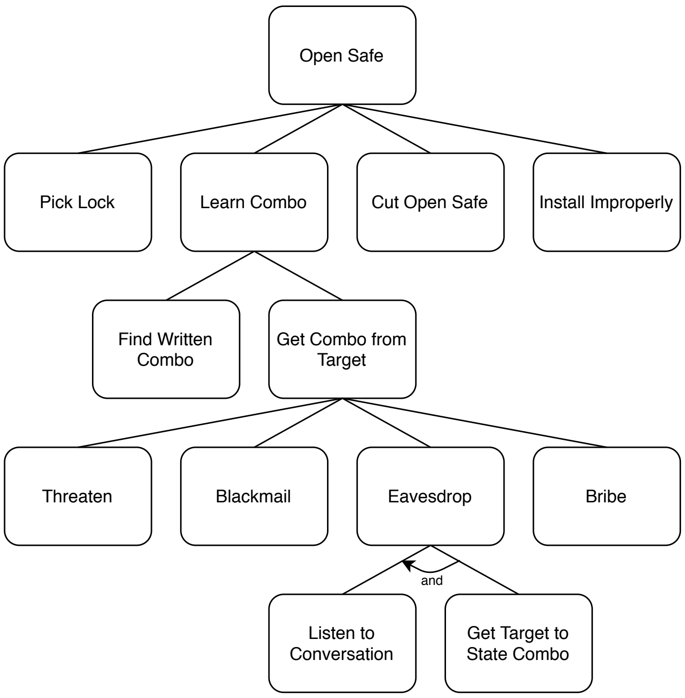


Figure 2.1: AT example, with the primary goal of opening a safe, adapted from [Sch99].

is applying a defensive procedure such as data confidentiality.

Lately, Attack Countermeasure Tree (ACT) models have been gaining traction also, proposing a combinatorial analysis between attacks and countermeasures. As a novelty, ACT brought the possibility to represent countermeasures in all the tree nodes, not only at the leaf nodes [RKT10].

An attack is directly associated with a fault: a defect in a system can be the entrance of an attacker, and an attack can cause a fault. This idea lead to the rise *Attack Fault Tree* models also. This type of model provides both AT and Fault Tree outcomes, contributing to a trade-off between security and security [KS17].

AT, which was initially presented in 1999, has evolved in many ways to other models (as shown throughout this section). It is possible to find modifications from the original AT and subsequent models that were not presented here.

### 2.4.2 Diamond Model

The Diamond Model describes the core features for any intrusion event: adversary, capability, victim, and infrastructure. Each element is connected to another by an edge representing the fundamental relationships between the core features. Diamond Model works under the conception that each intrusion event has an adversary working towards a goal using its capabilities and infrastructure against a victim [CPB13]. Finally, the diamond model presents some meta-features which are not essential to evaluate an event but may help the defender to understand the occurring event.

**Adversary** (one of the core features) is the actor responsible for the attack execution and

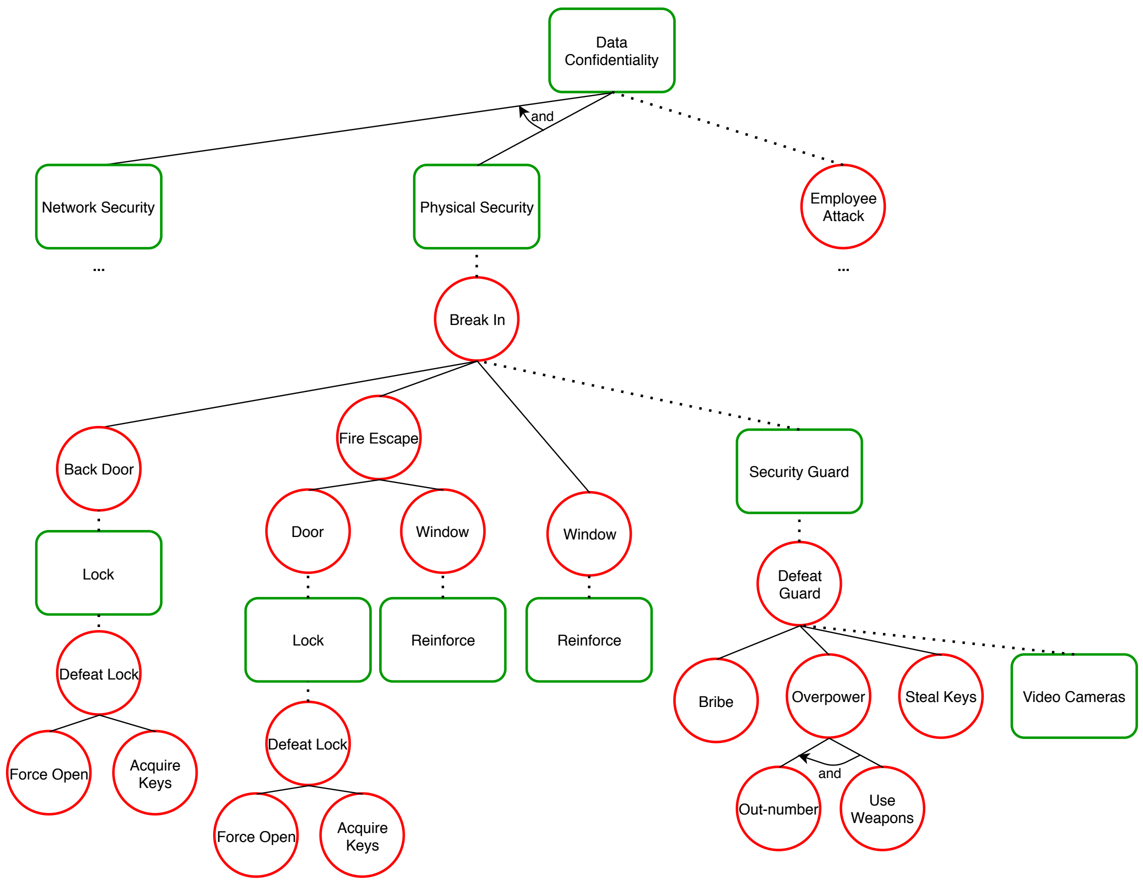


Figure 2.2: ADT example, with the root node being ensuring data confidentiality, adapted from [KMRS10]. The green bordered rectangles are defense nodes and red bordered circles are attack nodes. The dotted lines represent countermeasures.

accomplishment. The attack success probability highly relies on the attacker and, unlike other features, it is challenging to classify the attacker knowledge. The attacker can be a unique person or a group of persons and can be internal or external to the system.

The tools and techniques utilized during the intrusion event by the attacker are called **capability**. All the procedures should be registered, even if the strategies used are clumsy.

**Infrastructure** describes the physical and logical structures of the adversary to deliver its capabilities and extract information from the victim. This feature is where the adversary financial capacity is the main focus, and infrastructure increases if the adversary is a group.

Finally, vulnerabilities are exploited against the target of the adversary (**victim**). This victim could either be a persona or an asset, making this distinction crucial.

The Diamond Model mainly focuses on the attacker/adversary, meaning that it is decisive in an intrusion event. It is imperative to design possible attackers to defend the system better. Figure 2.3 shows a possible malware installation on the victim network (1), then a specialist performs malware reversion to expose the command and control domain (2). Resolution of the domain leads to the exposure of the underlying Internet Protocol (IP) address hosting the controller (3). Firewall logs reveal other compromised hosts in the network (4); finally, the IP address registration reveals the adversary details (5).

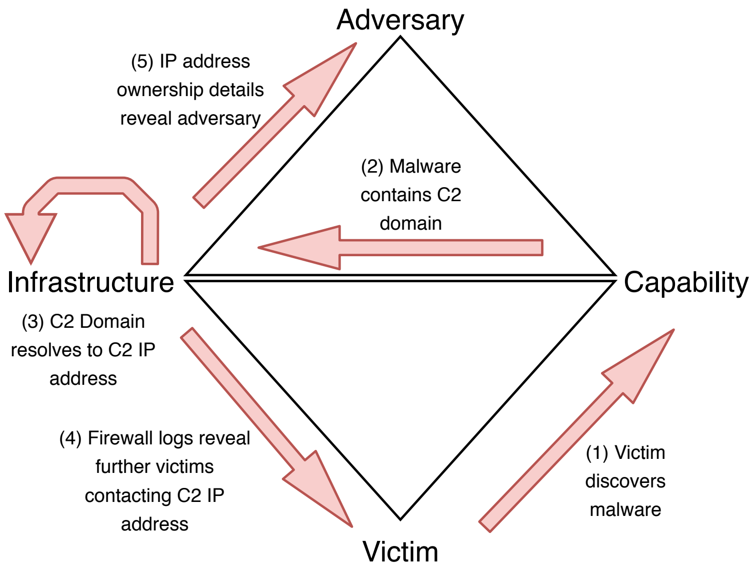

Adversary
(5) IP address ownership details reveal adversary
(2) Malware contains C2 domain
Infrastructure
Capability
(3) C2 Domain resolves to C2 IP address
(4) Firewall logs reveal further victims contacting C2 IP address
(1) Victim discovers malware
Victim

Figure 2.3: Diamond model usage example, adapted from [CPB13].

## 2.4.3 Cyber Kill Chain

As the community develops countermeasures for simple attacks, these are evolving into multiple stage attacks. These attacks can deceive the countermeasures, meaning that analyzing an attack only by itself is not enough. The Cyber Kill Chain (CKC) model intends to help fight against multi-stage attacks by defining the steps taken throughout an intrusion event.

*Reconnaissance*, *Weaponize*, *Delivery*, *Exploitation*, *Installation*, *Command and Control*, and *Acts on Objective* are the steps defined in CKC [YR15]. While a system is facing an ongoing attack, this model can help intersecting those attacks. It also helps the security analysts to have a mindset closer to the attacker. As this model works precisely as a chain, the attacker can not accomplish the following steps if the defender cuts out one step.

**Reconnaissance** is the process of collecting the maximum quantity of information about the potential target. The attacker identifies the possible targets, evaluates the most suitable victim, and profiles the target. Information gathered throughout this stage is used in the following steps. Passive and active are the two types of reconnaissance an attacker can perform, that differ from each other in the way the target perceives their occurrence. While doing passive reconnaissance, the attacker is gathering information without getting noticed. However, performing active reconnaissance opens the possibility of detection by the victim. Usually, attackers initially perform passive reconnaissance and move to active later on.

The next step is to utilize the information acquired during reconnaissance to design a backdoor and a penetration plan (**Weaponize**). This stage involves two components: client and server-side. The attacker designs a technique to deliver the first element to the target, then execute it, and create a network connection to the server (that exists in the attacker infrastructure). The server-side is used by the attacker to send commands to the client-side and observe its results on the client-side.

At this point, the attacker only designed the application to implement in the client and server-side. One critical stage of the CKC is to deliver this application to the target and is responsible for an efficient and effective cyber-attack. Contrary to the steps done before, the **Delivery** stage is a risky task because it leaves traces, resorting often to using anonymous services.

After the delivery completion, it is essential to trigger the exploit to install the payload silently. The target must have the software or operating system with a version in which the exploit still works, without any additional security mechanism capable of detecting the payload, and successfully triggering the exploit. **Exploitation** uses known vulnerabilities to deliver the application silently.

The attacker already delivered the payload to the target computer, and the next step is to install this application. Nowadays, malware is multi-staged, and it heavily relies on programs that install and run the malware on the victim system. During the **Installation** phase, there is always the risk of the malicious programs being detected and reported to defensive countermeasures.

**Command and Control** systems are another critical part of the CKC and are responsible for sending the attacker commands to other systems. There are various types of architectures, such as centralized, decentralized, and social networks. Making legitimate traffic indistinguishable from malicious is one of the vital parts of the Command and Control communications.

Finally, after gaining control of a system, the attacker can execute the intended commands. It can either perform a mass attack, affecting as many targets as possible, or a targeted attack, where only critical systems are attacked (**Act on Objective**).

Figure 2.4 shows an example of early intrusion detection, specifically at the delivery stage. For this specific intrusion, the defender should perform an analysis to understand why the attacker could reach that stage before being detected. Also, equally necessary, the defender must gather as much information from the unsuccessful intrusions to ease understanding of the current defenses. This process might help in future attacks where the attacker might circumvent the existing countermeasures.

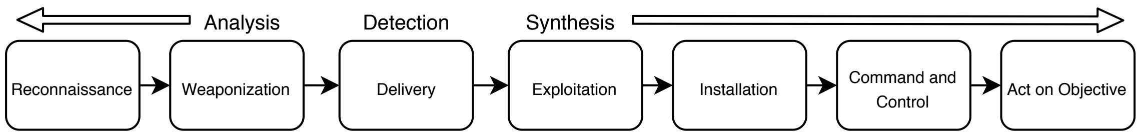


Figure 2.4: CKC application example, adapted from [HCA11].

## 2.5 Vulnerabilities and Weaknesses

It is essential to consider both software and hardware vulnerabilities to identify IoT system threats. A vulnerability is described as “a hole or a weakness in the application (...) that allows an attacker to cause harm to the stakeholders of an application.” according to Open Web Application Security Project (OWASP). The main purpose of this work is to identify which vulnerabilities threaten the IoT system being designed. Therefore, some of the available databases presented in subsection 2.5.1 and scoring systems presented in subsection 2.5.2 were considered.

### 2.5.1 Databases

Before 1999, identifying known vulnerabilities regarding a specific tool was an onerous task, requiring that experts would acquire information from multiple databases and ascertain if the vulnerabilities were not repeated across these databases. Motivated by this issue, Mann and Christey proposed a Common Vulnerability Enumeration that assigns a unique vulnerability identifier to each publicly known vulnerability and is freely available, which prompted the creation of the Common Vulnerabilities and Exposures (CVE) [Dav99, MG02] database. It contains 153,955 entries and provides a description and an identifier for each entry.

National Vulnerability Database (NVD) [Nat21, Len13] was instituted in 2005 and, in conjunction with other vulnerability databases such as CVE and Common Weakness Enumeration (CWE), provides impact metrics, namely the CVSS scores, references to remediation assistance, description, and respective CVE identifier. In addition, National Institute of Standards and Technology (NIST) does an ongoing analysis of the CVE database and assigns each vulnerability the CVSS base metrics, helping users to understand each issue severity and perform risk assessment analysis. NVD is updated daily and, at the time of writing of this dissertation, it had 153,923 entries (very recent vulnerabilities do not have a CVSS impact score).

VulDB [vul21b, vul21a] is a vulnerability database that combines experienced moderators and the crowd to identify zero-day vulnerabilities. It allows the registration of any person who wants to access this information but with limited access. The user starts with a free license, giving access to 10 Application Programming Interface (API) credits per day, which can be upgraded to a commercial or enterprise license, containing 200 and 10,000 API credits per day, respectively. This database had 175,243 entries in june 2021. Each database entry (referring to a vulnerability) contains the summary, details, CVSS version 3 and version 2 base and temp score.

CWE [The21b] is a catalog that contains common weaknesses found in software and hardware components. A *weakness* is defined as a flaw, fault, bug, or other error injected during the design and development phases that could result in a vulnerability if left unaddressed. The main objective of this database is to help developers and practitioners to describe, identify, and prevent software and hardware vulnerabilities. Additionally, CWE provides a top 25 most critical weaknesses found in software components, which helps developers prioritize the weaknesses to tackle. Finally, each weakness has a type, such as base, variant, class, category, presenting its relationship with other weaknesses, and can have multiple CVE entries associated. As of this project, this catalog contains 918 weaknesses, which were revised in December 2020.

OWASP [OWA21b, OWA21a] is a non-profit foundation that aims to improve software security. One of their projects identifies vulnerabilities, namely on Web applications, motivating the identification of 60 distinct vulnerabilities. Additionally, they provide the top 10 vulnerabilities found on Web applications, which eases the prioritization of risks. Besides this, OWASP has projects to identify attacks and countermeasures to help developers and security experts build secure applications.

Open Source Software Foundation (OpenSSF) [The21a] Vulnerability Disclosures Working Group is an emerging project that started in July 2020, which intends to help open-source software maintainers improve their software security. This project has the following objectives: 1) document and promote proper vulnerability disclosure and coordination practices; 2) encourage the development and adoption of an open-source software vulnerability exchange system. However, the documentation and systems developed under this project are not currently open access.

### 2.5.2 Scores

Each company had its scoring system for its products and followed its enterprise guidelines without publicly saying how it scored its vulnerabilities [MSR06]. This generated a significant problem for administrators who managed various systems and applications (e.g., vulnerability prioritization for heterogeneous systems). To tackle this issue, US National Infrastructure Assurance Council (NIAC) proposed using an interoperable scoring system – CVSS – that estimates and quantifies the impact of software vulnerabilities.

The previously described scoring system is divided into three metric groups: 1) *Base*, describing the properties of a vulnerability that do not change over time; 2) *Temporal*, including the properties that change over time; 3) *Environmental*, considering the representative properties of the Information Technology (IT) environment. In addition, this scoring system has two versions, version 2 and version 3, that differ from each other in the metrics considered for each group. Figures 2.5 and 2.6 present the metrics found in each metric group for version 2 and version 3, respectively.

*Confidentiality*, *Integrity*, and *Availability Impact and Requirement* are common metrics between version 2 and version 3. The **Impact** metrics evaluate the potential damage that can be caused by that security property being violated. The **Requirement** metrics are used on the environment group to give importance to a specific security property, i.e., assuring the availability property is more important than the integrity property. **Remediation Level** and **Report Confidence** are also existent in both versions and are

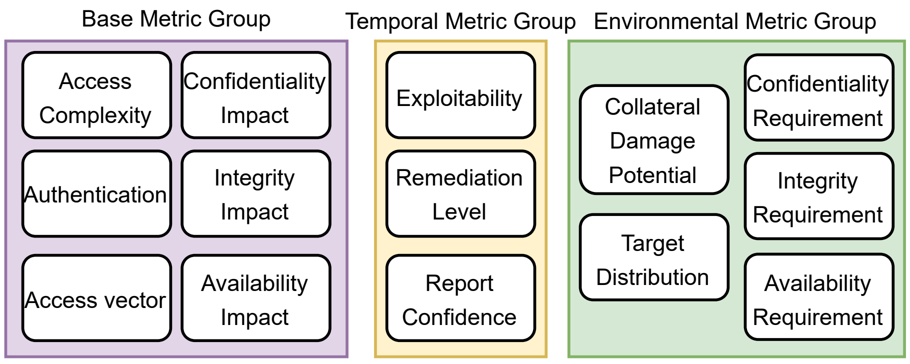


Figure 2.5: CVSS version 2 metrics for each group, adapted from [oIRT21a].

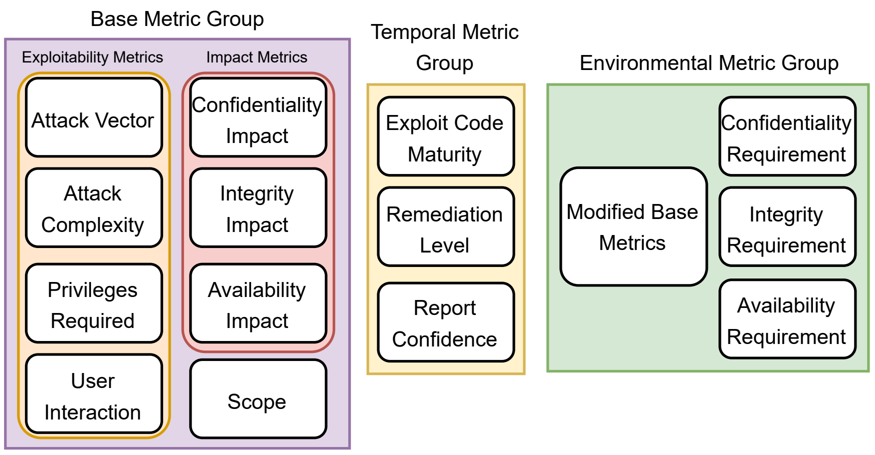


Figure 2.6: CVSS version 3 metrics for each group, adapted from [oIRT21b].

components of the temporal metric group. The first one quantifies the possibility of fixing the corresponding vulnerability, and the other one measures the degree of confidence in the existence of the corresponding vulnerability.

Considering the exclusive metrics in version 2 for the base metric group, the **Access Vector** indicates how the attacker can access the system; the **Access Complexity** reflects the complexity associated with the exploitation after having access to the system; the **Authentication** quantifies the number of authentications the attacker has to do before he can exploit the corresponding vulnerability. Additionally, the **Exploitability** metric quantifies the amount of code and techniques available to exploit the corresponding vulnerability; the **Collateral Damage Potential** reflects the potential loss of life and physical assets through damage; the **Target Distribution** evaluates the percentage of the environment that is at risk.

As for CVSS version 3, the base metric group was divided into two subgroups, **Exploitability** and **Impact**. The metrics under the Impact subgroup were described previously, and the Exploitability group reflects the characteristics of the vulnerable component. Though not included in these two subgroups, the **Scope** metric assesses if the vulnerability impacts the components beyond its security scope. The **Attack Vector** and **Attack Complexity** metrics are similar to the Access Vector and Access Complexity metrics existent in version 2, respectively. The Authentication metric was replaced by two other metrics that evaluate the level of privileges the attacker needs to exploit a vulnerability (**Privileges Required**) and if exploiting the vulnerability requires user interaction (**User Interaction**). **Exploit Code Maturity** evaluates the same characteristic as the Exploitability metric from version 2, and **Modified Base Metrics** allows the end-user to modify the metrics from the base group.

The previously presented scoring system is designed explicitly for vulnerabilities and does not apply to the weaknesses found on the CWE database. Therefore, the need to develop a system to classify these weaknesses prompted the implementation of a new scoring system called Common Weakness Scoring System (CWSS). It helps developers prioritize common bugs and mistakes found in their code by providing a consistent and flexible mechanism that enables organizations to reflect their business context [eS14].

Like CVSS, this scoring system is divided into three metric groups: 1) *Base Finding*, 2) *Attack Surface*, 3) *Environmental* [Cor18]. The first one classifies the inherent risk of the weakness, the accuracy of the finding, and the attacker capacity when controlling this weakness. The attack surface metrics evaluate the number of barriers and defenses the attacker has to surpass to gain access. Finally, the environmental metric allows the developer or system administrator to have the IT environment reflected on the scoring system.

The abovementioned metric groups are subdivided into smaller metrics, as presented in figure 2.7, to ease the calculation of the CWSS score. Regarding the base finding group, **Technical Impact** reflects the potential result of exploiting the system weakness, **Acquired Privilege** represents the privileges the attacker acquires, and **Acquired Privilege Layer** displays the operational layer to which the attacker gains access when exploiting the weakness successfully. Additionally, **Internal Control Effectiveness** describes the control ability to turn the weakness unable to exploit, and **Finding Confidence** indicates the confidence associated with the observed weakness being exploitable.

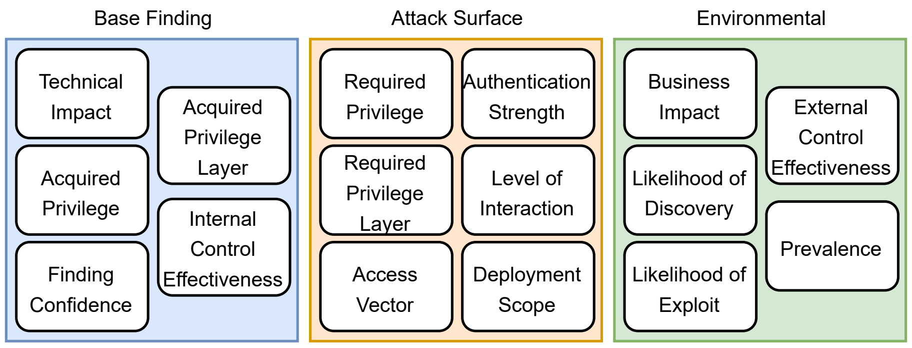


Figure 2.7: CWSS metrics for each group, used to prioritize weaknesses in a system. Adapted from [Cor18].

As previously mentioned, the attack surface conditions reflect the difficulties the attacker has to surpass to gain access. In this case, **Required Privilege** and **Required Privilege Layer** refer to the type of privileges and the operational layer, respectively, that the attacker must have to attack this weakness successfully. Furthermore, **Attack Vector** has the same meaning as in the CVSS version 3 scoring system; **Authentication Strength** defines the authentication procedures the attacker must surpass; **Level of Interaction** describes the degree of interaction between the attacker and the user. Finally, the **Deployment Scope** reflects the extension of the software or hardware that manifests the weakness.

Providing the impact magnitude on the business domain, if the weakness is successfully exploited, is provided by the **Business Impact** metric. Moreover, evaluating the environmental metric requires the determination of the **Likelihood of Discovery**, which describes the probability of the attacker finding the weakness; the **Likelihood of Exploit**, explaining the plausibility of the weakness being exploited by an attacker; the **External Control Effectiveness**, reflecting the mechanisms that hinder the attacker exploitation; and the **Prevalence**, which represents the weakness recurrence in the system.

## 2.6 Related Work

Companies produced a significant amount of IoT devices without considering security, leading to several problems lately, namely sensitive information breaches. Recently, researchers started developing frameworks to help businesses improve the security of IoT systems. This section presents some of these frameworks.

### 2.6.1 Main Literature Contributions

Mohsin *et al.* [MAH+16] proposed a formal framework for security analysis in IoT, called IoTSAT, focused in modeling the IoT system generic behavior. It uses the device configurations, network topologies, user policies, and IoT specific attack surface to perform behavior modeling. This model is used to measure the system resilience against potential attacks and identify threat vectors and specific techniques. This framework automatically reveals complex attack vector chains based on the adversary goals, which can be flexibly chosen by the user considering essential security requirements and mission objectives. IoTSAT develops two higher-level models applicable to generic IoT systems: the first model formalizes interaction among different IoT entities, and the second model captures IoT specific threat classifications. The first model considers five critical sets of IoT elements: environment features, sensors, controllers, actuators commands, and actuators. The techniques performed by the attacker depend on capabilities, device network configurations, and exposed IoT vulnerabilities. The second model reports these procedures.

Casola *et al.* [CDRV19] proposed an automated framework to perform threat modeling to different assets in IoT. It relies upon IoT system architectural components and security properties, enabling applicable threats identification, security risk analysis and evaluation, and proper countermeasures selection. The required user pieces of information are a high-level architectural model of the IoT system and technical information on the implementation/deployment. This framework performs its inference throughout three steps, namely: system modeling, threat modeling and risk analysis, and security controls identification. In system modeling, the user must specify each network device type and its layer, designing a graph with the components, type, and connections. On the contrary, threat modeling is an entirely automated process, made possible by using a security knowledge database, which maps threats to assets, countermeasures, among others. In this process, the tool classifies threats based on the asset and maps them to the standard security controls representing the countermeasures to adopt. At this point, the framework has identified all relevant threats to the system. This tool evaluates 16 parameters, such as threat likelihood, threat agent factors, and business factors, to perform risk analysis and security controls identification. In this stage, the analyst only has to evaluate the business factors. This framework classifies threats as LOW, MEDIUM, or HIGH, outputting the list of security controls to apply.

Alharbi *et al.* [AA18] proposed a framework to analyze the security associated with smart cameras, covering five major components. The primary purpose of this framework is to demonstrate vulnerabilities that exist on home monitoring smart cameras and how they affect user security and privacy. A camera monitors its surroundings and sends notifications to the user application whenever an event is detected. The user can log in to his/her account and watch the camera video streams that can be sent directly from the camera or through a server. Based on this, the video stream data, personally identifiable information, and smart cameras are assets that demand protection. While performing threat modeling, the tool considers four types of threats: unauthorized video stream retrieval, tamper with smart cameras, unauthorized account hijack, and unauthorized capture of personally identifiable information. The five major components considered in this framework are smart camera security, smart camera physical and network security, web interface security, mobile application vulnerability to information leakage and account hijack, and privacy policy and agreements. With this information, the framework infers possible attacks for each component based on the threats mentioned above. This framework considers android applications security analysis, smart camera applications analysis, OWASP testing guide, Institute of Electrical and Electronics Engineers (IEEE) IoT security best practices, and network attackers. As a result, this tool produces a set of test cases for each major component that a smart camera must surpass to be considered secure.

Ge *et al.* [GHGK17] proposed a framework for modeling and assessing IoT security, which identifies all the possible attack paths in an IoT system. It constructs a graphical security model and a security evaluator to automate the security analysis of the IoT. The security model captures potential attack paths in the network, and the security evaluator uses various security metrics to assess the security and interacts with an analytic modeling and evaluation tool to output the analysis results. This tool consists of five phases: data processing, security model generation, security visualization, security analysis, and model updates. Through these stages, the framework can identify all possible attack paths, evaluate the security level of IoT through security metrics, and assess multiple defense strategies. In data processing, security decision-maker provides system information and security metrics. The system information includes the subnets, their network topology, and vulnerability information for each node. In the next stage, the tool generates an extended Hierarchical Attack Representation Model (HARM) of the IoT network to compute all the potential attack paths. Security visualization produces a set of Attack Graph (AG) to visualize the three layers compounding the HARM. Attack path, security metrics, and other information are the input for the security evaluator, which will output the analysis results directly or generate a file that the user can employ in another tool. Finally, if the IoT system has changed due to defense strategies, the model is updated. The framework performs the previous phases to re-analyze the security and suggests which part of the system is the most vulnerable.

### 2.6.2 Framework Comparison

The solution developed in the scope of this dissertation is automated and does not require the intervention of a security expert. Initially, the proposed solution will ask questions to a user regarding the IoT system to be designed, evaluating the user given answers to present the threats regarding the system. Then, it considers the usage of Hardware Description Language (HDL) and the existence of hardware components by displaying a set of hardware-specific threats obtained from the CWE database. Finally, the CWSS score to classify and prioritize threats is used in the proposed solution.

Table 2.1 highlights and summarizes the main differences between the proposed solution and the ones found in the literature. It shows when the platform prioritizes the presented threats, if it displays hardware threats, if a solution presents the threats without user intervention, and if it requires reduced security knowledge to obtain the results.

| **Solution** | **Threat Priorization** | **Hardware Threats** | **Automated Approach** | **Reduced Security Knowledge** |
|---|---|---|---|---|
| Mohsin *et al.* | ✓ | | ✓ | ✓ |
| Casola *et al.* | ✓ | | ✓ | ✓ |
| Alharbi *et al.* | | | | ✓ |
| Ge *et al.* | | | ✓ | |
| Ours | ✓ | ✓ | ✓ | ✓ |

Table 2.1: Proposed framework comparison with related work. Empty cells correspond to component unavailability.

The proposed solution uses the knowledge stored in the previously presented databases to display a set of threats for software and hardware components. Specifically, the CWE database was updated in December 2020, when hardware weaknesses were introduced into their system, supporting this unique characteristic found in the proposed solution. Furthermore, it considers the scoring systems to prioritize the threats displayed to the user, allowing the user to define which threats to tackle first.

Some solutions require that a security specialist intercedes in the attack modeling process. However, hiring a security specialist is expensive for companies, meaning that they can not continuously use platforms with this requirement. The proposed solution is designed to help users with basic security knowledge and can be used by most IoT manufacturers. Finally, the proposed solution is an automated approach that only needs the user to provide data about the system through questions and correlates the acquired data with the possible threats.

## 2.7 Conclusion

Regarding security issues, multiple tools and techniques can be used to perform attack and threat modeling. Threat modeling consists of identifying threats that can be found in a system. However, attack modeling consists of distinguishing steps that must be accomplished to succeed in an attack. These two procedures are not independent since the attacker cannot perform an attack if the underlying threat does not exist in the system. Thus, the proposed solution only considers the identification of threats that commonly reside in a system, reducing the success chances of the attacker. Specifically, only the OWASP and CWE databases were considered and only weaknesses were displayed (vulnerabilities were mapped to weaknesses). Finally, regarding similar tools, the main contribution of the proposed solution is to present weaknesses considering hardware components.

# Chapter 3

# System Proposal

## 3.1 Introduction

The design phase is an essential step to ensure the proper development of a proposed solution. Though the main idea behind what had to be done in this scope was approximately delineated from the start, it was necessary to make it consistent and approached in the proper manner in terms of software development. Therefore, section 3.2 describes functional and non-functional requirements, section 3.3 exhibits the system integration into Security Advising Modules (SAM), and section 3.4 details the threat selection and question identification.

## 3.2 Scope and Requirements Analysis

The main objective of the solution proposed under the scope of this dissertation is to provide information about typical threats found on the system the developer specifies. The system under development is described by answering a series of questions across multiple modules and plugins in a security framework called SAM, developed under the scope of the SECURIoTESIGN project. Since it was to be incorporated into SAM, the system is a module of this framework (sometimes, *module* is also used to refer to this system hereinafter).

Regarding functional requirements, the proposed solution must obtain the answers given to previously available modules, namely the modules named Security Requirements Elicitation (SRE) and Security Best Pratice Guidelines (SBPG) [SSS+20, SLA+20]. Additionally, if needed, this solution might have supplementary questions in order to more adequately infer underlying threats. The proposed solution needs to output recommendations to the developer, which are either *vulnerabilities* or *weaknesses* that the system might contain based on the given description. Moreover, the information outputted from this solution must contain a unique identifier, allowing the developer to search for details about the vulnerability or weakness. Furthermore, it also should provide a brief description about the recommendation, enabling the developer to perceive the relation between the vulnerability or weakness and the system characteristics. Finally, this solution should prioritize vulnerabilities or weaknesses, allowing the developer to understand which of these should be tackled first.

Regarding nonfunctional requirements, the proposed module was developed using Python. Furthermore, the API provided by the SAM framework, built to ease the implementation and deployment of modules, was widely adopted and extended. Additionally, it is expected that the solution provides the results to the developer in real-time, meaning that recommendations are given within a reasonable timeframe (within seconds, not minutes). Finally, this solution should follow a modular approach and remain functional even if modules are added or updated.

## 3.3 System Design

The proposed solution was included in a wider security framework composed of several modules or tools. Therefore, to thoroughly understand our solution, this framework and already existent modules are described. According to this framework, a *module* is defined as a component related to a specific security topic that presents questions to the user and provides recommendations. Additionally, a *plugin* is similar to a module but does not provide questions to the user.

The existent modules cover various topics that allow the developer to understand which security properties should be applied in their IoT and Cloud devices, which best practices should be followed in these devices, and which lightweight cryptographic algorithms can be used in systems with constrained resources and power. At the time of writing of this dissertation, two other components were under development, whose purpose was to complement the previously presented ones, displaying to the developer a set of threats that might exist in their system and how they can test for its presence.

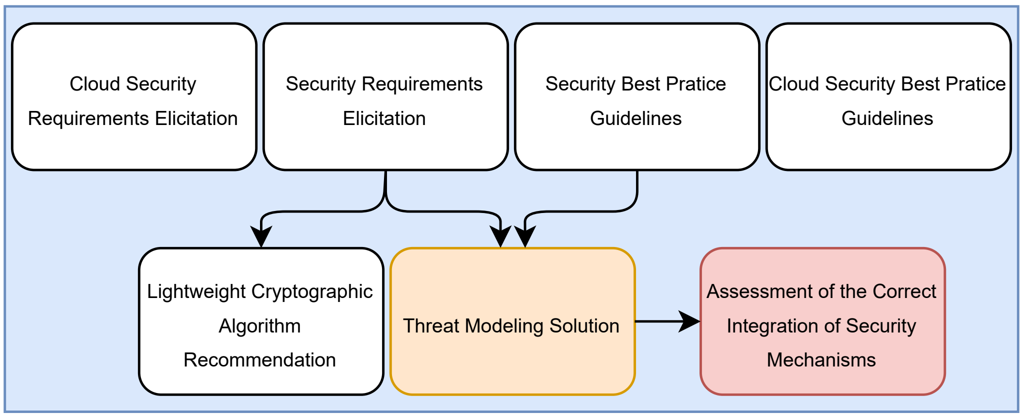

Cloud Security Requirements Elicitation
Security Requirements Elicitation
Security Best Pratice Guidelines
Cloud Security Best Pratice Guidelines
Lightweight Cryptographic Algorithm Recommendation
Threat Modeling Solution
Assessment of the Correct Integration of Security Mechanisms

Figure 3.1: Overview of finished and under development modules at the time of writing of this dissertation. The *blue* rectangle represents the SAM platform, *white* rectangles represent the concluded modules, the *orange* rectangle denotes the proposed module, and the *red* rectangle depicts a module being developed concurrently.

Figure 3.1 shows an overview of the security framework components and their connections. Analyzing this figure, it is possible to observe that the LightWeight Cryptographic Algorithm Recommendation (LWCAR) receives the security properties outputted from the SRE module. Additionally, the proposed module, called *Threat Modeling Solution*, receives the answers to the questions used in the SRE and SBPG [SSS⁺20, SLA⁺20] modules. Finally, the threats outputted from our module are an input for the Assessment of the Correct Integration of Security Mechanisms (ACISM), which outputs system tests to check for the presence of threats.

To carefully understand how these modules interact with the framework and the user, the data flow is displayed in figure 3.2. The entry point is the framework front-end, with which the user interacts and selects the security topic. After the user finishes the questionnaire of a module, the front-end communicates with the back-end to get the recommendations from the selected security topic. Then, the back-end gathers the available recommendations and answers from the database and transfers this data to the respective module. Based on the given answers, the module selects the strictly necessary recommendations and sends them to the back-end. Finally, the back-end stores this data in the database and transmits it to the front-end, displaying it to the user.

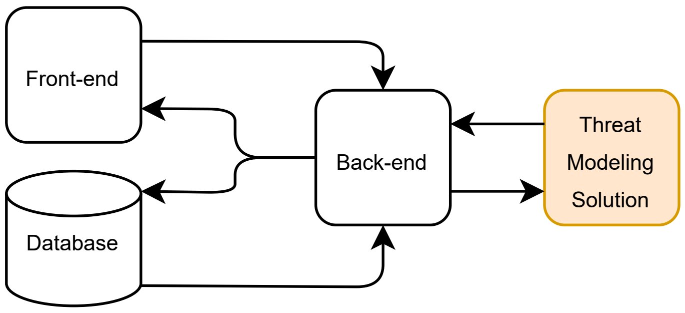


Figure 3.2: Framework front-end, back-end, and proposed module data flow diagram. The *orange* rectangle represents the proposed module.

## 3.4 Threat and Question Selection

As the primary purpose of the developed module is to provide a set of threats based on answers given by the user, the modeling process is divided into threat selection and question selection.

### 3.4.1 Vulnerability and Weakness Selection

As presented in chapter 2, there are multiple databases available from which the intended threats can be selected. However, the balance between the number of threats to display to the user and their spread throughout multiple categories was considered in the proposed solution. Furthermore, vulnerabilities listed on NVD, CVE, and VulDB are specific to the software, vendor, and version. Data that might not be completely available during the design phase of an IoT system. Besides this, there was the need to find threats regarding the hardware components due to the presence of hardware modules in IoT environment.

The two finite lists that do not increase daily and are not specific to the software are OWASP Vulnerabilities and CWE Weaknesses. Furthermore, CWE presents a set of weaknesses that are specific to hardware. Therefore, opting to advise the end-user on what weaknesses he/she can find in his/her IoT system.

First, the top 25 weaknesses found in software are presented to the user. Then, vulnerabilities from OWASP were mapped to the top 25 weaknesses. This process was done by searching for the OWASP vulnerability name or description and finding a matching weakness in the CWE catalog. With this, the 38 remaining vulnerabilities from the OWASP list were submitted to the same searching process and clustered if they had the same parent. Finally, hardware weaknesses were added from CWE Hardware View, selecting only weaknesses for *System on a Chip* platforms and with *DoS: Amplification*, *DoS: Resource Consumption (Other)*, and *Gain Privileges or Assume Identity*, common attacks in IoT systems.

Finally, to help to identify which weaknesses to prioritize when enumerating the list of weaknesses in an IoT system, the available CWSS score for the entries in CWE was used. However, only the top 25 weaknesses contain an associated score. Therefore, the remaining weaknesses were classified using a methodology [Cor20] presented by The MITRE Corporation to tackle this issue.

Since the aforementioned method is dependant on CVE information related to the weakness, only the weaknesses with associated CVE identifiers could be classified. Initially, it is essential to gather information about the number of CVE identifiers associated with the weaknesses. Therefore, information about the maximum and the minimum number of CVE identifiers associated with one weakness was gathered. Additionally, the number of CVE identifier for each weakness was calculated and the following formula (defined in [Cor20]) was used to calculate the corresponding frequency:

$$Fr(ID) = (count(\text{cve} \in ID) - min(Freq)) \div (max(Freq) - min(Freq)), \quad (3.1)$$

where $ID$ is the CWE identifier for a specific weakness, cve are the CVE identifiers, and $Freq$ is the set containing the number of CVE associated with each weakness.

It is also crucial to calculate the severity of a specific weakness, which is done by calculating the severity score for each weakness in our set. Therefore, the following formula (defined in [Cor20]) was applied to calculate the corresponding severity:

$$Sv(ID) = (average\_ID - min(\text{cvss})) \div (max(\text{cvss}) - min(\text{cvss})), \quad (3.2)$$

where $ID$ is the CWE identifier for a specific weakness, $average\_ID$ is the CVSS score average for all the CVE identifiers related to this weakness, and cvss is the set of CVSS scores linked to this weakness.

The final score for each weakness is obtained by multiplying the frequency score ($Fr$) by the severity score ($Sv$). Finally, to ease the interpretation of these scores, this final score is multiplied by 100, allowing each CWSS score to have a value between 0 and 100. Weaknesses that do not have associated CVE identifiers were ordered by their CWE identifier.

### 3.4.2 Question Selection and Formulation

As the proposed module works with an existing security framework, it is essential to reuse questions already asked in other components of this framework. Specifically, and considering previously described functional requirements, the selected questions were those available on the SRE and SBPG modules [SSS+20, SLA+20].

From the SRE module, the answers to the following questions were used:

- **Q1** - "What is the domain type for your IoT system?";
- **Q2** - "Will the system have a user?";
- **Q3** - "Will the system have user login?";
- **Q4** - "Will this information be sent to an entity?";
- **Q5** - "Will it store data in a database?";
- **Q6** - "Will the system receive regular updates?";
- **Q7** - "Will the system work with third-party software?";
- **Q8** - "Is there a possibility of the communications being eavesdropped?";
- **Q9** - "Can someone try to impersonate a user to gain access to private information?";
- **Q10** - "Can someone with bad intentions gain physical access to the location where this software will be running and obtain private information?";
- **Q11** - "Can someone gain physical access to the machine where the system operates or some of the system components and perform some modification to its hardware?".

Figure 3.3 displays an overview of the relation between the answers given to the previously described questions and the CWE weakness identifier.

Besides the first question, that lets the user pick between seven possibilities of IoT domains, the remaining previous questions only have two possible answers: `Yes` or `No`. The first question (**Q1**) allows the identification of the weakness that refers to uncontrolled resource usage, based on the battery usage of the IoT devices on the considered domain. The questions that are related to the existence of a user (**Q2**) and login (**Q3**) are relevant to warn the developer about the possibility of poorly managing password storage, the need for a proper authentication mechanism, and the plausibility of using the same credential across multiple devices.

Regarding data exchanged between the system and an external entity (**Q4**), the user is advised about insecure identifier mechanisms, unexpected behavior from the external entity, and the possibility of the attackers using this channel to transmit encoded information. Additionally, when storing data permanently in a database (**Q5**), it is essential to ensure

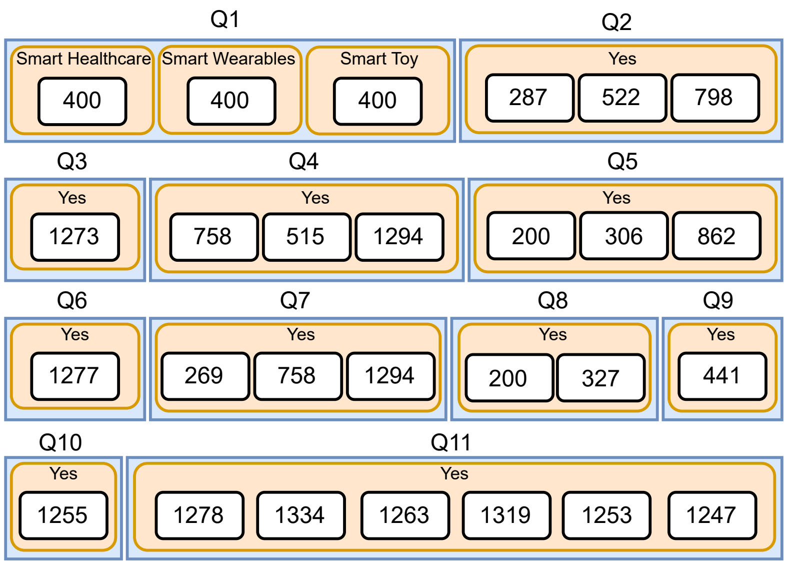


Figure 3.3: Summary of the correlation between the questions gathered from the SRE module and weaknesses. Blue rectangles represent the questions, orange rectangles depict the answer given to that question, and the white rectangles denote the CWE identifier.

that only authorized users have access to it and that each user has the strictly necessary permissions (e.g., if the user is reading from the database, it should only have reading permissions).

If the system does not receive regular updates (**Q6**), it is critical to guarantee that the developers can update or patch the system to avoid future exploitation in case of a vulnerability exposure. In addition, when working with third-party software (**Q7**), the developer must ensure that its behavior is known, does not give more than necessary permissions, and has identifiers for the entity transactions.

If there is any chance that the communications might be eavesdropped on (**Q8**), the developer should be informed to not use broken or risky cryptographic algorithms and that he/she should be aware of the exposure of sensitive information to an unauthorized actor. Furthermore, one of the main issues already encountered in IoT is the easiness of user impersonation (**Q9**). Therefore, suggesting that the developers must implement a mechanism that uniquely identifies the device to which the request was sent.

Finally, when the attacker might gain physical access to the system (**Q10**), he/she can monitor a set of properties, such as power consumption and heat production, to identify which actions are being performed by this system. This set of actions might include attack known as Side-Channel Attacks. Furthermore, if the attacker can perform modifications to the hardware (**Q11**), the developers should be aware of multiple security procedures that can be applied to prevent the attacker from gathering or altering data.

As previously referred, questions from another existent module (referred to as SBPG) were also used. From all the available questions, the following were selected:

- **Q1** - "What is the type of data storage?";

- **Q2** - “Which will be the programming languages used in the implementation of the system?”;
- **Q3** - “Will the system allow user input forms?”;
- **Q4** - “Will the system allow file uploading?”;
- **Q5** - “Will the system store or communicate logs?”.

Figure 3.4 displays an overview of the relation between the answers given to the previously described questions and the CWE weakness identifier.

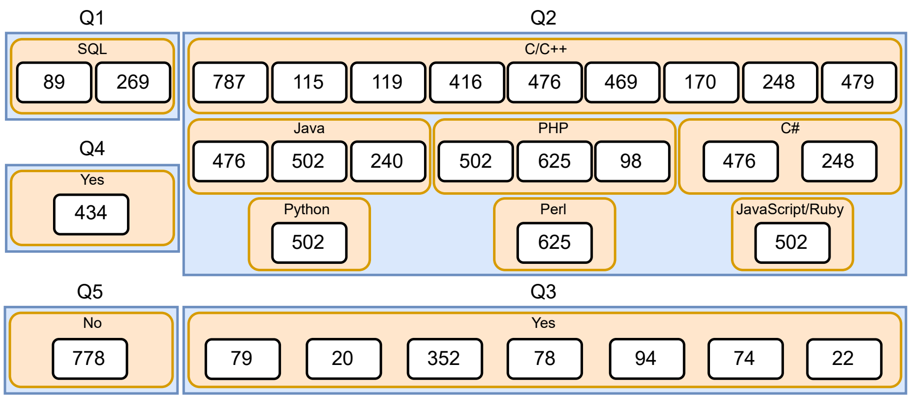


Figure 3.4: Summary of the correlation between the questions gathered from the SBPG module and weaknesses. Blue rectangles represent the questions, orange rectangles depict the answer given to that question, and the white rectangles denote the CWE identifier.

Structured Query Language (SQL), No SQL, Local Storage, and Distributed Storage are the possible answers to the first question. Meanwhile, the user may select C/C++, Java, Python, Hypertext Preprocessor (PHP), Perl, JavaScript/Ruby, and C♯ for the programming languages. Like the previously presented module, the remaining questions only have the `Yes` or `No` options.

When the user selects SQL for the type of data storage used (**Q1**), it must be advised about the plausibility of the attacker performing SQL injections in the queries and the incorrect assignment of user permissions. C/C++ languages are more versatile, containing a significant amount of weaknesses due to mishandling of memory, read and write procedures, and mathematical operations. The *NULL Pointer Dereference* and *Uncaught Exception* are weaknesses that affect both Java and C♯. *Deserialization of Untrusted Data* is a weakness found among multiple languages, without reference to its existence in C/C++, C♯, and Perl languages. Additionally, it is essential to consider restricting user-inputted information using a regular expression on PHP and Perl. The previously mentioned weaknesses were explicitly related to a language because CWE supplies information about it (**Q2**).

Allowing the user to input data (**Q3**) into the application is the source for multiple weaknesses to emerge and should only be applied when strictly necessary. For example, improper neutralization, validation, and limitation are weaknesses that arise from implementing this functionality in the system. Furthermore, it is crucial to define the possible file extensions to upload into the system when allowing the file upload possibility (**Q4**). Finally, if the developers do not implement the registration of operations made by users (**Q5**), they are advised to do it following guidelines.

Along the work, it was concluded that the previously presented questions did not cover the whole scope of the weaknesses considered as most relevant. Therefore, additional questions were included to help identify other typical bugs that can be found in software and hardware components.

Initially, some hardware flaws were grouped based on the language used due to the existence of an associated HDL. Moreover, the two distinguished languages commonly appeared in all these weaknesses. Therefore, they were grouped in one larger category that asks the developer the following question: "Will the system use a Hardware Description Language?" (**Q1**).

Then, the weaknesses that are considered language-independent were grouped based on common keywords. Following this procedure, it was possible to identify which questions should be asked the user. However, some of the keywords were not related to the design phase of the system, which is assumed when using this platform. Therefore, a subsection of questions related to the post-design phases was created to tackle this issue, and the user can choose if he/she wants to see them. This iterative procedure to devise questions is important for this work, posing as a secondary contribution.

Concerning the design phase, the following keywords were identified: 1) cryptographic algorithms; 2) privilege escalation; 3) Extensible Markup Language (XML); 4) standardized error handling; 5) documentation; 6) external hardware units; and 7) external events. These keywords were reformulated as questions, namely to the following ones:

- **Q2** - "Will there be any cryptographic algorithms?";
- **Q3** - "Will the system have functionalities that need privilege escalation?";
- **Q4** - "Will the system use XML to store or transport data?";
- **Q5** - "Will the platform have a standardized error handling mechanism?";
- **Q6** - "Are all functionalities correctly documented?";
- **Q7** - "Does the system depend on additional hardware units?";
- **Q8** - "Is the system properly isolated/prepared to handle external events?";
- **Q9** - "Will the system be outsourced to Outsourced Semiconductor Assembly and Test (OSAT) entities?".

Figure 3.5 displays an overview of the relation between the answers given to the previously described questions and the CWE weakness identifier.

For the cryptographic algorithms (**Q2**), the most common mistakes are the usage of broken and risky algorithms and insufficiently random values when the algorithm depends on

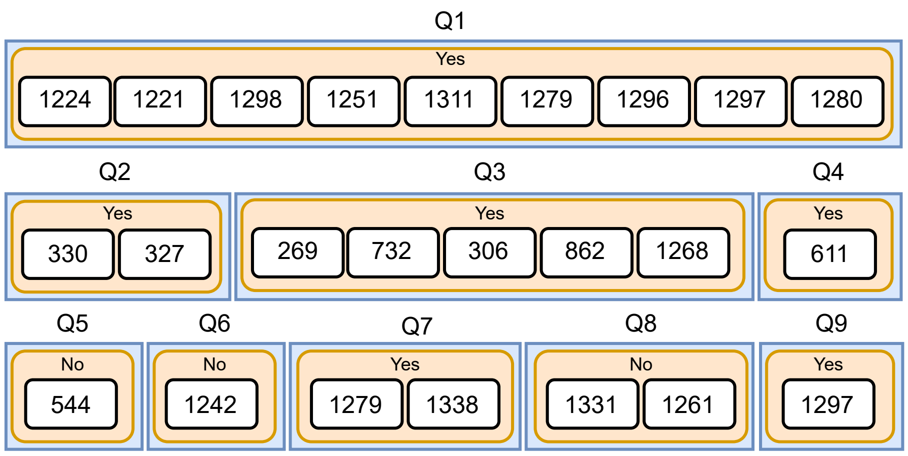


Figure 3.5: Summary of the correlation between the questions for the design phase and weaknesses. Blue rectangles represent the questions, orange rectangles depict the answer given to that question, and the white rectangles denote the CWE identifier.

the unpredictability of these values. Furthermore, as it is known, functionalities requiring privilege escalation (**Q3**) present an easy way for attackers to gain higher privileges, so it is crucial to secure these functionalities. Therefore, incorrect permission assignment, lack of authentication, and inconsistent permissions are some of the weaknesses found while implementing privilege escalation.

The usage of XML to store or transport data (**Q4**) represents an entry point for the attackers since it is possible to misuse the XML reading functionality to access unintended places of the system. Additionally, it is essential to have a standardized error handling mechanism (**Q5**) so that unexpected behavior does not unveil any information about the system to the attacker.

Documenting all the available functionalities (**Q6**) is important to help system testing and maintenance in assuring its security. Usually, undocumented functionalities end up being an entry point for the attacker because they were not documented and, consequently, not tested. In addition, if the system depends on external hardware units (**Q7**), it is necessary to ensure that these units are not overheating and interact accurately with the system.

If the system is not adequately isolated (**Q8**), two significant weaknesses arise 1) information leakage from the communication between the hardware components; and 2) it is possible to introduce errors in the circuit. Finally, if the system is going to be outsourced to OSAT entities (**Q9**), sensitive information theft is possible, and the company must apply the required security mechanisms before outsourcing their products.

Concerning the post-design phases, the user is questioned about the testing specifications of the system, code refactoring and maintenance, compliance with third-party software stipulations, protection of sensitive code, and debug functionalities. Therefore, the following questions originated:

- **Q1** - “Will the system be tested to evaluate the program behavior?”;

- **Q2** - "Will the code be actively reviewed and refactored?";
- **Q3** - "Is assumed-immutable data stored in write-once memory?";
- **Q4** - "Does the code follow the specifications provided by the third-party software to be used?";
- **Q5** - "Is there any protection mechanism assigned to sensitive code (e.g., boot code)?";
- **Q6** - "Will unnecessary debug functionalities be removed from production devices?".

Figure 3.6 displays an overview of the relation between the answers given to the previously described questions and the CWE weakness identifier.

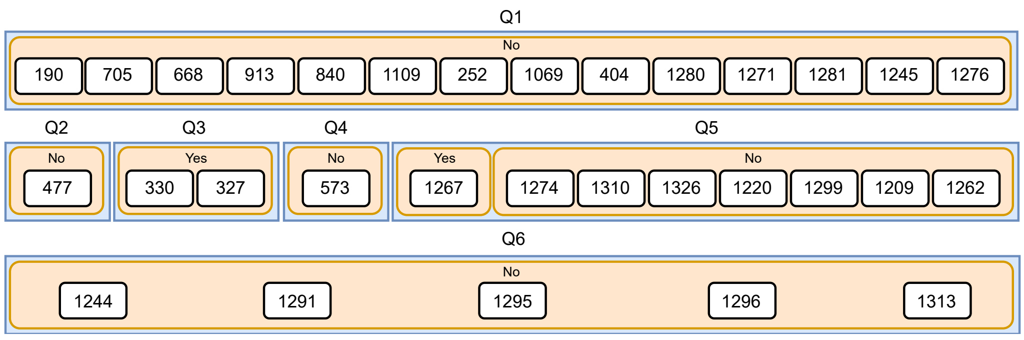


Figure 3.6: Summary of the correlation between the questions for post-design phases and weaknesses. Blue rectangles represent the questions, orange rectangles depict the answer given to that question, and the white rectangles denote the CWE identifier.

If the developed system is not tested to evaluate if it behaves as expected (**Q1**), various weaknesses may serve as the entry point for the attacker. Some common flaws such as unchecked return values, usage of the same variables for different purposes, improper resource shutdown were identified. Thus, testing is a critical phase before the system is released to production and allows developers to fix unexpected behavior.

Code review and refactor (**Q2**) are necessary to assure that no obsolete code is found on the production phase, serving as an entry point for the attacker. Additionally, if any piece of code is assumed to be immutable (**Q3**), it should be written in write-once memory so that attackers can not erase this information. Furthermore, to avoid any attack coming from an external party (**Q4**), the developers should rigorously obey the documentation provided by the third-party software.

Sensitive code (**Q5**) such as certificates, boot, and security settings should have an additional security mechanism associated, reducing the risk correlated with the attacker obtaining sensitive information about the system. Finally, to ease the maintenance of the code during production, developers might leave debug functionalities (**Q6**) on the devices, representing an entry point for the attackers to exploit among multiple systems fabricated by the same enterprise.

## 3.5 Conclusion

With the identification of functional and non-functional requirements and the comprehension of the interaction between SAM and the proposed module, the module implementation and integration can be performed. Additionally, the displayed set of questions and weaknesses and their association provide the suitable logic behind the proposed module reasoning. The identified threats resulted in the establishment of 15 questions to be asked in this module and the selection of 16 questions asked in other modules. These are the foundations that support the development of the proposed module.

# Chapter 4

# Module and Framework Implementations

## 4.1 Introduction

One objective of the proposed module was its integration into an existing security framework. Following this approach, the implementation details were divided into two sections: module incorporation and platform enhancement. Therefore, the first section (section 4.2) details the functions and procedures adopted to integrate the proposed model into the SAM environment – An API provided by SAM was employed to ease the implementation and deployment of models and plugins. Finally, a set of improvements done in the back-end and front-end of the security platform is displayed in section 4.3.

## 4.2 Module Incorporation

When creating a new module in SAM, it is required to upload a Python file that contains the logic of that module, comprising the mapping between the answers and the recommendations. Additionally, the existing platform was designed to have modularity in mind, facilitating the integration of new modules. Therefore, it provides a sample code that can be used as a guide for developing our code. This example contains one function, called `run`, which receives the questions and answers given in one session and the recommendations stored in the database with the respective description and unique identifier. Moreover, functions and global variables can be implemented as long as the `run` function is considered the entry point of the file. Finally, the file associated with a module is called after a session for that module has finished.

Since the data exchanging between SAM and the modules is done using a JavaScript Object Notation (JSON) file, additional functions are required to be implemented as per API, namely `get_answer_content`, `get_recommendation_id`, and `get_module_answers`, which can be used in future modules. The headers for these functions can be seen in code snippet 4.1.

The first one is responsible for retrieving a list containing the text of the answers, given the question identifier and the current session information. The following function returns the recommendation database identifier, given the recommendations JSON and the recommendation text. Finally, the last function is essential to obtain the answers given in a previous session for the dependencies of our module. This function receives the database identifier for the dependency module and the dictionary containing the data for the current session, returning the set of answers given to that module.

**Listing 4.1** Headers of the functions that can be used throughout different modules to ease modification and maintenance.

```
...
def get_answer_content(questions, question_id):
...
def get_recommendation_id(recommendations, recommendation_name):
...
def get_module_answers(module_id, session):
...
```

Finally, considering that SAM stores the recommendations in the database with identifier, name, and description, a dictionary that contained as key the CWE identifier and as value the recommendation name was implemented. Therefore, allowing the obtainment of the recommendation name needed to use in the `get_recommendation_id` function while only providing the CWE identifier. Three additional functions were implemented, namely `add_recommendation`, `order_by_top_25`, and `retrieve_vulnerabilities`, whose headers are displayed in code snippet 4.2.

**Listing 4.2** Headers of the functions specifically designed to assist in the proposed module maintenance.

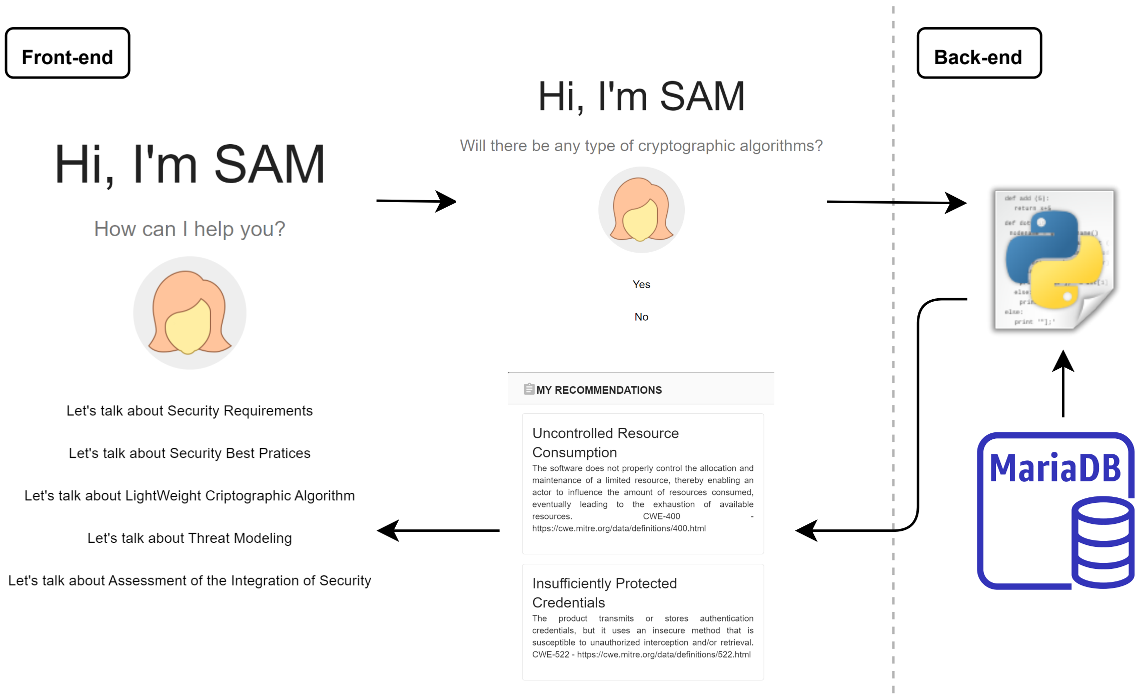
```
...
def add_recommendation(recm_list, recm_abbv, recommendations):
...
def order_by_top_25(recm_list):
...
def retrieve_vulnerabilities(SRE_answers, SBP_answers, TMS_answers, recommendations):
...
```

The first function adds a recommendation identifier to a list of recommendation identifiers, defined in the database. This function receives the previously mentioned list, the recommendation CWE identifier, and the recommendations JSON provided by SAM. The `order_by_top_25` function is responsible for prioritizing the weaknesses presented to the user, using the formulas discussed in subsection 3.4.1. Finally, the last function implements the logic described through subsection 3.4.2, returning an ordered list of recommendation identifiers to be sent back to SAM.

**Listing 4.3** Example of the mapping between the CWE identifiers and the question "Will there be any cryptographic algorithms?".

```
...
    if crypto_algo.lower() == "yes":
        add_recommendation(recms, 'CWE-330', recommendations)
        add_recommendation(recms, 'CWE-327', recommendations)
...
```

Code snippet 4.3 displays the process of mapping weaknesses to the question "Will there be any cryptographic algorithms?", which is also done with condition statements for the remaining questions. After implementing the logic file, it can be uploaded through the administration interface, and the proposed module is ready to be handled by end-users using the SAM front-end. Figure 4.1 displays a brief demonstration of the platform workflow when interacting with the proposed module. Initially, the end-user selects which security topic to discuss regarding an IoT system. Then, it answers the questions presented by the selected module. Finally, it receives the recommendations for that security topic.



Front-end
Hi, I'm SAM
How can I help you?
Let's talk about Security Requirements
Let's talk about Security Best Pratices
Let's talk about LightWeight Criptographic Algorithm
Let's talk about Threat Modeling
Let's talk about Assessment of the Integration of Security
Hi, I'm SAM
Will there be any type of cryptographic algorithms?
Yes
No
MY RECOMMENDATIONS
Back-end
MariaDB

Uncontrolled Resource Consumption

The software does not properly control the allocation and maintenance of a limited resource, thereby enabling an actor to influence the amount of resources consumed, eventually leading to the exhaustion of available resources. CWE-400 - https://cwe.mitre.org/data/definitions/400.html

Insufficiently Protected Credentials

The product transmits or stores authentication credentials, but it uses an insecure method that is susceptible to unauthorized interception and/or retrieval. CWE-522 - https://cwe.mitre.org/data/definitions/522.html

Figure 4.1: Demonstration of the SAM platform workflow when selecting a module, answering questions, and viewing the outputs and their correlation with the back-end.

## 4.3 Framework Enhancement

SAM is a Web application that allows developers to obtain recommendations about their under-development systems (thought it can also be used for suggesting improvements to already deployed systems). It is divided into front-end and back-end, implemented in React and Python, respectively. Since the proposed module is included in this framework (currently in *alpha* version), some contributions to the underlying logic were also performed, implementing new functionalities and improving already developed ones. The following sections describe these contributions.

### 4.3.1 Back-end Improvements Overview

The adaptation of previously developed modules (as shown in figure 3.1), namely SRE, SBPG, LWCAR, and Cloud Security Requirements Elicitation (CSRE), to work under the framework specifications were part of the work performed in the scope of this project. First, the implementation of the logic file for each of the previous modules and the addition of the questions, answers, and recommendations to the database was performed. Furthermore, it was discovered that the SBPG module needed support for multiple answers in the same multiple-choice question, leading to changes in the way SAM was processing questions. Finally, a bug was found on the file saving function where the file was not being saved with the correct file name, which was solved by adding a standardized naming scheme.

While developing the proposed module, the possibility of components that do not have any questions associated and acquire information based on answers from other modules emerged. At that time, SAM did not allow the creation of modules that did not have any questions, so this functionality was introduced on the platform. To implement this new feature, adding a service that returns if the module corresponding to the given database identifier is a *plugin* was the first step. Then, the validation procedures had to be changed to support the creation of modules without questions, yet demanding dependencies.

Also, when trying to integrate the proposed module into SAM, an issue with the deletion of logic files was discovered, as they were not being deleted. This issue required changes to the back-end, where a function containing the file deletion process was added. This function has validation steps for the HyperText Transfer Protocol (HTTP) request type, the user authenticity, and the file existence. Moreover, the recommendations were not displayed in the order reflected by the database. This issue was related to one of the views that were created, which had no order criteria, thus being ordered by the Database Management System (DBMS). It was solved by adding the recommendation database identifier as an order criterion to the view.

With the increase of sessions done by a unique user, it was observed that there was a time increase for creating a new session for the same user. This issue was happening due to two verification mechanisms: 1) check if all previous sessions for this module were closed; 2) check for sessions corresponding to the dependency modules. The first process verified for each existing session if it was closed, which was far from optimum. However, if each session ensures that the previous session is closed, it guarantees that all the sessions will be closed recursively, which was the optimization done for the first validation mechanism. The latter process was retrieving all session data from the database for a module and user to count the number of sessions. However, since SQL has a function that counts the number of rows in the query, it was possible to optimize this validation mechanism. Thus, a function that returned the number of sessions found for a module and a user was implemented. These optimizations provided the capability to handle a significant amount of sessions without experiencing idling time.

Additionally, a bug that was found in both database and back-end implementation was related to two modules having the same short and display name (which is the name that is shown to the users). The application must not present two different modules with the same name to the user, so this was considered a bug that needed to be fixed before the application went into the production phase. This issue was solved by adding the `UNIQUE` constraint in the columns referring to these names and implementing validation checks when adding or editing a module.

Finally, to store the user recommendations and the corresponding guides for a session, we implemented its storage in a `.zip` archive using the `shutil` library. This new functionality required using a specific library called `PyFPDF`, allowing the generation of Portable Document Format (PDF) documents in Python. Furthermore, the `.zip` was designed to be downloaded immediately after finishing the module/plugin questionnaire. Therefore, this functionality is divided into two main steps: 1) PDF production after the recommendation registration in the database and 2) archive creation after the PDF has been generated.

### 4.3.2 Front-end Improvements Overview

Like the back-end, the front-end was also enhanced to provide a better user experience. Several quality-of-life features were also implemented. The list of enhancements is as follows:

- Added the capability of the page to reload when a warning is closed;
- Enable the addition and edition of modules without questions, previously denominated as *plugins*, and creation of a column to specifically identify if a module is a *plugin* or not;
- When creating the module question tree, the hint text presented on the `value` property of the `input` was replaced to its `placeholder` property;
- Attached the automatical closure of the form when adding and editing a module and introduced a *loading* animation to this process;
- Added `Save` buttons throughout multiple forms to increase the user perceptiveness of having saved the previous modifications;
- Limited the upload file type for both module logic and question tree so that the user gets a hint about which type he should upload;
- When using the module question tree for read-only and link operations, it is no longer possible to alter the question and answer content;
- Graphical restriction of the amount of identifiers that the same dependency or recommendation can have while adding or editing a module;
- The possibility to return to the home page, when clicking the platform icon, presented in the top left corner;
- Added a custom error message when the user inserts an already existing short or display name;
- Added automatic download of the `.zip` archive when displaying the recommendations, after concluding module/plugin questionnaire and when viewing previous session recommendations.

## 4.4 Conclusions

While implementing the proposed module, the main challenges were the adaptation to the logic of the security framework and the ordered presentation of the weaknesses. However, after developing and correctly integrating the proposed module into the security framework, this module is ready to be automatically tested concerning its functionality. Therefore, it should be validated by security specialists and provided to end-users, giving feedback regarding the utility and simplicity of the platform.

# Chapter 5

# Documentation, Demonstration, and Evaluation

## 5.1 Introduction

To evaluate the proposed module, it was necessary to think of and specify a method or methods that could apply to this particular work. This chapter starts with the description of those methods in section 5.2, to then describe a user guide and possible application scenarios in section 5.3. These scenarios were part of the selected means to evaluate the tool. At the end, the overall feedback from end-users and security specialists is presented in section 5.4.

## 5.2 Methodology

The testing methodology is divided into three distinct phases: 1) test the proposed module behavior, 2) have the module validated by security specialists, and 3) retrieve end-user feedback (not necessarily specialists) from the adequacy and clarity of the module output.

First, to ensure that the module behaves as expected, automatic tests that this module should pass were developed. Selenium [Sof21], a framework to test Web applications, was used to perform these tests. Based on the selected questions and Selenium inputted answers, a set of expected weaknesses the module should provide to the user was created and configured in the framework.

As presented in subsection 3.4.2, the data is acquired from questions that come from already existing modules or from specifically created ones. Thus, behavior testing was further divided into three different acts:

1. Test if the expected weaknesses were produced for the specific answers withdrawn from the SRE, based on the mapping shown in figure 3.3;
2. Perform the same type of test for the SBPG module, based on the arrangement presented in figure 3.4;
3. Test each possible answer for the questions proposed for the developed module, based on the mapping shown in figures 3.5 and 3.6.

This testing process ensures that the proposed module is correctly outputting the intended weaknesses to the developer.

The developed module was then shown to security specialists who gave their feedback about the presented weaknesses adequacy and efficiency. This validation process was done by directly interacting with the security specialists, which provided relevant suggestions and comments regarding the developed module. Additionally, the security specialists completed a questionnaire provided to end-users, giving their feedback about the module.

The last phase started with the planning of a set of IoT scenarios for the end-users, whose main purpose was to simplify their interaction with the proposed module and guide them in the testing task. Additionally, a user guide that contains explanations about the questions presented and how they could use the security platform to view outputted recommendations was provided. Furthermore, a survey to retrieve end-user feedback was designed, which acquires data regarding the simplicity, adequacy, and efficiency of the weaknesses, end-user knowledge, and previous experience with a similar tool, resulting in six questions. Finally, the proposed module was ready to be tested by end-users, which was the final phase of the testing plan. At this stage, the proposed module was provided to end-users using the hosting security platform SAM (accessible anywhere from the Internet).

## 5.3 User Guide and IoT Scenarios

To ease the end-user testing phase, a user guide that describes potential interactions and provides a brief description of the questions in the module was designed during this part of the project. This user guide is presented in subsection 5.3.1. Additionally, a brief description and a high-level scheme for seven IoT scenarios, also provided to the user, is shown in subsection 5.3.2.

### 5.3.1 User Guide

The user guide is divided into two major components: (i) the explanation about questions (which provides a brief description of each question) and (ii), platform interaction (which explains how the user can interact with the platform to retrieve the recommendations from the proposed module). Notice that the text below is a well-formatted transcription (delimited with lines) of the user guide developed in the scope of this work.

---

**Question Explanation**

The question "*Will the system use a Hardware Description Language?*" indicates if the system functionalities will be implemented using Hardware Description Language. Despite the existence of multiple Hardware Description Languages, they have similar flaws. Thus, if the system uses at least one Hardware Description Language, the accurate answer is *Yes*.

The question "*Will there be any cryptographic algorithms?*" refers to the usage of algorithms that provide data encryption, authentication, integrity, and digital signatures. These algorithms are commonly applied in the communication between two devices, particularly if it is over the Internet, in access control to the device, especially in user registration and login, and in granting data cohesion among two devices or within one device, specifically to ensure the data is not altered intentionally.

The question "*Will the system have functionalities that need privilege escalation?*" concerns the possibility of needing root/administrator privileges to perform a specific action in the system. For example, when considering an insulin pump, the insulin injection functionality needs extra privileges since it interacts with the person healthcare.

The question "*Will the system use XML to store or transport data?*" regards the usage of XML in any system operation. This file format is usually adopted to store information, just like a database, since it has tags that outline its stored data. Additionally, it can also be used in data exchange between two devices for the reasons previously presented.

The question "*Will the platform have a standardized error handling mechanism?*" refers to using a customized error mechanism for multiple errors that might appear in the system. One typical example is using the same mechanism for different HTTP response codes in a Web application, essentially without disclosing any information about the system.

The question "*Are all functionalities correctly documented?*" indicates that all the system functionalities must be fully and rigidly documented. The developers must not leave any functionality incorrectly documented because this may lead to inadequate interaction with other devices. This deficiency might reveal flaws in the interoperability with third-party devices.

The question "*Does the system depend on additional hardware units?*" refers to the possibility of the system having additional hardware units attached to its motherboard. Usually, additional hardware units, called daughterboards, are developed and attached to the motherboard to support hardware upgrades. For example, it is frequent to use an additional hardware unit that performs cryptographic operations.

The question "*Is the system properly isolated/prepared to handle external events?*" regards the possibility of random external events modifying system or communication data and interfere with the system operability. Additionally, the attacker intentionally caused some external effects, so the capability of the system to handle these situations is required.

The question "*Will the system be outsourced to OSAT entities?*" considers the possibility of outsourcing the assembly and testing of the system to an external entity. After the design phase, information about the system schemes, functional and other requirements, technologies applied are sent to an external entity responsible for assembling the final product.

The question "*Will the system be tested to evaluate the program behavior?*" intends to evaluate if the system will be properly tested. This question demands that the user is sincere. If at least one of the implemented functionalities is not correctly examined, it might cause the whole system to fail.

The question "*Will the code be actively reviewed and refactored?*" refers to revising and refactoring the code after the preliminary versions of the system or throughout all versions. Additionally, it considers if the code will be reviewed and refactored, accordingly, before and during the production phase. It is crucial to ensure that the code has no flaws and remains in that state throughout the production stage.

The question "*Is assumed-immutable data stored in write-once memory?*" concerns the storage of data that remains constant throughout the whole device life into memory that can only be modified once. For example, data regarding the cryptographic keys of the system should not be changed throughout the device lifetime.

The question "*Does the code follow the specifications provided by the third-party software to be used?*" considers the possibility of not rigorously following the specifications for third-party software. Assuming that two different software use equivalent specifications because the same company develops them is a possible mistake.

The question "*Is there any protection mechanism assigned to sensitive code (e.g., boot code)?*" concerns the usage of defensive mechanisms that protect sensitive data found on the system – for example, using cryptographic algorithms to assure confidentiality of the system boot code or trusted entities.

The question "*Will unnecessary debug functionalities be removed from production devices?*" intends to ensure that the developers have removed all the unnecessary debug functionalities, such as providing information about the system operations, using a default password for different devices, leaving passwords storage in plaintext. This question requires that the user is as sincere as possible.

**Platform Interaction**

This platform contains user registration, meaning that each user can get their separate recommendations without conflicting with other possible users. Thus, the user must register using a valid email to start interacting with the platform. Then, it must authenticate itself with the system using the registered email and password.

At this point, the user is presented with a set of security topics the platform is ready to inform. All modules, except ACISM, display to the user a set of questions he must answer before getting the intended recommendations. Additionally, the user can view previous sessions and respective recommendations, change its account information, and sign out through the upper bar.

Recommendations from the developed module are provided after answering the questions under the "*Let's talk about Threat Modeling*" option. However, the user must previously answer the questions asked on the "*Let's talk about Security Requirements*" and "*Let's talk about Security Best Practices*" to have access to the developed module. This requirement results from the developed module acquiring data from the answers given to the two previously presented topics.

After finishing the proposed module questionnaire, weaknesses with the respective CWE identifier and description are shown. Additionally, each weakness references the source of the information that shows additional information about it, such as related weaknesses, system impact, and the extended description. This information can be downloaded in PDF format and is also registered in the previous user sessions section.

---

### 5.3.2 IoT Scenarios

It is essential to provide end-users with IoT scenarios to help them explore the platform and the proposed module. These scenarios, in combination with the developed questionnaire, are intended to help users evaluate the platform. Based on a security expert view, the IoT cases were divided into seven main categories:

- Smart Transportation and Logistic;
- Smart Grid;
- Smart Environment;
- Smart Healthcare;
- Smart Agriculture and Environmental Monitoring;
- Smart Wearable;
- Smart Manufacturing.

Smart Transportation and Logistics comprise all the IoT scenarios whose primary purpose is to provide a better traffic flow and on-demand supplies. This process is done by placing multiple sensors at strategic places, namely at crossroads, traffic lights, and using identifier tags on the products being delivered to the consumer [ZKKK19, JFK$^+$19]. The incorporation of sensors and communication networks into the electric system is called Smart Grid. The main objective is to provide a more flexible and responsive electric system, reducing costs and improving efficiency [Eh14].

Smart Environment is a category that encompasses intelligent components which represent smaller environments, such as Smart Cities and Smart Cars. These environments can supervise themselves automatically by scheduling preventive maintenance and monitoring security aspects while maximizing the services provided to people [CNW$^+$12]. Smart Healthcare comprises the complete set of technologies designed to provide customized healthcare or monitor person healthiness. These systems can increase efficiency when acting in emergencies, provide better access to healthcare in rural areas, and enable elderly independence [BXA17]. Smart Agriculture and Environmental Monitoring are the set of technologies designed to provide healthier food and atmosphere. These methodologies also consider the harm done to climate and benefits to human and animal health [GK+16, IEBM15].

Smart Wearable comprises a set of clothes or accessories that a person can wear to acquire data about its current status. The most commonly used smart wearable is a smartwatch, which acquires data about the person healthiness and provides a set of services that ease communication with others. The difference between this category and Smart Healthcare resides in the fact that no communication is done with healthcare entities, and it is designed for the person to gather data about itself [CEF+12]. Finally, Smart Manufacturing, also known as Industry 4.0, is a technology that aims at sustainable growth management and improvement of existent production agents. Its main objective is to respond to complex and varied situations in real-time while maintaining reduced assets in warehouses [KLC+16].

Below, an IoT scenario for each of the above-mentioned classes is presented. The proposed situations were initially designed by the authors and might not exactly correspond to actual scenarios. These scenarios were also provided to testers, and formatted to fit the template of this dissertation.

---

### Smart Transportation Monitoring

The Smart Transportation Monitoring scenario is included in the Smart Transportation and Logistics category. Figure 5.1 shows a high-level representation of the described scenario, displaying two main components: the *Truck* and the *Container*. The first one contains sensors that retrieve data about its location, velocity, and status, which is transmitted to the Cloud. Since the Truck is always in movement, it is essential to ensure that it can communicate from different locations, so Fourth Generation (4G) or Fifth Generation (5G) network technology is used. The Container has multiple sensors for the cargo status, such as temperature, humidity, and position. Additionally, this data needs to be sent to the Cloud while the Container is in movement, so it is equipped with 4G or 5G network technology.

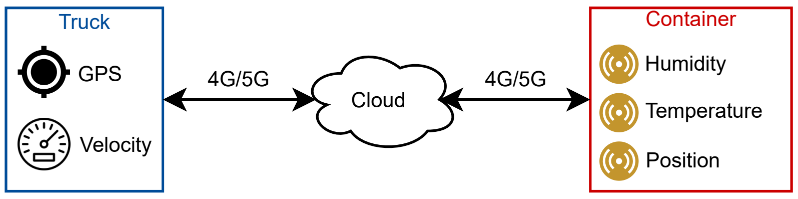


Figure 5.1: High-level design of the Smart Transportation Monitoring scenario.

Sensitive information is transmitted from the Container and Truck to the Cloud through the Internet, where a third entity can eavesdrop on the messages. Therefore, ensuring secure communication through the implementation of cryptographic algorithms is essential. Additionally, since the Truck and Container drive themselves on public roads, the probability of a third entity having physical access to the components is high. The Container will be temporarily associated with the Truck transporting it, and the Truck is registered within the company database. Hence, there is no user registration nor login within the system. Finally, it is possible to order the Truck to change the route if an unexpected event occurs. This command can have significant implications on the arrival time and cargo conditions, so it demands that the person requiring it is properly authenticated and has the correct authorization to do so.

### Smart Electrical Vehicle Home Charging

The Smart Electric Vehicle Home Charging is specifically designed to improve the efficiency of the electric grid while maintaining sustainability. It is divided into three main areas: *Residential Area*, *Distribution*, and *Generation*, as shown in figure 5.2. All these areas are connected with electric cables, meaning that the communications between them are done with cables to keep a low latency response. In addition, every end component that receives energy has a smart meter that allows reading and modifying the electric system. Furthermore, it is possible to observe that each house is equipped with an electric vehicle charger, which functions as a battery for the house demands. Regarding the Distribution area, each electric tower contains a smart switch connected to a substation, where all processing is done. Finally, the Generation section provides data about electricity generation from multiple sources, such as solar, wind, or fossil fuels.

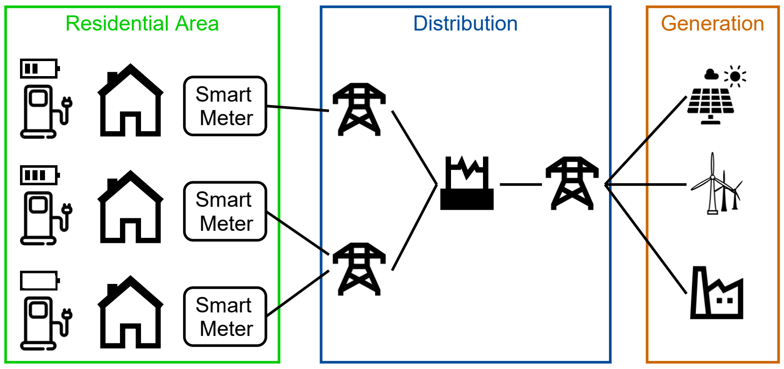


Figure 5.2: High-level scheme of the Smart Electrical Vehicle Home Charging scenario.

Smart meters are placed in an accessible location for maintenance purposes, making them available for third entities besides the product company. These meters are also responsible for measuring the power consumption of the house. Therefore, some users might try and impersonate others, avoiding paying taxes. Additionally, since communication is done through cables and not wireless, it is harder for a third entity to intersect and modify messages. Data about the house and its owner is considered sensitive content since it might reveal residence location or workplace, leading to privacy leakage. This system will store temporary information about the power consumption and outage in a database and send its information periodically to the power company, which stores it in their servers. Additionally, electric vehicle chargers can be used as electric grid power supplies during power shortages on the grid. The decision to switch electric vehicle chargers operating mode must be made by an authenticated person with proper authorization.

### Smart Car

The Smart Car is considered a Smart Environment because it can operate independently of other cars or contexts. Intelligent cars have multiple sensors to retrieve data from the car and the environment surrounding it. Some examples of the sensors and environment variables can be seen in figure 5.3. The main idea behind implementing smart cars and intelligent cities resides in improving efficiency and sustainability. This growth can be done by conjugating multiple data in Cloud and convert them into valuable information. Accordingly, this is the main reason why 4G and 5G network technologies are implemented into the *Car*, to provide local-independent data communication. Additionally, to better optimize this process, additional external factors to use the same technologies were considered, providing real-time data about traffic.

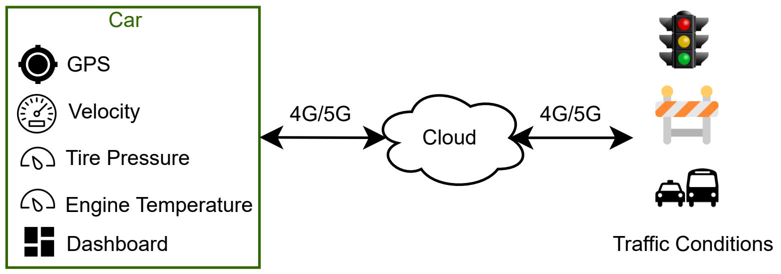


Figure 5.3: High-level design of the Smart Car scenario.

Like trucks, cars spend most of their life span in public spaces, meaning that third entities can easily access them. The data exchange between the Car and Cloud must be done reliably and without the possibility of being manipulated, suggesting the usage of cryptographic algorithms. For example, a malicious user might try and impersonate another user to obfuscate their current location and expected course. Besides that, revealing the expected route might be dangerous since it might reveal private information about the user. If for any reason, it is imperative to change the Car route, the person doing it must be correctly authenticated and have the necessary permissions.

### Smart Pacemaker

A Smart *Pacemaker* is a standard technology implemented into a patient, allowing a customized healthcare service. As shown in a report published by The Guardian [Ale13], some patients feared the usage of their pacemaker because it could be hacked and lead to tragedies. Because this intelligent system interacts directly with human lives, it must be correctly secured and tested. Figure 5.4 exhibits the entities involved in the Smart Pacemaker system, as well as their connections. Communication between this device and the *Hospital* might be essential in hostile conditions, so 4G or 5G network technologies were chosen. Additionally, the Hospital stores its data and information in a private Cloud, communicating with Wi-Fi technology.

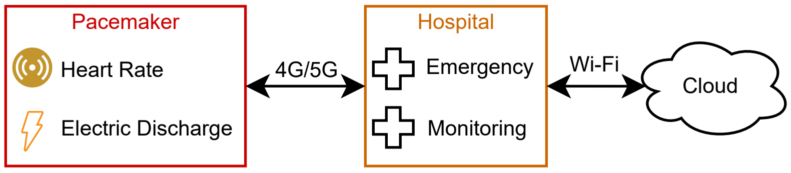


Figure 5.4: High-level scheme of the Smart Pacemaker scenario.

Maintaining communication between the Hospital and Smart Pacemaker is vital since every second counts when saving a life. Furthermore, it is necessary to guarantee that no fake emergencies are notified, which could deviate resources from real emergencies. These fake emergencies can be done by capturing messages and re-sending them to the Hospital. In this specific case, it is impossible to get physical access to the Smart Pacemaker since it involves surgery and assuring that the patient survives. Despite the information not being directly stored in the Smart Pacemaker, using a database, the type of data exchanged with the Hospital is critical. Besides, exposing data about the patient heart rate might reveal which activities the patient is performing, leading to privacy leakage. Finally, based on the medical record and the current heart rate, if it is vital to change electric discharge power, it must be done by an authenticated and authorized user.

### Smart Irrigation System

Optimizing water usage, consequently increasing sustainability, can be done through a Smart Irrigation System, like the one depicted in figure 5.5. This system is equipped with sensors that try to acquire the most accurate data about the *Crops*, such as temperature, humidity, and soil moisture. This scenario uses 4G or 5G network technology to communicate with the Cloud, ensuring reliable communication and avoiding the burden of constructing a Wi-Fi or cable infrastructure. Additionally, data about the *Weather Conditions* in the next few days is also provided to the Cloud using Wi-Fi technology. Thus, processing and joining it with sensory data provides the most informed decision to irrigate or not the Crops.

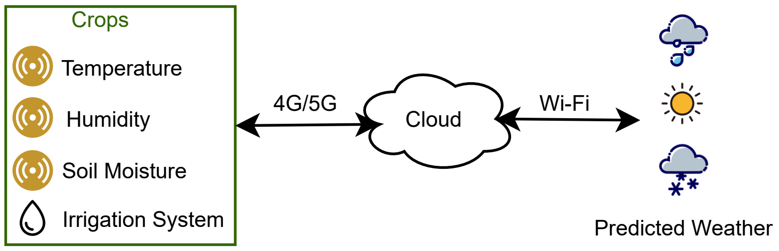


Figure 5.5: High-level design of the Smart Irrigation System scenario.

These irrigation systems are implanted in agricultural land, sometimes in remote locations or with difficult access, which are not frequently revisited. The possibility of malicious users physically accessing the system is significant and must be considered. Additionally, this system is designed to be entirely automatic, but its owner can alter its configurations. Thus, a unique user exists in this system, which allows viewing Crops data, but requires additional approval to change the system configuration. Data is sent through the Internet, being plausible to be eavesdropped on, making it essential to use cryptographic algorithms. Finally, a malicious user can capture messages and resend them to simulate a genuine request, making it possible for this user to destroy the Crops.

### Smart Watch

Smart Watches have currently replaced nearly all the ordinary watches in developed countries due to the human need to always being connected to others. These intelligent watches provide essential functionalities offered by the Smartphone and extra ones mainly related to exercising. If the user wants to receive *Smartphone* notifications and alarms, it must connect the *Smart Watch* to its Smartphone using Bluetooth technology, as shown in figure 5.6. This device is not directly connected to the Cloud, receiving updates directly from the Smartphone with the proper application installed.

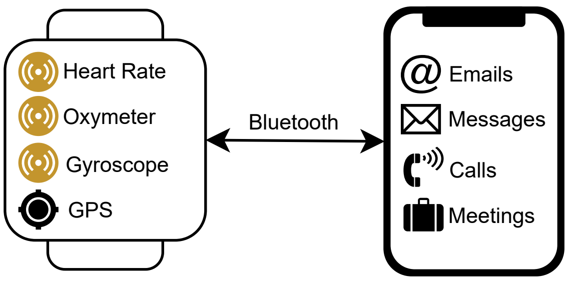


Figure 5.6: High-level scheme of the Smart Watch scenario.

This specific Smart Watch considers the possibility of usage without connecting to a Smartphone, providing only the exercise related functionalities. Therefore, it needs to store details about the user workout in a database included on the device. Additionally, this data is sent to the Smartphone when it connects to the Smart Watch. The loss of the Smart Watch is a plausible situation due to the removal of the watch for a leisure situation. This device connects through Bluetooth to the Smartphone, in which the communications can easily be eavesdropped on and captured. Usually, information exchanged between the Smart Watch, and the Smartphone is not critical nor sensitive. Finally, impersonating a user is possible due to Bluetooth technology actuating distance and the Smart Watch loss.

### Smart Factory

The Smart Factory scenario is considered the Industry 4.0 peak when factory production lines are connected to the market supply chain. Therefore, it is included in the Smart Manufacturing category. A Smart Factory contains sensors and dashboards for each *Machine* on the production line, as exhibited in figure 5.7. These provide data about the product and the production rate, such as temperature, pressure, and pieces per hour. This data is exchanged across the factory floor using Wi-Fi technology and allows the floor manager to decide the Machine operating mode. Additionally, it provides information about failures considering a specific Machine, which is stored in the factory server. Details about factory operation mode and rate are sent to the Cloud and are shared with other supply chain entities, which report *Demand Fluctuations* motivated by multiple factors.

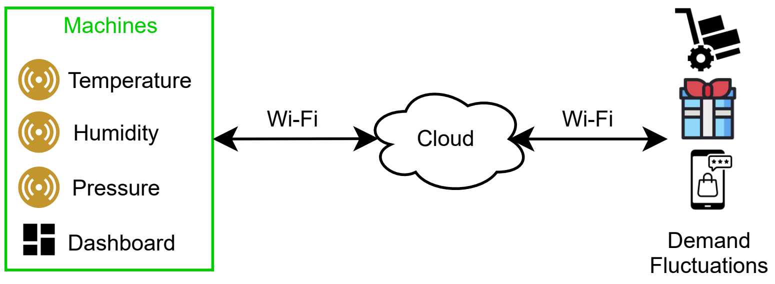


Figure 5.7: High-level design of the Smart Factory scenario.

Assuming only the factory operators have access to the factory floor, it is impossible to have access to the machine where the system operates. Additionally, most communications are exchanged inside the Smart Factory, which means there is a reduced possibility of eavesdropping. Considering that all factory employees are honest, there is no need to raise awareness on the possibility of capturing and re-sending messages. The weakest spot regarding security is the communication with the Cloud and other supply chain entities, which requires cryptographic algorithms. Since the factory stores information on their servers, they will use a database system that must be correctly secured, preventing access from unauthorized people. Finally, the communication with third-party entities must be done following the specifications.

---

## 5.4 Results Discussion

The module behavioral evaluation was done using an automated approach, where the expected results were provided according to the given inputs. As a result, the proposed module never outputted an error message for all the executed tests, as expected, and always gave the intended recommendations (weaknesses) for all the input answers provided.

Regarding the security specialist feedback, they evaluated this tool as very useful and helpful regarding detecting weaknesses found in the system. Additionally, they stated that the proposed module was simple and straightforward. Finally, they added the suggestion of presenting weaknesses in category groups, such as Authentication and Insecure Data, which will be considered in future work.

A survey was designed to gather information about the threat simplicity, adequacy, efficiency, and existence awareness and the hardware threats relevance to retrieve user feedback from developers. Each aspect was evaluated based on a scale from 1 to 7, with 1 being the most negative and 7 the most positive evaluations. The scale choice was based on the human mind judgment capability [Mil56]. Additionally, developers were asked about their previous experience with similar tools and were given the means to express their suggestions and difficulties found while using the proposed module.

Figure 5.8 exhibits the responses received to the previously presented aspects regarding 18 different end-users with a background in Computer Science, which was the only recruitment requirement. Two out of the 18 end-users were security specialists, nine of these users work in enterprise settings, and the remaining ones are students. The users were recruited using online technologies to spread the solution between class colleagues and exhibitions to IT companies. No personal and soft biometrics data were acquired during the questionnaire.

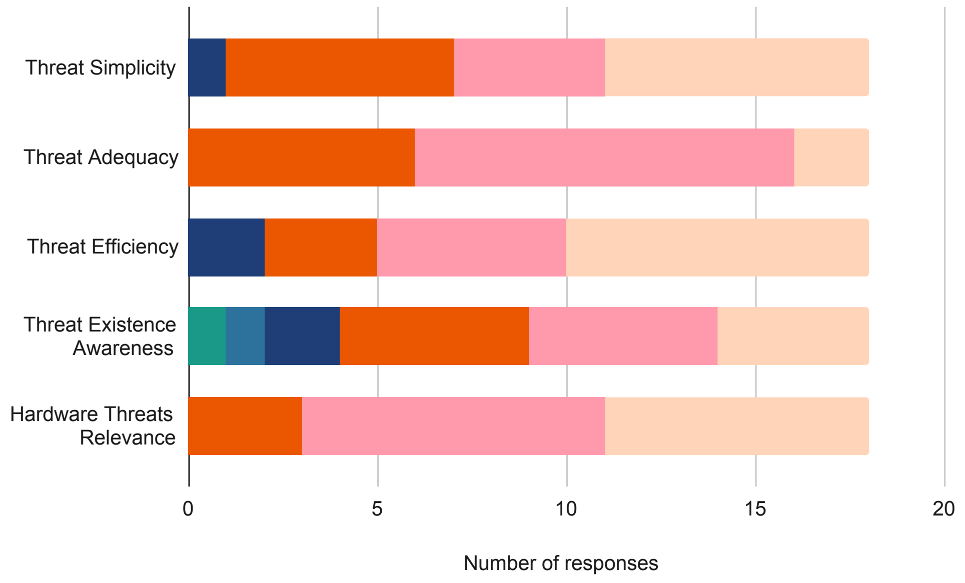


Figure 5.8: Graph summarizing the answers given to the evaluation questionnaire of the proposed module according to the scale from 1 to 7, where 1 is the most negative evaluation, and 7 is the most positive. *Green*, *Light Blue*, and *Dark Blue* represent options 2, 3, and 4, respectively. *Orange*, *Pink*, and *Cream* represent options 5, 6, and 7, respectively.

The Threats Existence Awareness category presents answers ranging from 2 to 7, meaning that developer security knowledge is volatile. This conclusion corroborates the motivation presented that some developers do not have the required knowledge to develop secure systems. Furthermore, by examining the Hardware Threats Relevance, it is possible to deduce that hardware threats are considered a substantial contribution of the proposed module.

Despite the simplicity associated with the threats, there is still a margin to improve them, namely by detailing threat descriptions. Another possible enhancement, suggested by security specialists, is to produce additional questions concerning specific IoT scenarios, increasing Threat Efficiency (the capacity of a threat to identify particular problems). Furthermore, it is possible to conclude that, overall, the developers find the identified threats to be adequate for the system they conceived in their heads.

Regarding the previous experience with similar tools, 16 out of 18 developers said they had never interacted with related tools. The remaining two answers were given by security specialists, affirming that they already had experienced a tool similar to the one proposed in this dissertation. Since the developed module is designed to help developers with limited knowledge in security topics, these results show that this module has a novelty factor, with prominent advantages to enable the planning and implementation of secure devices.

## 5.5 Conclusions

Automatically testing the proposed module to check if it provides the proper recommendations ensures that the developed module is qualified to be validated and approved by security specialists. After this approval, the developed module is ready to be tested by end-users, which provided valuable feedback regarding the novelty, quality, and requirement of the proposed module. Finally, the developed module is duly tested and validated after these steps, indicating it is prepared to work under production settings.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

The evident security problems of IoT, which might specially lead to potential leakage of sensitive information of users, call for a strong move towards the inclusion of security mechanisms in IoT devices. However, this process requires the intervention of security specialists to specifically identify these security requirements, design and implement the appropriate technology, which is expensive for companies. This dissertation provides a solution that enables users with limited security knowledge to create secure IoT systems.

The main contribution of the work described in this dissertation is a solution that outputs the weaknesses found on an IoT system or software that the user describes through questions. The developed tool provides a straightforward way of presenting the weaknesses, and its primary purpose is to support the design and implementation of secure systems without the requirement of having vast security knowledge. Therefore, the objectives established for this dissertation project were accomplished.

During the development of the project leading to this dissertation, the following interesting conclusions were reached:

- There is no clear distinction between IoT domains, and there is no consensus in how IoT layers are divided, regardless of most literature considering the three-layer model. Thus, the deployment of standards is essential to define details regarding IoT, which would later lead to better design of these systems and software;

- Literature alternately refers to *threat modeling* or *attack modeling* when discussing the necessary steps to succeed in an attack. Unfortunately, this usage hinders the comprehension of the difference between these two terms. Therefore, providing a possible distinction for these processes is of utmost importance;

- Prioritization of vulnerabilities and weaknesses is still a manual process, where humans describe each identified threat and specify the values for each group category. This method still requires highly qualified experts and poses sometimes a severe burden when classifying new vulnerabilities or weaknesses. This open issue lead to the development of a CVSS calculator with embedded artificial intelligence, which will be subject of a paper.

## 6.2 Future Work

As future work, some improvements to the developed framework are suggested, mainly regarding information disclosure. The first enhancement considers the clustering of the presented weaknesses in category groups concerning the security requirements. Furthermore, adding an optional extended description to each recommendation might help end-users understanding higher complexity weaknesses. Another possible improvement is to thoroughly investigate IoT scenarios and provide questions that mold to the weaknesses. Furthermore, another supplement is to add an automatic manner that supports the prioritization of vulnerabilities found on the NVD database, providing the vulnerabilities found in the last 30 days. Finally, the addition of an Artificial Intelligence based approach that can easily renew the logic of the proposed module is another possible future path.

Some of the steps of the aforementioned lines of future work were already performed at the time of writing of this dissertation. E.g., an analysis of machine learning algorithms suited for binary multi-output problems was done for the SRE module, resulting in one accepted for publication paper [LCS$^+$21]. Furthermore, this analysis can easily be spread to other platform modules, namely the proposed one. Finally, a prototype tool that communicates with the NVD API and gathers 30-day vulnerabilities is also under development.

# Bibliography

[AA18] R. Alharbi and D. Aspinall. An iot analysis framework: An investigation of iot smart cameras' vulnerabilities. In *proceedings of Living in the Internet of Things: Cybersecurity of the IoT - 2018*, pages 1–10, 2018. 20

[Ale13] Alex Hern. Hacking risk leads to recall of 500,000 pacemakers due to patient death fears. https://www.theguardian.com/technology/2017/aug/31/hacking-risk-recall-pacemakers-patient-death-fears-fda-firmware-update, 2013. Online; accessed 30 June 2021. 48

[Bir05a] Alex Biryukov. *Chosen Ciphertext Attack*, pages 76–76. Springer US, Boston, MA, 2005. Available from: https://doi.org/10.1007/0-387-23483-7_59. 8

[Bir05b] Alex Biryukov. *Chosen Plaintext Attack*, pages 76–76. Springer US, Boston, MA, 2005. Available from: https://doi.org/10.1007/0-387-23483-7_60. 8

[Bir11a] Alex Biryukov. *Adaptive Chosen Ciphertext Attack*, pages 21–21. Springer US, Boston, MA, 2011. Available from: https://doi.org/10.1007/978-1-4419-5906-5_543. 8

[Bir11b] Alex Biryukov. *Adaptive Chosen Plaintext Attack*, pages 21–21. Springer US, Boston, MA, 2011. Available from: https://doi.org/10.1007/978-1-4419-5906-5_545. 8

[Bir11c] Alex Biryukov. *Ciphertext-Only Attack*, pages 207–207. Springer US, Boston, MA, 2011. Available from: https://doi.org/10.1007/978-1-4419-5906-5_560. 8

[Bir11d] Alex Biryukov. *Known Plaintext Attack*, pages 704–705. Springer US, Boston, MA, 2011. Available from: https://doi.org/10.1007/978-1-4419-5906-5_588. 8

[BXA17] Stephanie B. Baker, Wei Xiang, and Ian Atkinson. Internet of things for smart healthcare: Technologies, challenges, and opportunities. *IEEE Access*, 5:26521–26544, 2017. 45

[CBA17] Yassine Chahid, M. Benabdellah, and A. Azizi. Internet of things security. *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, pages 1–6, 2017. 2

[CDRV19] Valentina Casola, Alessandra De Benedictis, Massimiliano Rak, and Umberto Villano. Toward the automation of threat modeling and risk assessment in iot systems. *Internet of Things*, 7:100056, 2019. Available from: http://www.sciencedirect.com/science/article/pii/S2542660519300290. 19

[CEF+12] Marie Chan, Daniel Estève, Jean-Yves Fourniols, Christophe Escriba, and Eric Campo. Smart wearable systems: Current status and future challenges. *Artificial Intelligence in Medicine*, 56(3):137–156, 2012. Available from: https://www.sciencedirect.com/science/article/pii/S0933365712001182. 46

[CNW+12] Hafedh Chourabi, Taewoo Nam, Shawn Walker, J. Ramon Gil-Garcia, Sehl Mellouli, Karine Nahon, Theresa A. Pardo, and Hans Jochen Scholl. Understanding smart cities: An integrative framework. In *proceedings of 2012 45th Hawaii International Conference on System Sciences*, pages 2289–2297, 2012. 45

[Cor18] The MITRE Corporation. CWE - Common Weakness Scoring System (CWSS). https://cwe.mitre.org/cwss/cwss_v1.0.1.html, 2018. [Last accessed in 7th June, 2021]. xv, 18

[Cor20] The MITRE Corporation. CWE - 2020 CWE Top 25 Most Dangerous Software Weaknesses. https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html, 2020. [Last accessed in 7th June, 2021]. 26

[CPB13] Sergio Caltagirone, Andrew Pendergast, and Christopher Betz. The diamond model of intrusion analysis. Technical report, Center For Cyber Intelligence Analysis and Threat Research Hanover Md, 2013. xv, 11, 13

[Dav99] David E. Mann, Steven M. Christey. Towards a Common Enumeration of Vulnerabilities. https://cve.mitre.org/docs/docs-2000/cerias.html, 1999. Online; accessed 17 July 2021. 15

[DOL07] R. Derakhshan, M. E. Orlowska, and X. Li. Rfid data management: Challenges and opportunities. In *proceedings of 2007 IEEE International Conference on RFID*, pages 175–182, 2007. 7

[DSLA17] Piergiuseppe Di Marco, Per Skillermark, Anna Larmo, and Pontus Arvidson. Ericsson white paper: Bluetooth mesh networking. Technical Report 284 23-3310 Uen, Ericsson, 7 2017. Available from: https://www.ericsson.com/49d589/assets/local/reports-papers/white-papers/wp-bluetooth-mesh_ver2_171115-c2.pdf. 6

[Eh14] Mohamed E. El-hawary. The smart grid—state-of-the-art and future trends. *Electric Power Components and Systems*, 42(3-4):239–250, 2014. 45

[eS14] Structured Threat Information eXpression (STIX™). Common Weakness Scoring System — CWSS™. Technical report, International Organization for Standardization, 202 Burlington Road, Bedford, MA 01730-1420, 2014. 18

[Far08] Shahin Farahani. Chapter 1 - zigbee basics. In *ZigBee Wireless Networks and Transceivers*, pages 1 – 24. Newnes, Burlington, 2008. Available from: http://www.sciencedirect.com/science/article/pii/B9780750683937000017. 6

[FPAF18] M. Frustaci, P. Pace, G. Aloi, and G. Fortino. Evaluating critical security issues of the iot world: Present and future challenges. *IEEE Internet of Things Journal*, 5(4):2483–2495, 2018. 5

[fS18] International Organization for Standardization. Information technology — Internet of Things (IoT) — Vocabulary. Standard, International Organization for Standardization, Geneva, CH, December 2018. 5

[GHGK17] Mengmeng Ge, Jin B. Hong, Walter Guttmann, and Dong Seong Kim. A framework for automating security analysis of the internet of things. *Journal of Network and Computer Applications*, 83:12 – 27, 2017. Available from: `http://www.sciencedirect.com/science/article/pii/S1084804517300541`. 20

[GK+16] Nikesh Gondchawar, RS Kawitkar, et al. Iot based smart agriculture. *International Journal of advanced research in Computer and Communication Engineering*, 5(6):838–842, 2016. 46

[HCA11] Eric Hutchins, Michael Cloppert, and Rohan Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1, 01 2011. xv, 14

[IEBM15] Mohannad Ibrahim, Abdelghafor Elgamri, Sharief Babiker, and Ahmed Mohamed. Internet of things based smart environmental monitoring using the raspberry-pi computer. In *proceedings of 2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC)*, pages 159–164, 2015. 46

[JDSTN19] Dan Bergh Johnsson, Daniel Deogun, Daniel Sawano, and Daniel Terhorst-North. *Secure by design*. Manning Publications Co., 2019. 1

[JFK+19] Bilal Jan, Haleem Farman, Murad Khan, Muhammad Talha, and Ikram Ud Din. Designing a smart transportation system: An internet of things and big data approach. *IEEE Wireless Communications*, 26(4):73–79, 2019. 45

[KLC+16] Hyoung Seok Kang, Ju Yeon Lee, SangSu Choi, Hyun Kim, Jun Hee Park, Ji Yeon Son, Bo Hyun Kim, and Sang Do Noh. Smart manufacturing: Past research, present findings, and future directions. *International Journal of Precision Engineering and Manufacturing-Green Technology*, 3:111–128, 2016. Available from: `https://doi.org/10.1007/s40684-016-0015-5`. 46

[KMRS10] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. Foundations of attack–defense trees. In *proceedings of Lecture Notes in Computer Science*, volume 6561, pages 80–95, 09 2010. xv, 10, 12

[KS17] R. Kumar and M. Stoelinga. Quantitative security and safety analysis with attack-fault trees. In *proceedings of 2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, pages 25–32, 2017. 11

[LCS$^+$21] Carolina Lopes, Joana C. Costa, João B. F. Sequeiros, Tiago M. C. Simões, Mário M. Freire, and Pedro R. M. Inácio. Machine learning applied to security requirements elicitation: Learning from experience. In *proceedings of Atas do 12º Simpósio de Informática (INForum 2021)*, pages 0–12, September 2021. (accepted for publication). 3, 56

[Len13] Elizabeth B. Lennon. The national vulnerability database (nvd): Overview. Technical report, National Institute of Standards and Technology, December 2013. Available from: `https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=915172`. 15

[MAH$^+$16] M. Mohsin, Z. Anwar, G. Husari, E. Al-Shaer, and M. A. Rahman. Iotsat: A formal framework for security analysis of the internet of things (iot). In *proceedings of 2016 IEEE Conference on Communications and Network Security (CNS)*, pages 180–188, 2016. 19

[MAL$^+$19] I. Makhdoom, M. Abolhasan, J. Lipman, R. P. Liu, and W. Ni. Anatomy of threats to the internet of things. *IEEE Communications Surveys Tutorials*, 21(2):1636–1675, 2019. 9

[MG02] Peter Mell and Tim Grance. Use of the common vulnerabilities and exposures (cve) vulnerability naming scheme. Technical Report NSN 7540-01-280-5500, National Institute of Standards and Technology, January 2002. Available from: `https://apps.dtic.mil/sti/pdfs/ADA407728.pdf`. 15

[Mil56] George A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63:17, 1956. 51

[MQT20] MQTT.org. MQTT - The Standard for IoT Messaging, 2020. [Last accessed in $4^{th}$ December, 2020]. Available from: `https://mqtt.org/`. 6

[MSR06] Peter Mell, Karen Scarfone, and Sasha Romanosky. Common vulnerability scoring system. *IEEE Security Privacy*, 4(6):85–89, 2006. 16

[Nat21] National Institute of Standards and Technology. NVD - Search and Statistics. `https://nvd.nist.gov/vuln/search`, 2021. Online; accessed 17 July 2021. 15

[oIRT21a] Forum of Incident Response and Security Teams. CVSS v2 Complete Documentation. `https://www.first.org/cvss/v2/guide`, 2021. [Last accessed in 21th May, 2021]. xv, 17 [oIRT21b] Forum of Incident Response and Security Teams. CVSS v3.1 Specification Document. https://www.first.org/cvss/specification-document, 2021. [Last accessed in 21th May, 2021]. xv, 17

[OWA21a] OWASP Foundation. About Us | The OWASP Foundation. https://owasp.org/about/, 2021. Online; accessed 17 July 2021. 16

[OWA21b] OWASP Foundation. Vulnerabilities | OWASP. https://owasp.org/www-community/vulnerabilities/, 2021. Online; accessed 17 July 2021. 16

[PIS$^+$17] P. Patni, K. Iyer, R. Sarode, A. Mali, and A. Nimkar. Man-in-the-middle attack in http/2. In *proceedings of 2017 International Conference on Intelligent Computing and Control (I2C2)*, pages 1–6, 2017. 8

[pt14] Wireless Sensor Networks project team. Internet of things: Wireless sensor networks. Technical report, International Electrotechnical Commission, 11 2014. Available from: https://www.ericsson.com/49d589/assets/local/reports-papers/white-papers/wp-bluetooth-mesh_ver2_171115-c2.pdf. 7

[PTC16] Ed. P. Thubert and R. Cragie. Pv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Paging Dispatch. RFC 8025, RFC Editor, 11 2016. Available from: https://www.rfc-editor.org/rfc/rfc8025.txt. 7

[RKT10] Arpan Roy, Dong Seong Kim, and Kishor S Trivedi. Cyber security analysis using attack countermeasure trees. In *proceedings of Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, pages 1–4, 2010. 11

[Sch99] Bruce Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999. xv, 10, 11

[Sch01] Bruce Schneier. Attack trees. *Dr. Dobb's journal*, 2001. 10

[SLA$^+$20] M. G. Samaila, C. Lopes, E. Aires, J. Sequeiros, T. Simões, M. Freire, and P.R.M. Inácio. A Preliminary Evaluation of the SRE and SBPG Components of the IoT-HarPSecA Framework. In *proceedings of Global Internet of Things Summit, Endorsed by IEEE Global IoT Summit GIoTS*, pages –, June 2020. 23, 24, 27

[SOBG19] Amar Seeam, Ochanya Ogbeh, Xavier Bellekens, and Shivanand Guness. Threat modeling and security issues for the internet of things. In *proceedings of 2019 Conference on Next Generation Computing Applications (NextComp)*, 10 2019. 1

[Sof21] Software Freedom Conservancy. About Selenium. https://www.selenium.dev/about/, 2021. Online; accessed 17 July 2021. 41

[SP18] Douglas Robert Stinson and Maura Paterson. *Cryptography: theory and practice*. CRC press, 2018. 8

[SSS+20] Musa G Samaila, João B. F. Sequeiros, Tiago Simões, Mário M. Freire, and Pedro R. M. Inácio. IoT-HarPSecA: A framework and roadmap for secure design and development of devices and applications in the IoT space. *IEEE Access*, 8:16462–16494, 2020. 23, 24, 27

[Tea20] The HiveMQ Team. Getting Started with MQTT, 4 2020. [Last accessed in $4^{th}$ December, 2020]. Available from: `https://www.hivemq.com/blog/how-to-get-started-with-mqtt/`. 6

[Tew20] Tewari, Aakanksha and Gupta, B. B. *An Analysis of Provable Security Frameworks for RFID Security*, pages 635–651. Springer International Publishing, 2020. Available from: `https://doi.org/10.1007/978-3-030-22277-2_25`. 7

[The21a] The Linux Foundation. Home - Open Source Security Foundation. `https://openssf.org/`, 2021. Online; accessed 17 July 2021. 16

[The21b] The MITRE Corporation. CWE - About - CWE Overview. `https://cwe.mitre.org/about/index.html`, 2021. Online; accessed 17 July 2021. 15

[vul21a] vuldb.com. Vuldb vulnerability and threat intelligence. Technical report, VulDB, 2021. Available from: `https://vuldb.com/download/whitepaper/vuldb_vulnerability_and_threat_intelligence.pdf`. 15

[vul21b] vuldb.com. Vulnerability Database. `https://vuldb.com/?`, 2021. Online; accessed 17 July 2021. 15

[WTB+12] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550, RFC Editor, March 2012. Available from: `https://www.rfc-editor.org/rfc/rfc6550.txt`. 6

[YR15] Tarun Yadav and Arvind Mallari Rao. Technical aspects of cyber kill chain. In *proceedings of International Symposium on Security in Computing and Communication*, pages 438–452. Springer, 2015. 13

[Yu14] Shui Yu. *Distributed denial of service attack and defense*. Springer, 2014. 9

[ZKKK19] Fotios Zantalis, Grigorios Koulouras, Sotiris Karabetsos, and Dionisis Kandris. A review of machine learning and iot in smart transportation. *Future Internet*, 11(4), 2019. Available from: `https://www.mdpi.com/1999-5903/11/4/94`. 45

[ZMKd17] Bruno Bogaz Zarpelão, Rodrigo Sanches Miani, Cláudio Toshio Kawakani, and Sean Carlisto de Alvarenga. A survey of intrusion detection in internet

of things. *Journal of Network and Computer Applications*, 84:25 – 37, 2017.
1