UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Semi-Automatic Generation of Tests for Assessing Correct Integration of Security Mechanisms in the Internet of Things

Carolina Galvão Lopes

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

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

Covilhã, julho de 2021

Acknowledgements

I would first like to thank my family for all the unconditional love and support during this stage of my life. Without their continuous efforts, I could not have reached this far. Having that said, thank you mother, Júlia, father, Frederico, and sister, Beatriz, you are the reason I keep moving forward and trying to reach higher.

Another very special thanks goes to my aunt, Ana Patricia Ferreira, for all the love, support and advice given during the last years of my academic life. Without her guidance, this goal would have been a lot harder to reach.

No less important, I would like to thank my advisor and co-advisor Pedro Inácio and Tiago Simões, respectively, for giving me the opportunity to work/learn with them and for always being available when I had doubts or ideas to discuss.

I am also thankful for the support, suggestions and availability provided by my colleagues Bernardo Sequeiros and Joana Costa.

Last but not least, I would like to thank all my friends for being there and for making this journey more easy and lighthearted. An honourable mention goes to my friend Eduardo Almeida, who has been accompanying me on this journey since day one Thank you for the much-needed coffee breaks without you this journey would have been tremendously more hard and stressful.

The work described in this dissertation was carried out at the Instituto de Telecomunicações, Multimedia Signal Processing – Covilhã 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/Nº11/2019-B00701.

Resumo

A Internet das Coisas (IoT) é um dos paradigmas com maior expansão mundial à data de escrita da dissertação, traduzindo-se numa influência incontornável no quotidiano. As empresas pretendem ser as primeiras a implantar novos sistemas de IoT como resultado da sua rápida expansão, o que faz com que a maior parte do software seja criado e produzido sem considerações de segurança ou testes de segurança adequados. A qualidade do software e os testes de segurança estão intimamente ligados. A abordagem mais bem-sucedida para obter software seguro é aderir aos princípios e práticas de desenvolvimento, implantação e manutenção seguros em todo o processo de desenvolvimento. O teste de segurança é um procedimento para garantir que um sistema proteja os dados do utilizador e execute conforme o esperado.

Esta dissertação descreve o esforço despendido na concepção e desenvolvimento de uma ferramenta que, tendo em consideração as ameaças às quais um sistema é vulnerável, produz um conjunto de testes e identifica um conjunto de ferramentas de segurança para verificar a susceptibilidade do sistema às mesmas. A ferramenta mencionada anteriormente foi desenvolvida em Python e tem como valores de entrada uma lista de ameaças às quais o sistema é vulnerável. Depois de processar estas informações, a ferramenta produz um conjunto de ataques derivados das ameaças e possíveis ferramentas a serem usadas para simular esses ataques.

Para verificar a utilidade da ferramenta em cenários reais, esta foi testada por 17 pessoas com conhecimento na área de informática. A ferramenta foi avaliada pelos sujeitos de teste de uma forma muito positiva. A grande maioria dos participantes considerou a ferramenta extremamente útil para auxiliar a realização de testes de segurança em IoT.

As principais contribuições alcançadas com esta dissertação foram: a criação de uma ferramenta que, através das ameaças às quais um sistema IoT é susceptível, produzirá um conjunto de ataques e ferramentas de penetração para executar os ataques mencionados. Cada uma das ferramentas será acompanhada por um breve guia de instruções; uma extensa revisão do estado da arte em testes.

Palavras-chave

Automatização de Testes, Internet das Coisas, IoT, Testes, Testes de Segurança.

Resumo alargado

Introdução

Esta secção da dissertação têm como principal objectivo apresentar, em português, um resumo alargado da dissertação de mestrado elaborada, visto que a mesma está escrita integralmente na língua inglesa.

Enquadramento, Descrição do Problema e Objectivos

A Internet das Coisas (IoT) está a crescer a um ritmo acelerado em todo o mundo, criando um impacto na nossa vida quotidiana, em 2025 acredita-se que o total de conexões IoT chegará a 22 biliões [Lue18].

Com este crescimento elevado, as empresas competem para ser as primeiras a lançar novos sistemas de IoT, o que leva a que a maioria do software seja projectado e desenvolvido sem medidas e testes adequadas para garantir a segurança. Relatórios indicam que 70% dos dispositivos IoT são vulneráveis *out of the box* [Mat20]. Este tipo de prática pode ser prejudicial para as empresas, visto que o software ou sistema é susceptível a ataques, fazendo com que o cliente perca a confiança na marca/empresa.

Os testes de segurança estão directamente relacionados com a qualidade do software. A forma mais eficaz de obter um software seguro é quando o seu processo de desenvolvimento é executado de uma forma rigorosa em conformidade com os princípios e práticas seguras de desenvolvimento, implementação e manutenção. Os testes de segurança são um processo para garantir que um sistema protege os dados do utilizador e que o mesmo mantém as suas funcionalidades principais conforme o necessário. Num ambiente com um ritmo tão acelerado, os testes precisam de ser rápidos e com baixo consumo de recursos. Para lidar com essa necessidade, a automatização de testes é uma ferramenta essencial.

Objectivos

O objectivo principal desta dissertação é avançar no estado da arte nos tópicos de automatização de testes, testes baseados em requisitos e em modelos. O trabalho de investigação realizado para este objectivo deve levar à criação de uma ferramenta que aconselhe os utilizadores nos métodos de teste específicos para a IoT que estão a desenvolver, ferramentas específicas que eles podem usar para executar o referido teste e, por fim, apresentar ao utilizador alguns detalhes de teste.

Os seguintes objectivos secundários foram definidos para atingir o objectivo principal desta dissertação:

- Levantamento de informação sobre automatização de testes em sistemas IoT, um dos objectivos desta dissertação é contextualizar os testes de segurança no ambiente IoT;

- Uma pesquisa sobre ferramentas de teste de penetração compatíveis com IoT. As ferramentas investigadas nesta etapa serão utilizadas como ferramentas recomendadas para apresentar aos utilizadores;
- O terceiro e último objectivo é o de desenvolver uma ferramenta que, de acordo com o tipo de ataques a que o sistema seja susceptível, produza um relatório com especificações de teste e ferramentas que possam testar o sistema contra essa possível vulnerabilidade.

Principais Contribuições

De acordo com os objectivos estabelecidos na secção anterior, as principais contribuições desta dissertação são: uma revisão extensa do estado da arte nos tópicos mencionados acima; o levantamento de um conjunto de ferramentas de testes compatíveis com IoT; um protótipo de ferramenta que, de acordo com o tipo de ameaças a que o sistema é susceptível, crie uma lista de possíveis ataques associados a testes de penetração e ferramentas que executem os mesmos.

Background e Conceitos Principais

Neste segundo capítulo são abordados os conceitos fundamentais para o desenvolvimento desta dissertação de mestrado. É apresentada a definição de IoT e a sua arquitectura bem como as suas principais vulnerabilidades e ataques a que esta susceptível. De seguida, explora-se área de testes de software e os vários níveis existentes dentro deste tópico.

IoT é definida como um paradigma em que um conjunto dispositivos comunicam através de tecnologia *wireless* e partilham dados/informações para atingir um objectivo [SJK17]. Estes dispositivos, habitualmente, possuem uma arquitectura de três camadas, sendo elas: *Perception Layer*, *Network Layer* e *Application Layer*. Tal como grande parte das tecnologias emergentes, os dispositivos IoT estão sujeitos a vulnerabilidades que os podem incapacitar. De acordo com a OWASP, o top 10 das vulnerabilidades mais comuns em IoT são: *Passwords* fracas, fáceis de adivinhar ou *hard-coded*; Serviços de rede inseguros; Interfaces com ecossistemas inseguros; Falta de mecanismos de atualização seguros; Uso de componentes inseguros ou desactualizados; Protecção física insuficiente; Protecção de privacidade insuficiente; Transferência e armazenamento de dados inseguro; Falta de gestão de dispositivos; Configurações por defeito inseguras; Falta de robustimento físico (*physical hardening*). Após esta análise das principais vulnerabilidades, é feito um levantamento dos principais ataques a que os dispositivos IoT estão sujeitos. Reuniu-se um total de 20 ataques acompanhados de uma breve descrição e de possíveis formas de mitigação dos mesmos.

Teste de software é o processo de verificação e validação de que uma aplicação ou programa vai de encontro aos requisitos comerciais e técnicos que orientaram o seu design e desenvolvimento. O processo de teste garante que o software está a funcionar conforme o esperado e também revela possíveis falhas ou erros no sistema [SPPS17]. Testar soft-

ware é uma tarefa muito complexa e extensa, sendo esta dividida em três níveis de teste [JP16]: testes unitários, testes de integração e testes de sistema. Uma breve explicação de cada nível é feita acompanhada de uma apresentação dos vários testes que se podem fazer dentro de cada nível. Finalmente, é feita a associação entre os vários tipos de teste e em que fase do ciclo de vida de desenvolvimento de software é que estes se devem aplicar.

Trabalho Relacionado e Ferramentas

Nesta terceira secção é apresentado o estado da arte dentro de vários tópicos na área de testes, nomeadamente: Testes baseados em modelo; testes baseados em requisitos; automatização de testes. Por fim, é feito um levantamento de várias ferramentas de testes de segurança automáticas.

No tópico de testes baseados em modelo destacam-se os trabalhos de Naveed *et al.* [Nav17], de Matić *et al.* [MNA+19] e de Mahmoodi *et al.* [MRV+18]. Na área de testes baseados em requisitos os trabalhos de Freudenstein *et al.*[FJR+18], Li *et al.* [LMY+19], Eo *et al.* [ECG+15] e de Aniculaesei *et al.* [AHDR18] foram abordados. Por último, os trabalhos de Swain *et al.* [SMM10], Darmaillacq *et al.* [DRG08], Gafurov *et al.* [GHM18], Yatskiv *et al.*[YVY+19] foram estudados no âmbito de automatização de testes, com especial ênfase nos trabalhos de Varghese *et al.* [VS20] , Yadav *et al.* [YPAO20] e Hwang *et al.*[HAS+20] que são específicos para automatização em IoT.

Para terminar esta secção, são apresentados um conjunto de ferramentas de testes automáticas compatíveis com IoT. Foram apresentadas um total 18 ferramentas cada uma acompanhada de uma breve descrição e para que tipos de ataques a mesma pode ser utilizada.

Projeto, implementação e demonstração do sistema

Nesta secção são identificados os principais requisitos funcionais e não-funcionais do sistema proposto. Esta definição é feita com o intuito guiar o autor no seu desenvolvimento da ferramenta proposta, de modo a que esta produza os resultados esperados e se insira correctamente na plataforma onde vai ser alojada. Os principais requisitos identificados são: a produção de uma ferramenta que produza, de acordo com ameaças possíveis ao sistema, um conjunto de ataques associados a ferramentas de penetração; a solução deve ser modular e utilizar ao máximo os recursos já disponibilizados pela plataforma. De seguida é apresentada uma visão geral do *design* do sistema a desenvolver juntamente com as relações entre o mesmo e os outros componentes da plataforma.

São, também, discutidos os detalhes da implementação da ferramenta desenvolvida no âmbito desta dissertação de mestrado. Primeiramente, é feita uma breve apresentação da *Security Advising Module* (SAM), a plataforma onde foi alojado o módulo desenvolvido. De seguida é apresentado o módulo desenvolvido no âmbito desta dissertação. Por fim, são apresentados os detalhes de implementação do módulo em questão.

A SAM é uma plataforma modular, ou seja é constituída por um conjunto de módulos

todos eles com diferentes objectivos. De momento, na área de IoT, apenas existem três módulos: um para elicitação de requisitos de segurança, um para sugestão de boas práticas e, ainda um outro módulo para recomendação de algoritmos criptográficos leves. Os módulos inseridos na SAM podem ser de dois tipos: o módulo *tradicional*, constituído por um conjunto de questões e respostas; e os *plugins* que não necessitam de nenhum questionário para produzir as suas recomendações.

A ferramenta desenvolvida no contexto desta dissertação intitula-se de *Module for Assessing Correct Integration of Security Mechanisms (ACISM)*. Este módulo é considerado um *plugin*, dentro dos padrões da SAM. Este apenas depende das recomendações dadas pelo módulo de recomendação de algoritmos criptográficos leves e pelas recomendações de um outro módulo, desenvolvido em paralelo por outro colega de mestrado, que apresenta as ameaças a que o sistema em questão é susceptível. Com as recomendações destes dois módulos, é produzido um relatório com os possíveis ataques a que o sistema IoT a ser avaliado é susceptível. Para além dos possíveis ataques, é também produzido um conjunto de ferramentas de teste que verificam se medidas de segurança foram implementadas de modo a mitigar as vulnerabilidades e, consequentemente, impedir esses ataques de acontecer. Se o utilizador necessitar de mais informação, associado a cada recomendação, existe um ficheiro do com notação *markdown* com instruções de instalação e utilização de cada ferramenta para simular o ataque em questão. Por fim, é feita a associação entre cada ficheiro de ataque, as ferramentas sugeridas e as ameaças que resultam na possibilidade do ataque em questão.

Teste e validação do módulo

Para avaliar a utilidade e correcção do módulo desenvolvido, primeiro foi necessário alojar a SAM de modo a permitir, aos participantes, acesso á plataforma. A plataforma foi hospedada num servidor dentro da Universidade da Beira Interior (UBI) utilizando nginx e gunicorn. Foi também criado um certificado digital de modo a disponibilizar aos utilizadores comunicação segura através do protocolo *Hyper Text Transfer Protocol Secure* (HTTPS). A plataforma e todos os seus conteúdos encontram-se disponíveis em `https://securiotesign.di.ubi.pt/`. No entanto, antes de disponibilizar a ferramenta aos utilizadores foi necessário garantir que a mesma estava a funcionar correctamente e de acordo com o esperado. Para automatizar esta parte dos testes foi utilizada a ferramenta Selenium de modo a realizar testes que corressem as várias combinações possíveis de perguntas e respostas. Após esta fase de testes automáticos ter sido realizado com sucesso a plataforma e os seus módulos foram considerados aptos para testes com utilizadores.

De modo a facilitar e de certa forma guiar a interacção dos utilizadores com a plataforma e com o módulo desenvolvido foi elaborado um documento com vários cenários de IoT, um para cada uma das grandes áreas de IoT, nomeadamente: Carga Perecível e Monitorização de Transporte; Optimização da Infraestrutura e Uso de Energia para Carregamento Doméstico de Veículos Eléctricos; Lâmpada Inteligente; *Pacemaker* com uma Interface Móvel; Sistema de Irrigação Inteligente; *Smartwatch*; Fábrica Inteligente. O objectivo

deste documento é que os sujeitos de teste escolham um cenário e que o apliquem nos vários módulos da SAM.

O módulo foi testado com 17 participantes, todos eles com alguma experiência na área de informática. Os resultados destes testes foram apresentados neste capítulo. De um modo geral, o módulo foi considerado bastante útil para auxiliar na fase de testes de segurança, produzindo relatórios claros e com ferramentas adequadas para os testes em questão.

Conclusões e Trabalho Futuro

No sétimo capítulo estão presentes as principais conclusões retiradas após a conclusão desta dissertação. São também apresentadas possíveis melhorias a aplicar, num futuro próximo, ao trabalho desenvolvido.

Após a realização de testes com utilizadores, foi possível concluir que a ferramenta desenvolvida consegue, com sucesso, aconselhar os utilizadores no processo de testes de segurança. Para além disso, é ainda possível concluir que todos os objectivos propostos nesta dissertação foram alcançados.

Como trabalho futuro foram apontados algumas melhorias a realizar. Nomeadamente, converter os guias individuais de *markdown* para *Portable Document Format* (PDF). Foi também sugerido compilar os vários guias produzidos dentro do PDF das recomendações principais. A última sugestão presente nesta secção é a de elaboração de uma ferramenta que, através das recomendações dadas por este módulo e, após algumas questões ao utilizador sobre o sistema desenvolvido, instale as ferramentas e corra os testes sugeridos.

Abstract

Internet of Things (IoT) is expanding at a global level and its influence in our daily lives is increasing. This fast expansion, with companies competing to be the first to deploy new IoT systems, has led to the majority of the software being created and produced without due attention being given to security considerations and without adequate security testing. Software quality and security testing are inextricably linked. The most successful approach to achieve secure software is to adhere to secure development, deployment, and maintenance principles and practices throughout the development process. Security testing is a procedure for ensuring that a system keeps the users data secure and performs as expected. However, extensively testing a system can be a very daunting task, that usually requires professionals to be well versed in the subject, so as to be performed correctly. Moreover, not all development teams can have access to a security expert to perform security testing in their IoT systems. The need to automate security testing emerged as a potential means to solve this issue.

This dissertation describes the process undertaken to design and develop a module entitled *Assessing Correct Integration of Security Mechanisms* (ACISM) that aims to provide system developers with the means to improve system security by anticipating and preventing potential attacks. Using the list of threats that the system is vulnerable as inputs, this tool provides developers with a set of security tests and tools that will allow testing how susceptible the system is to each of those threats. This tool outputs a set of possible attacks derived from the threats and what tools could be used to simulate these attacks.

The tool developed in this dissertation has the purpose to function as a plugin of a framework called Security Advising Modules (SAM). It has the objective of advising users in the development of secure IoT, cloud and mobile systems during the design phases of these systems. SAM is a modular framework composed by a set of modules that advise the user in different stages of the security engineering process.

To validate the usefulness of the ACISM module in real life, it was tested by 17 computer science practitioners. The feedback received from these users was very positive. The great majority of the participants found the tool to be extremely helpful in facilitating the execution of security tests in IoT.

The principal contributions achieved with this dissertation were: the creation of a tool that outputs a set of attacks and penetration tools to execute the attacks mentioned, all starting from the threats an IoT system is susceptible to. Each of the identified *attacking* tools will be accompanied with a brief instructional guide; all summing up to an extensive review of the state of the art in testing.

Keywords

IoT, Internet of Things, Testing, Security Testing, Test Automation.

Contents

List of Figures

List of Tables

Listings

Acronyms

4G	Fourth Generation
ACISM	Assessing Correct Integration of Security Mechanisms
ACM	Association for Computing Machinery
API	Application Programming Interface
ARP	Address Resolution Protocol
ASSERT	Analysis of Semantic Specifications and Efficient generation of Requirements based Tests
CCS	Computing Classification System
CEG	Cause-Effect-Graphs
CSRF	Cross-Site Request Forgery
DoS	Denial-of-Service
DDoS	Distributed Denial-of-Service
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
ECU	Electronic Control Unit
FTP	File Transfer Protocol
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IoT	Internet of Things
IP	Internet Protocol
IPS	Intrusion Prevention System
IT	Information Technology
LAN	Local Area Network
LWCAR	Lightweight Cryptographic Algorithm Recommendation
LTL	Linear Temporal Logic
MITM	Man-in-the-middle
MQTT	Message Queuing Telemetry Transport
NIST	National Institute of Standards
NVD	National Vulnerability Database
ODB	On-board diagnostics
OWASP	Open Web Application Security Project
PDF	Portable Document Format
RC	Requirements Coverage
RFID	Radio Frequency Identification
RPA	Robotic Process Automation
SAM	Security Advising Modules
SBP	Security Best Practices
SDN	Software-Defined Networking

SME	State Machine Executor
SQL	Structured Query Language
SRE	Security Requirements Elicitation
TCGD	Test Case Generator and Driver
TCP	Transmission Control Protocol
TMS	Threat Modeling Solution
UBI	Universidade da Beira Interior
UDP	User Datagram Protocol
UML	Unified Modelling Language
URL	Uniform Resource Locator
VP	Virtual Prototypes
XML	Extensible Markup Language
XSS	Cross-site Scripting
XXE	XML external entity
ZAP	OWASP Zed Attack Proxy

Chapter 1

Introduction

This chapter describes the scope of the dissertation and the underlying open issues that need to be addressed. The hypothesis of this work is presented and the chapter ends with the organization of the document.

1.1 Scope and Motivation

Internet of Things (IoT) is defined as a set of connected devices that communicate through the internet [SJK17]. The architecture of this type of systems is usually divided into three layers, each one of them with individual components. Ensuring security on all of these components is therefore a complex task which needs to be addressed right from conception to avoid the release of insecure IoT systems.

Security testing is a process where the tester tries to find all the vulnerabilities and weaknesses of a system, which could result in a security breach, or where he or she tries to obtain confirmation that a given security requirement is met (e.g., correct integration of a cryptography protocol). In this specific area, there are six main test categories: vulnerability scanning, penetration testing, risk assessment, security auditing, ethical hacking and posture assessment. The ultimate goal of this type of testing is to ensure that a system can not be exploited.

To correctly perform security testing in such heterogeneous environments as the ones where IoT devices live in, it is advisable that this task is performed by experts in the subject matter. However, it is not always possible to have a security expert in the development team, and with the popularity of these devices rising quickly, this has resulted in the release of new products without proper security measures implemented [Mat20]. If insecure devices are placed into production and released into the *wild*, there is a risk of exposing the private information of users or incur in potentially dangerous situations, given that IoT is rapidly spreading to all sectors of modern life. To aid developers ensure that their IoT devices are secure, it is essential to develop automated security testing tools, which is in part, the motivation behind the work described in this dissertation.

The scope of this dissertation falls into the intersection of the areas of information security, security testing and test automation. Under the 2012 version of the Association for Computing Machinery (ACM) Computing Classification System (CCS), this scope can be defined by the categories named:

- **Security and Privacy~Systems Security;**
- *Security and Privacy~Network Security;*

- *Security and Privacy~Software and application security;*

1.2 Problem Statement and Objectives

IoT is growing at an accelerated pace worldwide and its impact in our everyday lives is rapidly increasing. It is forecasted that the total of IoT connections will reach 22 billion by 2025 [Lue18]. With such rapid growth, companies compete to be the first to release new IoT systems and this has lead to the bulk of software being created and produced without the due attention being given to security considerations and without adequate security testing. It has been reported that 70% of IoT devices are vulnerable out of the box [Mat20]. This type of practice can be damaging to the company. If a given software is subject to attacks this could lead the customer to lose trust in this brand/company.

Security testing is directly connected to software quality. The most effective way to achieve secure software is for its development to rigorously conform to secure development, deployment, and sustainable principles and practices. Security testing is key to guarantee that a system maintains functionality as required while protecting the data of the user. In such a fast-paced environment, testing needs to be fast, and low on resource consumption. To cope with increasing demand, the ability to automate tests would be key for any tool aimed at supporting the development of secure systems.

The main objective of this dissertation is to advance the state of the art in terms of test automation, requirement based testing and model specification. It is intended that this research will lead to the development of a tool that guides users in the selection of specific test methods for the IoT system being developed, what tools can be used to run those tests and output some tests specifications for each one of the recommended tools.

The following objectives are key to achieve the main objective of this dissertation:

- Conduct a survey regarding test automation and IoT systems. One of the objectives of the dissertation is to contextualize security testing in the IoT environment;
- Conduct a survey regarding IoT compatible penetration testing tools. The tools investigated in this step will then be used as recommended tools to present to the users;
- Develop a tool that, according to the type of attacks the system may suffer, will produce a report with test specifications and tools that can test the system against a possible vulnerability.

1.3 Approach Taken to Achieve the Objectives

Before development of the proposed tool began, a study of the main related topics in this research area was performed, to contextualize the author and assess the state-of-the-art. The research performed encompassed areas such as IoT, Software Testing and Security

Testing. After it became apparent that the developed tool should be inserted into penetration testing, the author investigated the most popular penetration tools compatible with IoT. These penetration tools were then used as the output of the developed module. Finally, a module titled ACISM was created and inserted into a larger security framework. The module receives a set of threats as input, processes them, and generates a set of attacks the IoT system is vulnerable to. Each of the attacks is accompanied by a set of penetration testing tools to perform those attacks, along with the guidelines on how to use them. After the development of the prototype, several tests were conducted (both automatic and user tests) to validate the correctness and usefulness of the module.

1.4 Main Contributions

The main contributions achieved from the research and development performed in the scope of this Master's dissertation are:

1. An extensive research on the state of the art in testing namely, in Model-Based Testing, Requirement-Based Testing and Test Automation, which is materialized in chapters 2 and 3;
2. A tool that, by receiving a list of threats the system is vulnerable to, will produce a report with a set of attacks derived from those threats. Each one of the suggested attacks will be accompanied by a set of penetration tools to simulate it, followed by an instructional guide on how to use each tool. This tool is the main subject of chapters 4 and 5 is it is the main foreground of this work.

It should be mentioned that the development of the guides on how to use the tools has led the author to secondary contributions that are only briefly mentioned along the dissertation, but need to be emphasized here. For example, in section 6.2, a tool for production of test values for conditional coverage testing, specially crafted for Python, is presented. Additionally, initial efforts to embed artificial intelligence into the tools and modules of the framework were done 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.

1.5 Dissertation Outline

This dissertation is organized into six chapters, which can be summarized as follows:

- Chapter 1 – Introduction – presents the scope and motivation, states the problems and objectives, the proposed approach and ends up with the main contributions of this work;
- Chapter 2 – Background and Main Concepts – covers the fundamental concepts that contextualize and need to be understood to enable the development of the proposed tool;

- Chapter 3 – Related Work and Underlying Tools – describes state of the art in model-based testing, requirement based testing and test automation. Additionally, it discusses some popular security testing tools;
- Chapter 4 – System Design, Implementation and Demonstration – presents the requirements and flow of the proposed tool, followed by the main technical details of the implementation of the developed tool;
- Chapter 5 – Testing and Module Validation – describes the steps taken to prepare the developed tool for user testing and discusses the results of the tests done by users;
- Chapter 6 – Conclusion and Future Work – captures the main conclusions and achievements of this work, and suggests possible future improvements;
- Appendix A – includes the document elaborated for the test subjects to guide them while interacting with the developed tool.

Chapter 2

Background and Main Concepts

2.1 Introduction

This chapter describes the basic concepts of IoT and automatic security testing. Firstly, section 2.2 will introduce the topic of IoT, its operation, vulnerabilities and threats this type of systems face. This description is followed by a fairly detailed discussion to security testing in section 2.3.

2.2 Internet of Things

The Internet of Things, a term coined by Kevin Ashton in 1999, is defined as a collection of many connected devices that can communicate through wireless links and share data/information to achieve an objective [SJK17]. The IoT architecture is usually divided into three layers: the perception layer, the network layer and the application layer. Although it has been proven that this is not the most security-friendly architecture when it comes to design such system, it is the most popular amongst these types of devices [KSH20]. A brief explanation of each layer is presented bellow [SJK17, CL18, KSH20]:

- The first layer of an IoT is the **Perception Layer**, which collects information from the real world through Radio Frequency Identification (RFID), sensors, Global Positioning System (GPS), other hardware devices and incorporates the information into the digital world. Some of the attacks this layer is vulnerable to are Skimming, Eavesdropping, Cloning, Buffer overflow, and others;
- This is followed by the **Network Layer**, which is responsible for transporting information between the perception layer and the application layer. This layer uses a variety of networks to communicate such as WiFi, Fourth Generation (4G), satellite, etc. This layer is susceptible to Network traffic sniffers attacks, Signal replay, Signal hijacking, and others;
- The last layer is the **Application Layer**, which provides the user with a variety of services. IoT can be interacted with by users through various applications in different environments. The application layer protocols define the application interface with the lower layers to send data over the network. Data Integrity, Data Confidentiality and Data Authenticity should be guaranteed by this layer. If these requirements are not implemented in the system, this layer can suffer from buffer overflow, Cross-site Scripting (XSS), SQL injection, social engineering attacks and password attacks.

2.2.1 Vulnerabilities in IoT

A vulnerability is a weakness or a programming fault found within a system that can be exploited by an attacker to cross privilege boundaries. Open Web Application Security Project (OWASP) identified the top 10 most common security risks in IoT. If an IoT system possesses any one of these vulnerabilities, the security of the user is compromised. The remaining part of this section describes these top 10 security risks.

Weak, Guessable, or Hard-coded Passwords is when the system contains badly implemented authentication functions that allow attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume, temporarily or permanently, identities of other users [OWA18]. This type of vulnerability can appear in the form of hard-coded passwords, which according to Rizvi *et al.* [ROC+20] is one of the most common vulnerabilities in IoT.

Insecure Network Services is when insecure or unneeded services are running on the device and can compromise the confidentiality, integrity/authenticity, or availability of information and even allow unauthorized remote control [OWA18, ROC+20].

Insecure Ecosystem Interfaces is related with insecure Web, back-end Application Programming Interface (API), cloud, or mobile interfaces in the ecosystem outside of the device that enables it or its related components to be compromised. A lack of authentication/authorization, a lack of poor encryption, and a lack of input and output filtering are all common issues [OWA18].

Lack of Secure Update Mechanism refers to the lack of mechanisms to securely update the system, for example lack of firmware validation on the device, lack of secure delivery (un-encrypted in transit), lack of anti-rollback mechanisms, and lack of notifications of security changes due to updates [OWA18].

Use of Insecure or Outdated Components refers to the use of insecure or deprecated software libraries/components that can allow for the security of the device to be compromised [OWA18].

Insufficient Privacy Protection concerns the vulnerability that is present when there is personal information from the user stored in the ecosystem or in the device that is used insecurely, improperly, or without permission [OWA18].

Insecure Data Transfer and Storage is defined in [OWA18] as the *"lack of encryption or access control of sensitive data anywhere within the ecosystem, including at rest, in transit, or during processing"*.

Lack of Device Management is the lack of support, namely security support, on devices deployed in production including update management, asset management, systems monitoring, secure decommissioning and response capabilities [OWA18].

Insecure Default Settings is when there are *"devices or systems shipped with insecure default settings or lack the ability to make the system more secure by restricting*

operators from modifying configurations.”, as per definition of OWASP [OWA18]. For example, malware like Mirai tries to use default username and password combos to gain access to systems and compromise them [FMCS20].

Lack of Physical Hardening or physical security weaknesses can allow an attacker to access sensitive information that can help the attacker take control of the device in the future [OWA18]. An example of this vulnerability is when USB ports allow access to the device using features intended for configuration or maintenance. This can cause unauthorized access to the device [FMCS20].

2.2.2 Main IoT Threats

In such a connected world, where everyday objects have the ability to collect and transmit data through the Internet to make the everyday life easier, the danger of harmful or critical security threats to materialize is imminent. To better contextualize this subject, the definition of several threats, previously highlighted, and countermeasures are presented bellow. Table 2.1, presented at the very end of this section, shows a summary of this information also:

1. **Brute Force** (attack) is when an attacker tries to guess the user credentials by trying a significant number of different combinations. Some IoT devices come with default user credentials which, if not changed, can provide an attacker with easy access to the system. This attack can be prevented by using Intrusion Prevention System (IPS) technology [RRPB19];

2. **Buffer Overflow** is when an attacker takes advantage of improperly used pointers and memory management to gain access to information from or change the behaviour of the system. Secure coding and pointer restriction can be used to stop this kind of attack. Using virtualization for sandboxing is also a way to prevent this attack [RRPB19];

3. **BlueBorne attack** explores a vulnerability present in devices that can communicate through Bluetooth and it can happen even if the device is not paired with the attacker. Once exploited, this vulnerability allows the attacker to gain control of the device. This attack can be countered by updating vulnerable systems [RRPB19];

4. **Sybil attack** is when an attacker creates malicious nodes or compromises legitimate ones in order to create a bottleneck or alternate paths in the network to decrease the performance of the overall system. The prevention of this attack can be done through the implementation of node validation [RRPB19, DV17];

5. **Injection Attack** concerns the situation in which assailants use weaknesses in the communication protocols, APIs or entry points to inject information into the network. A way to prevent this attack is by creating a defence mechanism as the one described in [HEN+13, RRPB19, DV17];

6. **Man-in-the-middle (MITM) attack** is the designation used to define attackers

that might manipulate or eavesdrop the traffic between devices. As an example, an attacker might eavesdrop traffic between a smart device and the gateway by using an Address Resolution Protocol (ARP) poisoning to redirect all traffic to his/her device. This particular attack can be prevented by using a Semi-state ARP cache table [Dat18] [RRPB19] [DV17];

7. **Domain Name System (DNS) poisoning** is performed by poisoning the cache of DNS servers, which is responsible for the translation from domain names to Internet Protocol (IP) addresses and *vice-versa*, in which case the attacker can redirect the data from the device to a destination designed by himself (e.g., to perform eavesdropping). The prevention of this attack is already possible via the utilization of Domain Name System Security Extensions (DNSSEC) [RRPB19];

8. **Replay attacks** occur when the attacker analyses the traffic and keeps a copy of it so that he/she can use it in a different context in order to control the device. This attack can be prevented through encryption or message authentication [RRPB19];

9. **Wormhole** concerns the scenario in which two devices are placed in an IoT network (e.g., a network of sensors) so that network traffic is captured in one place and sent to another, aiming to create performance issues and congestion of the network. By monitoring the nodes of the system this attack can be prevented [RRPB19, DV17];

10. **Structured Query Language (SQL) injection** is the attack in which someone can execute commands if the user input from the Web application is not properly validated. This attack can lead to the gathering of all database information by a malicious entity. To prevent this attack, input validation should be applied to the Web application [RRPB19];

11. **Command Injection**, whose purpose is to execute arbitrary commands on the host operating system via a compromised program. When an application sends insecure user-supplied data (forms, cookies, Hypertext Transfer Protocol (HTTP) headers, etc.) to a device shell, command injection attacks are possible [Zho]. Since IoT devices are usually run under a Linux operating system they are susceptible to this type of attack [ROC+20]. Such vulnerabilities could be mitigated by applying input validation [ACSC20];

12. **Code Injection** occurs when an attacker physically injects malicious code into one of the IoT nodes. With this attack, the perpetrator can gain total control of the system [DV17]. Once again, as the attacks presented above, this attack can be solved by applying input validation into the system;

13. **Log Injection Attack** enables the alteration of the log files to the attacker benefit. With log injection the attacker can cover the tracks of any cyber attack rendering the logs useless for further investigation. It is also possible to inject malicious codes or scripts with this type of attack [NSCC20]. The solution to this attack can be achieved by encrypting the logs and saving them at the device level or at a trusted third party or even by using a logs security scheme specific for IoT like the one proposed by

Noura *et al.* [NSCC20];

14. **Distributed Denial-of-Service (DDoS) attack** is when an attacker depletes the network by flooding the device with traffic. This can be achieved by compromising a large number of computers forming a botnet and controlling them to flood the network with requests. This attack can be prevented by the utilization of a Firewall or through the implementation of Software-Defined Networking (SDN) [WZLH14, RRPB19, DV17];

15. **Weak authentication** is when an IoT device has a weak authentication system and the attackers can gain access to a said device by using brute force, in order to discover the access credentials. This attack can be mitigated by using proper authentication methods [RRPB19];

16. **Malicious Applications** are payloads that an attacker tries to push to devices so that he can spread its control over to the IoTs connected to the infected system. To prevent this type of attack the user should install an Anti-Virus software [RRPB19];

17. **Cross-Site Request Forgery (CSRF)** is an attack that takes place when an intruder uses the Web browser of the victim to conduct an unauthorized activity on a trustworthy website, without the knowledge or consent of the user. This attack takes advantage of the confidence the user has on a particular website. This attack can be mitigated through the usage of random tokens, using POST rather than GET (when possible) and limiting the lifetime of authentication [SV11];

18. **Format String Attack** is when the application evaluates the submitted data of an input string as a command. The attacker could then execute code, read the stack, or trigger a segmentation fault in the running program, resulting in new behaviours that could jeopardize the protection or stability of the system. The mitigation of this attack can be done by creating the practice of writing secure code and treating format functions vulnerable to this attack such as `printf` or `fprintf` [OWA];

19. **Path/Directory Traversal** is an attack that makes use of inadequate security validation of input file names that include `/.` from the user. Malicious users will achieve a directory jump in this scenario, allowing them to traverse to the parent directory of the server. This attack will give the application access to files that the default privilege does not allow [AYL+21]. This attack can be mitigated by doing validation of the input provided by the user;

20. **XSS** is when the Web browser of the victim is used to execute a malicious script by XSS. It essentially redirects the victim to another website, causing the victim to engage in DDoS attacks or even stealing the session of the user. To mitigate this attack it is needed to perform input validation [RKPR18].

Threats	Countermeasures
Brute Force	Intrusion Prevention System
Buffer Overflow	Secure Coding Pointer Restrictions Virtualization
BlueBorne Attack	Updates
Sybil Attack	Node Validation
Injection Attacks	Defense Mechanisms [HEN+13]
Man-in-the-Middle	Semi-state Address Resolution Protocol cache table
Domain Name System poisoning	Domain Name System Security Extensions
Replay Attack	Encryption
Wormhole	Monitoring the Nodes
Structured Query Language injection	Input Validation
Command injection	Input Validation
Code injection	Input Validation
Log injection	Encrypting Logs
Distributed Denial-of-Service attack	Firewall Software-Defined Networking
Weak Authentication	Proper Authentication Methods
Malicious Applications	Anti-Virus
Cross-Site Request Forgery	Random Tokens Using Post Limit lifetime of authentication
Format String	Secure Coding
Directory Traversal	Input Validation
Cross-site Scripting	Input Validation

Table 2.1: Threats targeted to IoT and respective countermeasures [RRPB19, DV17, HEN+13, ROC+20, Dat18, Zho, ACSC20, NSCC20, WZLH14, SV11, OWA, AYL+21, RKPR18].

2.3 Software Testing

Software Testing is the process of verification and validation that an application or program meets the business and technical requirements that guided its design and development. The process of testing ensures that the software is working as expected and it also uncovers flaws or errors in the system [SPPS17]. Testing software is a very complex and extensive task and it is divided into three levels of testing [JP16]: **Unit Testing; Integration Testing; System Testing**.

Figure 2.1 contains a diagram with all the testing levels and techniques approached in this dissertation, which will be further discussed in the following sections.

2.3.1 Unit Testing

This type of testing is where the most basic parts of the system are tested. Inside this level of testing, there are three main techniques [SM17]: (i) **Black Box**, also known as functional or specification-based testing; (ii) **White Box**, a technique that derives test data from the code; and (iii), **Grey Box**, a testing technique performed with limited knowledge of the code.

In the Black-Box method, the user can perform *Equivalent Class Partitioning* tests that divide the input data into equal partitions so as to derive tests from each partition [Tute];

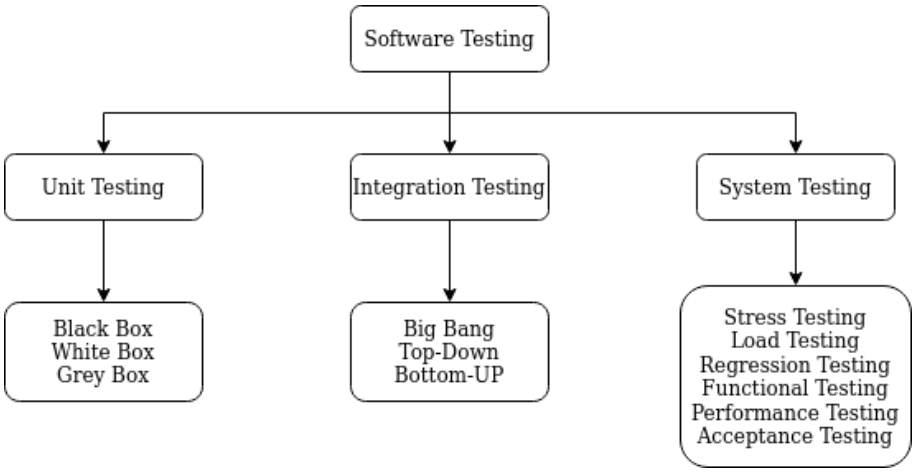

Figure 2.1: Diagram representing testing levels and techniques.

Boundary Testing is a technique in which boundary values are tested [Tutc] and *Security Testing*, which is used to ensure that an information system protects data while maintaining functionality as expected [MAM09].

The White Box testing method, can utilize various types of testing such as: *Statement Coverage*, in which all statements of the code are tested, *Branch Coverage*, which ensures that all branches are tested at least once, *Path Coverage*, that consists of testing all possible paths in the code [Tuti] and *Condition Coverage*, that is when all the boolean conditions in the code are evaluated as both true and false [Tutd].

2.3.2 Integration Testing

After the completion of Unit testing, the units undergo the processes of integration that is followed by Integration Testing, where the functioning between units is verified [Tutf].

Through this method **Big Bang Testing** can be performed, a strategy where all units are linked at once, resulting in a complete system [Tuta]; **Top-Down Testing** can also be performed in this method, it is used to simulate the behaviour of lower-level modules that are not integrated yet, in order to test the higher-level modules that are already implemented [Tuth]; finally, the **Bottom-Up technique** can also be used, in which the lower components of the hierarchy are tested individually and the components that depend on this lower modules are tested after [Tutb].

2.3.3 System Testing

Lastly, after the integration phase, the system is tested as a whole and the results evaluated in comparison to the requirements, this process is called System Testing [Tutg]. To test all aspects of software, various types of testing were also introduced by Mustafa *et. al* [MAM09]. Below, several types of testing are briefly described:

- **Stress Testing** which is used to evaluate a system when forced to go beyond limits of the systems requirements, to determine the load under which it fails;

- **Load Testing** is conducted to evaluate the compliance of a system with the performance requirements that were specified;
- **Regression Testing** is performed to a system that was, previously, working correctly but, after some changes, it stopped working as intended;
- **Functional Testing** this type of testing is applied to a fully complete and functional system, so as to evaluate its conformity with the specific requirements;
- **Performance Testing** is the type of testing that evaluates the performance of the system in various scenarios;
- **Acceptance Testing** involves running a set of tests to a complete system.

2.4 Security Testing and Software Development Life Cycle

Security Testing of a system is about trying to find all the potential system vulnerabilities and weaknesses, that could result in a security breach. The primary goal of testing is to keep your system free from any threats and vulnerabilities, so that the system cannot be exploited, according to A. Kalwan [Kalb].

There are six types of Security Testing, namely:

- **Vulnerability Scanning** An automated software scans the system against vulnerabilities that were identified. This scanning can be performed both Manually and Automatically and it identifies the weaknesses in the network and system;
- **Penetration Testing** The kind of test that simulates an attack from a malicious agent. The main objective of this type of testing is to check for potential vulnerabilities in case there is an external hacking attempt;
- **Risk Assessment** Analyzes the security risks observed in the organization. Risks are categorized as Low, Medium and High;
- **Security Auditing** This is a kind of internal inspection of the application and operating system in order to check for security flaws;
- **Ethical Hacking** is the set of activities used to expose the flaws in the computer systems (e.g., of an organization), in which hackers attempt to attack said systems;
- **Posture Assessment** combines security scanning, ethical hacking and risk assessments to represent the overall security of the organization.

The *Software Development Life Cycle* contains several phases in which security tests can/should be performed. In the requirements phase, where the requirements for a specific system are collected, it is important to do a Security analysis for requirements and checking for any sort of misuse cases. The *design phase* of software should include security risks analysis for designing. In the *coding and unit testing phase*, the system should be

submitted to security tests by using the white box testing method. In the *integration testing phase*, security can be assured by performing black box testing. In the *system testing phase*, it is important to again perform black box testing and vulnerability scanning to guarantee security. Finally, in the *deployment phase*, it is important to conduct penetration testing and vulnerability scanning along the time.

2.5 Conclusion

In this chapter, a small introduction to IoT was given, followed by an explanation of the architecture used amongst this kind of devices and possible attacks they might be subjected to. The topic of software testing, with a focus on Security Testing, was also introduced.

It can be concluded that, even though IoT systems are becoming more and more popular, there are some risks associated with their use due to potential security flaws. To prevent these flaws and increase system security and user confidence, it is essential that security testing is implemented in every phase of the IoT system development life cycle.

The work presented in this chapter provides the basis for what is proposed later on in terms of the recommendations regarding testing tools provided by the module implemented in Security Advising Modules (SAM).

Chapter 3

Related Work and Underlying Tools

3.1 Introduction

In this chapter, the state of the art regarding Software and Security testing will be presented and discussed. This discussion, featured in section 3.2, is focused on three key topics: Model-Based Testing, Requirement-Based Testing and Test Automation. The research was conducted on these topics to guide the author in the process of developing the proposed tool, since many different approaches could be taken at the initial phase of the project. A review of the tools available to test IoT systems is also performed in section 3.3 including details on which system vulnerabilities each tools aims to test.

3.2 Related Work

This section will present the related work in three main topics: Model-Based Testing, Requirement-Based Testing and Test Automation. Each of the subsections included bellow will contain the work of authors inserted in the respective topic along with a brief description.

3.2.1 Model-Based Testing

In order to create a complete set of tests, a model of the environment is required. With this model, it is possible to automatically generate a test suite with good coverage. Usually the level of abstraction of these tests is intrinsically related to the level of abstraction of the model itself.

In the work of Naveed [Nav17], the authors state the importance of validation, through testing during the development process. The execution of tests consists on the creation of test cases that are designed as a sequence of inputs. To generate the sequence of inputs needed to test a system against the specifications, the authors propose a tool called TEAGER that takes Unified Modelling Language (UML) models that are generated through the Papyrus UML tool and, after being imported and executed, these models generate test cases.

Two independent components are made available by the TEAGER tool: Test Case Generator and Driver (TCGD) and State Machine Executor (SME). With TCGD, it is possible to generate test cases so as to validate the specifications against the system under analysis. With this tool, requirements, facts and assumptions are modelled, along with patterns defined for the different state machines.

Matić *et al.* [MNA+19] focus on the topic of home automation. In this type of environment, one of the greatest challenges are testing the product, since it is necessary to test the connection with a large number of devices of various manufacturers, due to the possibility of interaction with multiple devices. In this paper, the authors modelled an end-to-end device and client components of an existing home automation IoT system. Their goal was to test the cloud component of said system and verify the ensured high availability and throughput.

The system was tested through simulations, due to the impossibility to manually set up the real-world system. For this purpose, the authors created models of the gateway and mobile application. The system was tested to verify if the desired number of gateways could be connected within an acceptable period and if the system was still stable after running for long periods.

Through this approach, they were able to develop models mimicking the system behaviour and conduct a series of tests. By the use of these models, it was possible to verify the correct implementation of some services while, also, detecting problems with other services. Model-based load tests enable the identification of the bottlenecks of the system ahead of deployment to production, which represent a significant cost reduction compared to finding and fixing issues after system release.

Mahmoodi *et al.* [MRV+18] proposed an approach to utilize Virtual Prototypes (VP) at the system level to enable security evaluation during the design stage. The proposed tool simulates both the hardware and the software of a single IoT device, allowing for the emulation of the attack surface, attack behaviour modelling and extensive system analysis. The authors state that through the use of VPs, instead of physical prototypes, it is possible to make tests in an early development phase, allowing to address the weak points during the development phase, enabling the resolution of any weaknesses identified at this stage, thereby supporting the Security by Design practice. The tool developed by Mahmoodi *et al.* consists of a simulation framework, which is C-based, modular, and re-configurable. It includes also a graphical front end to ease the usage of the system. It is also possible to configure the VP to adjust the attack surface and protection goals. An injection mechanism is also made available to the user.

To conclude, their work the authors state that with abstract simulation models, it is already possible to detect unintended behaviour of a system in its early stages. They proposed a comprehensive framework that creates VPs and an injection tool that allows the user to specify different attacks easily through Python expressions. This tool also allows for the modification of the internal information to simulate an attack scenario without any manual adaptation of the VP.

3.2.2 Requirement-Based Testing

Requirement-based testing is a testing method in which test cases, data and conditions are derived from requirements [EMB15]. This type of testing is very susceptible to hu-

man error (since the requirements are produced by humans). To make this process both easier and more precise, a set of tools were developed to transform the man-made requirements into more formal requirements. The devised tools are able to convert those formal requirements into test cases, completely ready to be undertaken.

To facilitate the work of generating tests through requirements, minimizing cost while maintaining quality, Freudenstein *et al.* [FJR+18] created *Specmate*, an open-source tool that designs system tests from requirements, provides lightweight modelling techniques to capture these requirements, test generation facilities to create test specifications, and functions to derive test procedures or test-scripts from specifications. Currently, *Specmate* supports two main workflows that can be applied to two different types of tests:

- The first workflow takes as input text requirements that describe a property that the system must have and uses these requirements to create Cause-Effect-Graphs (CEG). Secondly, it generates test specifications containing logical test cases derived from the CEG and adapts the test cases to its objective. Finally, a test-procedure is created for each logical-test-case;
- The second workflow consists on designing end-to-end tests based on business process models. This kind of model describes an action taken to reach a goal in a system. To design the end-to-end test Specmate first uses a modelling language to collect the business process modules and then generates test cases from the collected models.

A similar tool to the one previously discussed is presented in the work of [LMY+19], here a toolchain called Analysis of Semantic Specifications and Efficient generation of Requirements based Tests (ASSERT) was developed to formally capture requirements and to generate a complete set of test cases that satisfy the DO-187C standards.

The process of generating tests using the ASSERT tool begins with the Requirements Capture Environment (RCE) that allows the users to write a set of requirements in a formatted way. The requirements, once defined, are sent to the Requirements Analysis Engine (RAE) to be analyzed and to eliminate conflicting aspects. Finally, using the Automated Test Generation (ATG) tool, the set of requirements are converted to test cases. To comply with the DO-187C standards the authors apply Satisfiability Modulo Theories (SMT) solvers to generate an optimal set of test cases and procedures.

The authors state that with the ASSERT tool it is possible to formally capture the requirements, that are translated to XML files, analyzed and a set of test cases is produced. Later, these test cases are filtered, optimized and simplified to reduce the number of tests to the minimum while still maintaining quality. This tool is already being used in several projects and its use has resulted on a reduction of testing costs and an improved cycle time.

Eo *et al.* [ECG+15] have proposed a Combinational Testing Framework (CTF) that combines different requirement-based methods in the generation of tests such as Equivalence Class Partitioning (ECP), Boundary Value Analysis (BVA), Choice Relationship Framework (CRF) and Predicate Testing to minimize the number of tests and improve the overall

code coverage of the tests performed to a system.

The proposed framework operates through four steps: First, after an input domain of the parameters is given the tool uses equivalence partitioning to generate sub-domains. The boundaries of each partition are identified. Then, the parameters must be prioritized according to the requirements given by the user. Within this tool, high priority parameters are those that interact with values from other parameters. These will be used in the choice relation framework.

With all the equivalent classes identified, a choice relation table is constructed. In this table, each parameter is treated as a category and each equivalent class as its choice. If any incomplete test cases are generated uni-dimensional partitioning is applied to guarantee that at least one value is coming from each partition.

After experimenting with this method in the automotive domain the authors were able to efficiently reduce the final effective number of test cases by 42% to 88%. They were also able to obtain improved code coverage by introducing this technique during earlier phases of software development.

SCADE is a toolchain used by Aniculaesei *et al.* [AHDR18] in order to automate the construction of test cases from requirements, formalized trough Linear Temporal Logic (LTL), and mutant testing within this toolchain. In this work, the authors also implemented a language called SCADE LTL to integrate LTL in SCADE.

This system undergoes a series of tasks before it completes the generation of requirement-based tests. Firstly, the requirements of the system under test are manually created following the LTL. The models of the system are built using the SCADE SUITE and are compliant with the requirements, therefore, satisfying the LTL obligations. After the construction of the system models, the tool uses Requirements Coverage (RC) to build the trap proprieties of the correspondent LTL obligations in order to create traces.

A finite-state automaton is built from a given LTL obligation and the SCADE explores the automaton to find violations of the obligations. Then, the test suite is applied to a set of system mutants in order to execute the created test cases. After the evaluation of the test suite generated in the scope of an Adapative Cruise Control System, the authors stated that the generated test suite killed 80% of the system mutants.

3.2.3 Test Automation

Software testing is a very complex and extensive task, in order to achieve an efficient level of testing it is necessary to have a good set of tests. Manual test generation requires a lot of time and effort and it does not guarantee good coverage but this type of testing will increase the expenses of software testing. To decrease the expenses of software testing developing automated testing tools became of uttermost importance [VG16].

Swain *et al.* [SMM10] presents a comprehensive technique to generate tests from UML models. In their approach test cases are derived from analysis artefacts such as use cases,

their corresponding sequence diagrams and constraints specified across all these artefacts.

The tool proposed was implemented in Java and it is called Comprehensive Test (ComTest). This tool takes UML use case and sequence diagrams converted to XML format. Then, it parses the XML file constructing a virtual graph, finally, it generates test sequences. To create test scripts with this tool the user needs to generate the system model using a CASE tool, like MagicDraw, import the XML model into ComTest and the generation of the tests is executed.

This tool is divided into three modules: the XML parser which parses a file into XML format; the handler which handles the parsed document in order to convert the extracted UML elements into objects to be, logically, prepared into UML diagrams; the Test Case generator which transverses the graph developed by the handler and it generates test cases as specified.

With this tool, the authors can construct Use Case Dependency Graph (UDG) from use case diagrams and through sequence diagrams they can generate Concurrent Control Flow Graph (CCFG) for test sequence generation. This tool can be used for integration and system testing and the test cases generated are suitable for detecting object integration and operation faults, synchronization and dependency of use cases.

In the work of Darmaillacq *et al.* [DRG08], the authors propose a formalization of the requirements to generate abstract test generalizations from the formal rule. After the generation of abstract tests, the tool also initializes the variables needed to run a concrete test and executes the test and deliver a verdict.

The main functions made available by this tool are:

- Security Rule Formalization Function $\longrightarrow$ Takes informal requirements and turns them into a formal rule;
- Test Generation Function $\longrightarrow$ Turns the formal rules into abstract tests;
- Variable Instantiating Function $\longrightarrow$ Converts abstract tests into concrete tests;
- Test Execution Function $\longrightarrow$ Using the concrete tests it delivers a verdict about the conformance of the rule.

The authors state that it is not possible to fully automatize the process of generating and executing requirement-based tests, specifically it is not possible to create a method to automate tasks such as formalizing informal requirements or deriving functional information about the system under test. Due to this impossibility, the method proposed by the authors is semi-automatic since some tasks need to be done manually, such as security rule formalization, tile production and system description.

After testing this tool in the domain of electronic mail, the authors state that it allowed the uncovering of mistakes in the network setup that causes policy violations.

Gafurov *et al.* [GHM18] proposed a tool in which the test engineer implements the test steps and these steps are automatically executed. The test steps are specified by a natural language that is understandable by a non-technical person with domain knowledge so that a test analyst, with no coding skills, can organize these automated steps combined with test input to create an automated test case.

The architecture proposed in this article is divided into various layers:

- Test Implementation Layer $\longrightarrow$ In this layer the Test Automation Engineer implements step by step the test set. The automated test steps, alone, are not very meaningful so they are organized into automated test cases by the Test Automation Engineer. There are two types of automated test cases: the template automated test cases, created by the Test Automation Engineer, and the ordinary automated test cases that are designed by a test analyst;
- Test Definition Layer $\longrightarrow$ In this layer the test analyst is responsible for using the automated test steps created in the previous layer to create new automated test cases;
- Test Execution Layer $\longrightarrow$ The Test Analyst, in this layer, will arrange a set of Automated Test Cases into test suites, execute them and analyze their outcome;
- Test Adaptation Layer $\longrightarrow$ In this layer the Test Automation Engineer implements the test interface at which the Automated Test Cases shall interact with the system under test.

To test their method, the authors used this system on a large Web application of electronic health services in Norway (`Helsenorge.no`). They state that their solution enabled moving a part of the test automation tasks from a Test Automation Engineer to a Test Analyst with domain knowledge. As of the writing of this article, *Helsenorge* has automated 193 test cases and the efficiency ratio of the tool is 0.48.

In the work of Yatskiv *et al.* [YVY+19] a testing method based in Robotic Process Automation (RPA) is presented. RPA is an approach to business workflow automation, in which the program emulates user actions within the graphical user interface to achieve a result.

The authors enumerate some of the advantages of using RPA such as: not having the necessity of changing the existing Information Technology (IT) to deploy the RPA, making it very easy to implement this solution; the robots are available to work 24 hours a day and make no errors while performing their tasks. RPAs are usually assigned with frequent and repetitive tasks, tasks with extreme importance for the business, tasks that require the processing of large quantities of data and tasks that can be turned into a set of strict rules.

Usually, a RPA solution includes a process design environment to describe the rules and procedures, the process execution environment, the managed environment to control all the components, the process analysis environment and some additional components that might be needed for a specific task.

To test their method the authors compared a RPA tool named Work Fusion RPA Express to a well-known testing tool - Selenium Web Driver. As a result, they observed that even though the execution time between Selenium and the RPA is the same, the first tool does not provide the possibility of data generation. One of the most important characteristics in favour of the RPA is that the tests can be written without any code. However, it needs more resources and might lead to bigger execution times when compared to Selenium.

Testing IoT is considered a very hard task due to their highly dynamic and heterogeneous nature. The main difficulties in testing IoTs come from integration, fast autonomous responses, triggering functionalities and operational optimization. In [VS20], it is studied the use of commercial tools, namely Selenium, to test IoT systems and the efficiency of said testing tools against these types of systems.

After some experiments with an IoT system developed in Node-Red the authors concluded that the use of commercial tools to test IoT is feasible, but some problems, such as high data volumes and parallel transmission and processing of data, need to be addressed comprehensively for complete integration.

In the work of Yadav *et al.* [YPAO20] IoT-PEN, a penetration test framework for IoT is introduced. This framework consists of a server-client architecture and it is an end-to-end, scalable, flexible and automatic penetration testing framework. The goal of this tool is to discover all the ways an attacker can breach the system.

IoT-PEN, while performing an examination takes into consideration the product version, the product name and vendor, this information about the products and possible vulnerabilities is stored in a National Vulnerability Database (NVD). The tool follows the simple principle of plug and play for penetration testing, it is composed of different modules to consider the heterogeneity of IoT systems, the user has the possibility to select the module that best fits the system under test and a personalized framework is created.

The tool developed in the scope of this article can be divided into independent microservices capable of running their own:

- Pentesting setup installation $\longrightarrow$ Since IoT-PEN follows a server-client architecture and all the nodes and server are prepared to initiate testing to install this tool it is only needed to apply a patch to all the nodes;
- Get current state information of each node $\longrightarrow$ To communicate to the server IoT-PEN uses MQTT protocol. Each node uses a Extensible Markup Language (XML) format to state its information, and said information gets collected by the server for further automatic processing;
- Extract CPE from .xml file generated by Nmap $\longrightarrow$ In this service the tool parses the XML to extract all the information needed from the nodes and later generate CPE;
- Pre-Post condition generation for the reported vulnerabilities & Target-graph generation $\longrightarrow$ after locating all the possible vulnerabilities in the previous step the system finds the prerequisites and post-conditions for each-reported vulnerability;

- Analysis of attack-paths & Recommendations $\longrightarrow$ The tool generates recommendations of all possible attack paths and optimization techniques for each target path.

In conclusion, the authors state that IoT-PEN is a first-of-its-kind tool for automatic end-to-end penetration framework for IoT. Most of its computation is done on cloud servers and possible on the edge rather than in the IoT devices.

In this article, it is stated that one of the most challenging aspects in IoT is ensuring interoperability between the system components. To guarantee this requirement conformance and interoperability testing are required. Hwang *et al.* [HAS+20], have created a tool entitled AUTOCON-IoT that automatically executes conformance testing in a large number of constrained IoT systems.

The tests are started by AUTOCON-IoT by receiving as input control file with all the data required to perform testing. It contains a type of the system under test, features integrated, information about the communication protocols to create a test connection, the testing protocol, a list of test cases to execute, and serialization formats. If the test suite is developed in a format not accepted by IoT the system converts the test suite into an IoT supported format. Then, the test suite executes each case synchronously with the system.

With this tool, the authors claim to be able to assure interoperability of multiple implementations, without setting up interoperability testing between each implementation individually. Furthermore, by minimizing the number of errors and accelerate the IoT market by certifying IoT applications, AUTOCON-IoT is expected to reduce costs and human intercession.

3.3 Testing Tools

In this section, it will be presented the tools studied to apply to an IoT device with the purpose to verify the correct implementation of security measures in order to avoid the attacks mentioned in Section 2.2.2. Each tool will be inserted inside a specific category, followed by a brief explanation of its functioning. A summarized version of these associations can be observed in table 3.1, at the end of this section.

3.3.1 Password Cracking Tools

Cain & Abel is a tool that can be used to recover (or *crack*) a variety of passwords using techniques such as network packet sniffing and password hash cracking [PJB+17]. This tool can be useful to test a system against Brute Force/ Weak Authentication, DNS Spoofing/Poisoning and MITM attacks.

THC Hydra is a password cracking tool that can conduct very fast dictionary attacks against over fifty protocols, including Hyper Text Transfer Protocol Secure (HTTPS), HTTP and File Transfer Protocol (FTP). It is a quick and stable network login hacking tool that attempts different password and login combinations on a login page using a dictionary

or brute-force attack [KMJ18]. It is possible to use this tool to test against Brute Force/ Weak Authentication.

John the Ripper is a slow password cracker that was originally designed for Unix. However, as time went by, it became more flexible, and it can now be used on Windows, OpenVMS, and macOS. Its main function is to identify weak Unix passwords. This tool is free and can be used to perform both Brute Force and Dictionary attacks [KMJ18].

3.3.2 Binary Code Analyzers

OllyDbg is an x86 debugger for Windows that focus more on binary code analysis. It traces registers, recognizes procedures, API calls, switches, tables, constants and strings, as well as locates routines from object files and libraries [Oll]. This tool can be used to identify Buffer Overflow attacks.

3.3.3 Penetration Testing Tools

Burp Suite is a software developed in Java by PortSwigger, to perform security tests on Web applications. BurpSuite includes an intercepting proxy, a Web application spider, and a configurable Web application fuzzer among its testing tools. Participants use the spider to map the features of the Web application, then use the intercepting proxy to manually or programmatically observe and alter HTTP requests based on patterns applied by a fuzzer [SWJ13]. This tool can be used to test against a set of vulnerabilities such as Command Injection, CSRF, Path/Directory Traversal File, Reflected XSS, Stored XSS.

The **OWASP Zed Attack Proxy (ZAP)** is a simple integrated penetration testing tool for detecting Web application vulnerabilities. It is designed to be used by people with a wide range of security expertise, making it suitable for newcomers to penetration testing such as developers and practical testers, as well as a valuable addition to the toolbox of an experienced pentester [MK15]. CSRF, Path/Directory Traversal File, Reflected XSS, Stored XSS are some of the vulnerabilities that can be identified by this tool.

3.3.4 Distributed Denial of Service Attack Tools

hping is a Transmission Control Protocol (TCP)/ IP packet assembler/analyzer with a command-line interface based on the Unix command `ping(8)`, but `hping` can do more than just submit Internet Control Message Protocol (ICMP) echo requests. It has a traceroute mode, the ability to send files between covered channels, and many other features. It supports TCP, User Datagram Protocol (UDP), ICMP and raw-IP protocols [Kala]. This tool can be used to simulate DDoS attacks.

HULK is a DDoS attack method that avoids detection by using UserAgent forging. It can start 500 threads to send high-volume HTTP GET FLOOD requests to the target [AAM20]. As stated above, this tool is useful to simulate DDoS attacks.

Goldeneye is a multi-threaded attack tool that uses HTTP GET and POST requests to

initiate DDoS attacks. It does not have IP spoofing capabilities, but it works on all major operating systems, including Windows, Linux, and macOS. This tool can also be used to simulate DDoS attacks.

3.3.5 Network Protocol Analyzers

Wireshark is the most common and widely used network protocol analyzer in the world. It is the *de facto* go-to tool for many commercial and non-profit companies, government departments, and educational institutions because it allows one to see what is happening in the network in terms of traffic. The production of Wireshark thrives thanks to the volunteer contributions of networking experts from all over the world [Wir]. This tool is useful for testing several aspects of network communications.

3.3.6 Spoofing Tools

ARPspoof is a tool for tricking the computer of the victim into sending its traffic to the machine of the attacker or another network gateway by sending false ARP messages to it [O'R]. This tool can be useful to test against DNS poisoning vulnerabilities.

On the Local Area Network (LAN), **DNSspoof** forges responses to arbitrary DNS address/pointer queries. This can be used to get around hostname-based access controls and carry out several MITM attacks. This tool can be used to simulate a DNS poisoning attack to test a system against this vulnerability.

3.3.7 Man-In-The-Middle Attack Tools

Ettercap is an all-in-one solution for man-in-the-middle attacks. It has live connection sniffing, on-the-fly content filtering, and several other features related with this kind of attacks. It can dissect many protocols both actively and passively, and it has a lot of features for network and host analysis [Ett]. DNS poisoning and MITM vulnerabilities can be tested with this tool.

3.3.8 Network Traffic Replayers

Tcpreplay is a collection of free Open Source tools for editing and replaying network traffic that has previously been captured and stored in a supported format. Originally intended to replay malicious traffic patterns to Intrusion Detection/Prevention Systems, it has undergone several changes, including the addition of the ability to replay to Web servers [Tcp]. This tool can be used to test a system for replay vulnerabilities.

3.3.9 SQL Injection Tools

sqlmap is an open-source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over database servers. It comes with a powerful detection engine also. This tool is compatible with a large number of database

management systemsm namely: MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB, Informix, MariaDB, MemSQL, TiDB, CockroachDB, HSQLDB, H2, MonetDB, Apache Derby, Amazon Redshift, Vertica, Mckoi, Presto, Altibase, MimerSQL, CrateDB, Greenplum, Drizzle, Apache Ignite, Cubrid, InterSystems Cache, IRIS, eXtremeDB, FrontBase, Raima Database Manager, YugabyteDB and Virtuoso [GS].

The **SQLi Dumper** tool is a Windows tool that automates the process of detection and exploitation of SQL Injection vulnerabilities. This tool is capable of performing from detection or identification of vulnerabilities to exploitation of said vulnerabilities, automatically.

3.3.10 Cryptographic Protocol Analyzers

A random/pseudo-random generator serving as the source for cryptographic applications is a key aspect of secure software. If the used generator provides low entropy, the entire cryptographic application integrity can be in danger of being compromised.

The tool developed by National Institute of Standards (NIST), entitled **Statistical Test Suite for Random and Pseudo-random Number Generators for Cryptographic Applications**, can assess if a random/pseudo-random number generator is truly unpredictable by applying a total of 15 tests to prove randomness:

1. **The Frequency (Monobit) Test** that determines if the number of zeros in a sequence is similar to the expected in a truly random sequence;
2. **Frequency Test** within a Block to verify if the frequency of ones in an M-bit block is roughly $M/2$, as would be predicted under a randomness assumption;
3. **The Runs Test** to evaluate if the number of runs (uninterrupted sequence of identical numbers) in a sequence is similar to the number of runs expected in a truly random sequence;
4. **Tests for the Longest-Run-of-Ones in a Block** to test the longest run in a given sequence against the longest run expected from a truly random sequence;
5. **The Binary Matrix Rank Test** to see whether the original fixed length substrings of the sequence are linearly dependent;
6. **The Discrete Fourier Transform (Spectral) Test** to detect periodic features in the checked series that would suggest a deviation from the principle of randomness;
7. **The Non-overlapping Template Matching Test** to find generators that output an excessive number of non-periodic patterns;
8. **The Overlapping Template Matching Test** to test the number of occurrences of pre-specified target strings;

Category	Tools
Password Cracking Tools	Cain & Abel, THC Hydra, John the Ripper
Binary Code Analyzers	OllyDbg
Penetration Testing Tools	Burp Suite, OWASP ZAP
Distributed Denial of Service Attack Tools	hping, HULK, Goldeneye
Network Protocol Analyzers	ARPspoof, DNSspoof
Man-In-The-Middle Attack Tools	Ettercap
Network Traffic Replayers	Tcpreplay
SQL Injection Tools	sqlmap, SQLi Dumper
Cryptographic Protocol Analyzers	Statistical Test Suite for Random and Pseudo-random Number Generators for Cryptographic Applications, Proverif

Table 3.1: Correspondence between the studied tools and their categories.

9. **Maurer's "Universal Statistical" Test** to verify if the given sequence can be compressed without loss of information;
10. **The Linear Complexity Test** to determine if the sequence is complex enough to be labelled as random;
11. **The Serial Test**, whose objective is to see if the number of occurrences of the $2^m m-bit$ overlapping patterns is roughly equal to what would be expected from a random sequence;
12. **The Approximate Entropy Test** to determine if the frequency of overlapping blocks of two consecutive/adjacent lengths (m and $m+1$) is similar to the predicted outcome of a random sequence;
13. **The Cumulative Sums (Cusums) Test** to compare the cumulative sum of the partial sequences occurring in the tested sequence against the expected behaviour of that cumulative sum for random sequences;
14. **The Random Excursions Test**, whose objective is to see how the number of visits to a specific state during a loop differs from that of a truly random sequence;
15. **The Random Excursions Variant Test** aims to find out whether the number of visits to different states in the random walk differs from what is predicted.

ProVerif is a well-known formal verification tool that analyzes protocols written in the applied pi-calculus using an extension with function symbols that can describe cryptographic operations. Internally, *ProVerif* transforms protocols into Horn clauses in classical logic, which then resolves to obtain the validity of a cryptographic protocol [BAF08].

This tool can deal with a variety of cryptographic primitives, such as shared- and public-key cryptography (encryption and signatures), hash functions, and Diffie-Hellman key agreements, which can be defined as rewrite rules or equations. The verifier produces false attacks, but when it says that the protocol fulfils a property, the property is indeed fulfilled. A wide range of protocols are terminated by the resolution algorithm under consideration (the so-called *"tagge"* protocols). With this tool, it is possible to prove secrecy, authentication, strong secrecy and equivalences between processes that differ only by terms [Bla].

It should be mentioned that *ProVerif* was studied in depth in the course of this project and that efforts were made to provide useful examples for applying this tool along with a recommendation to use it. This means that the tool presented later in this dissertation provides examples when necessary.

3.4 Security Advising Modules Framework

SAM is the framework used to implement and display the module developed under the scope of this dissertation with a service based architecture. This framework advises the users about various aspects of security during the design phase of the systems. At the moment of writing of this document, SAM includes modules to advise users in the IoT, Cloud and Mobile environments. Most modules of SAM receive inputs in the form of answers to questions related to system and software characteristics, though not necessarily related with security, to then produce security related recommendations.

In simple words, the architecture of this framework can be divided into two parts: the front-end and back-end. The front-end is responsible for handling all the direct human interaction such as registration, login, account management, recommendations, etc., while the back-end is responsible for processing all the information collected in the front-end. It is responsible for session management, module management, answer management, etc.

SAM was developed to be a modular framework, which means that the contributions to this platform are classified as modules. For the IoT, environment there were three complete modules at the time, namely: a module for advising the user on security requirements, one for advising best practices and one for suggesting lightweight cryptographic algorithms. During this project, another two modules were added to the framework, which was also greatly improved along the time.

3.5 Conclusion

In this chapter, a review of the state of the art in Software Testing is presented, focusing in Model-Based Testing, Requirement-Based Testing and Test Automation. It was observed that Software Testing, particularly Test Automation is a topic that has attracted the attention of many researchers. The area of test automation is not yet being widely applied and studied in the context of IoT which could be observed in the work of [VS20] by the incompatibilities found while using popular commercial testing tools to evaluate IoT devices. With IoT thriving and more security flaws being discovered every day, the study of test automation is a critical research area and developments in this filed that would greatly advance security.

It was also observed that the majority of the testing tools require the user to possess a high level of expertise in security and testing. However, not all companies have a security expert in their teams, which may lead to the release of systems without proper testing. With the tool proposed in this dissertation, it will be possible for anyone to test their IoT

systems for security flaws, as long as they possess some level of domain knowledge. This will result in the development of systems with far more quality and security. With this tool, it will be possible to tackle the lack of knowledge of IoT compatible testing tools and the need for security testing experts to develop and execute the tests.

Chapter 4

System Design, Implementation and Demonstration

4.1 Introduction

In this chapter, the functional and non-functional requirements of the tools to be developed in the scope of this dissertation are presented in section 4.2. This is followed by an explanation of the proposed system design, in section 4.3. The tool developed in the scope of this dissertation is presented in section 4.5. This tool was developed to be inserted within a wider framework, SAM, which is also described in section 4.4, including a description of the contributions of the author to the overall development of the framework. The chapter concludes in section 4.6, with an overview of the functionality of the tool developed and its inclusion into the workflow of the framework.

4.2 Requirements

The main objective of ACISM is to highlight potential attacks and suggest penetration tools to simulate them, with the purpose of verifying if the security measures were implemented correctly.

Regarding functional requirements, this tool uses as input the reports produced by the Threat Modeling Solution (TMS) and Lightweight Cryptographic Algorithm Recommendation (LWCAR) modules. The proposed module needs to output a report, listing a set of attacks the system is susceptible to accompanied by a set of IoT compatible tools that can support testing system vulnerability to the suggested attacks. Each one of the recommended tools should, if possible, also be associated with a guide that shows how to use it to simulate the attack with the tool in question.

Considering non-functional requirements, this module is to be developed in Python and integrated within an existent framework that intendeds to aid the user into achieving a secure design of IoT, cloud and mobile systems in various stages of the development. The proposed module should follow a modular approach and maintain functionality even if other modules are added or updated. The tool produced should also be easy to understand and apply in real-life scenarios.

4.3 System Design

The proposed solution will be part of a larger framework, entitled *Security Advising Modules* (SAM), and therefore will interact with the other included components. In this section, the relation between the proposed module and the modules available in SAM is explained.

As figure 4.1 aims to demonstrate, the tool proposed is directly dependent on two other modules: the LWCAR module, which outputs a list of lightweight cryptographic algorithms, taking into consideration the hardware and software specifications of the system under development; and the TMS module, which outputs a list of threats that the system might be vulnerable to. The latter was developed in parallel with the proposed tool. These two modules have some dependencies which should be mentioned here: the LWCAR module is dependent on the Security Requirements Elicitation (SRE) module solely, while the TMS module is dependent on both the SRE module (which outputs a list of security requirements) and the Security Best Practices (SBP) module(which produces a report with good practices). These other modules were developed in the past, but also in the scope of the SECUR IoT ESIGN project. More information on these modules can be found in the works of Samaila *et al.* [SSS+20, SLA+20].

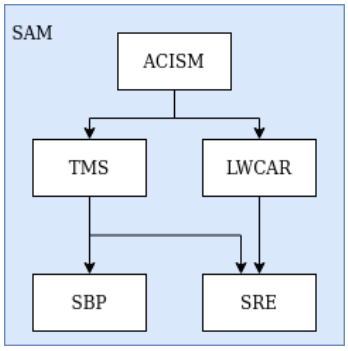

Figure 4.1: Diagram illustrating the direct and indirect dependencies of the ACISM module.

4.4 Deploying Security Components using the SAM Framework

SAM is a framework developed using Flask and React under the SECUR IoT ESIGN project. It has the objective of advising users in the development of secure IoT, cloud and mobile systems during the design phases of these systems.

SAM was developed to be service-based. The architecture of this system, as it can be observed in figure 4.2, is divided into two distinct components: the front-end and the back-end. The front-end is where all the interactions with the user are captured. This component deals with registration, login, user account management, recommendation outputs, etc. The back-end is responsible for processing the information collected in the front-end. This component is where the services are located, such as the module management

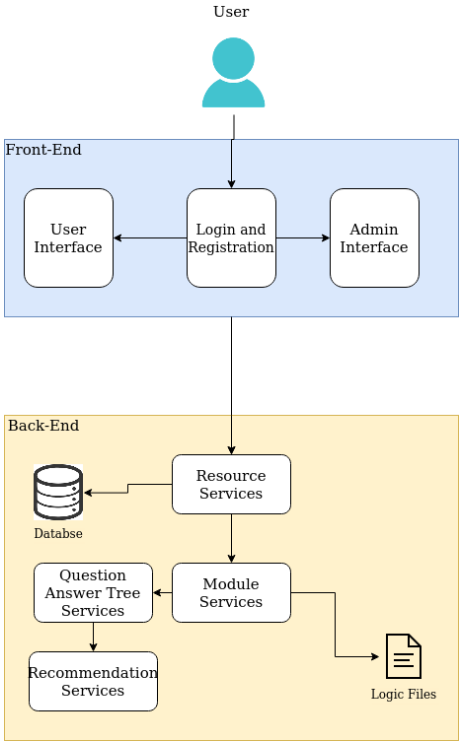

Figure 4.2: Diagram depicting the architecture of the SAM framework.

service, the statistic management service, the question management service, etc.

SAM is a modular framework composed of a set of modules that advise the user in different stages of the development of a system. At the time of writing this dissertation, and for the IoT environment, this framework possesses: (i) a module that makes the elicitation of security requirements; (ii) a module outputting good programming and implementation practices; and (iii) one that recommends secure lightweight algorithms appropriate for the type of system and hardware used.

In SAM, the modules, as we can see in figure 4.3, can be of two different types: regular modules, which are composed of a set of questions and a set of several pre-defined answers that the user selects to obtain a specific report, while plugin modules have no direct input from the user and instead use the answers provided by the user on other modules (dependencies) to produce a report. Inside of what is defined as a *traditional module*, there are two distinct categories: modules where each answer is associated with a specific output, defined as *static modules*; and *dynamic modules* that are associated with a logic file that processes the answers given by the user as a whole and produce the output.

The user needs to first have admin privileges to successfully insert a module into SAM. After a successful login into the admin account, the user then needs to select the `Resources` and then `Modules` tab. The respective interface will then be presented and, from there, the user needs to click on the `Add new module` button. A pop-up, exactly like the one shown in figure 4.4, will be shown to the user. To complete the insertion of the new module, the user simply needs to fill in its name, the abbreviation of the name, the display name, and then select what type of module it is. If the module is a *traditional module*, the user needs

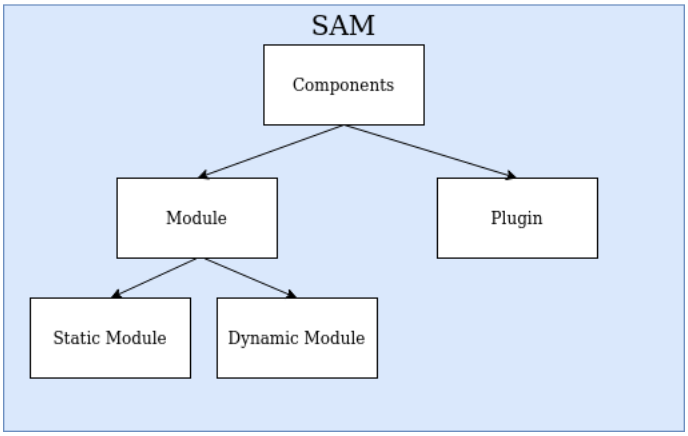

Figure 4.3: Diagram showing the representation of the different types of components present in SAM.

to click on the `Add questions and answers` button, and build the respective question-answer tree. On the one hand, if it is a `static module`, the association between each answer and respective output must be defined by clicking on the `Link recommendation` button. On the other, if the module is a plugin or a *dynamic module*, it is required to associate a logic file by clicking on the `Logic` button. It is also possible to link dependencies by clicking on the `Link dependencies` button and selecting them. Dependencies will be mentioned below in this chapter.

The contributions of the author to this framework, besides the new module developed as part of the research described in this dissertation, were the development of the SRE module (as a project elaborated for the attainment of a bachelor's degree in Computer Science and Engineering), the adaptation of a module that focuses on outputting a guide of security best practices for cloud and mobile devices to work on SAM, and the application of several corrections and enhancements to stabilize the platform. The author was thus one of the main developers of SAM.

4.5 Plugin Implementation Details

The tool developed during this dissertation, entitled *Assessing Correct Integration of Security Mechanisms* module, is considered a plugin in the SAM framework since it has no direct user input. To generate the output, this module uses the recommendations from two other modules:

- **Threat Modeling Solution (TMS)** – the ACISM module was developed in parallel with the TMS module, developed by another Master's student. TMS takes as dependencies the SRE module and the Security Best Practice Guidelines module and it produces a set of possible vulnerabilities that the system in question can suffer from;
- **Lightweight Cryptographic Algorithm Recommendation (LWCAR)** – in this module, as previously mentioned, after the questionnaire, a set of Lightweight

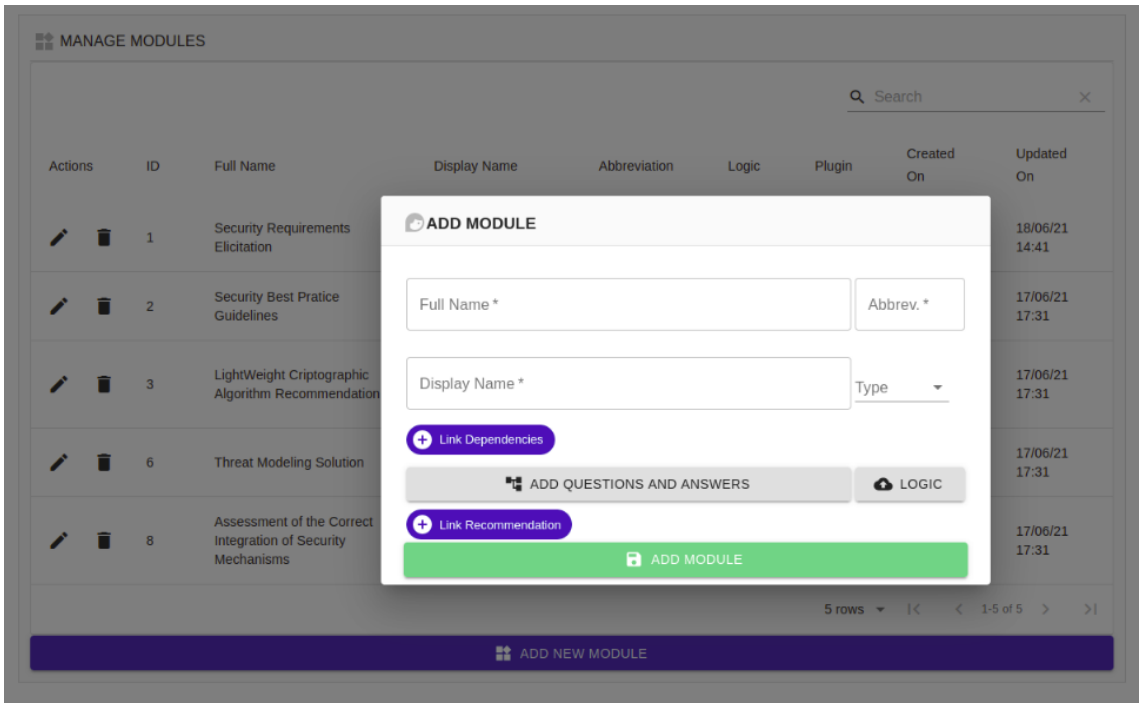

Figure 4.4: Screenshot showing how a new module can be added to SAM.

Cryptographic Algorithms that should be applied to the system are outputted [SSS+20], which are also fed into ACISM.

To initiate the ACISM module the user simply needs to click on the button that says `Let's talk about Assessment of the Integration of Security` when entering SAM. This will trigger the Python script containing the logic of this module. With the recommendations from these two modules (TMS and LWCAR), ACISM can produce a report with a list of potential attacks that the system is susceptible to, accompanied by tools that can be used to simulate those attacks and a simple tutorial for the installation and use of each of these simulation tools. With these tools, the tester can perform penetration tests to verify if the system under development has correctly integrated the security mechanisms that mitigate the vulnerabilities outputted by the TMS Module.

Each produced recommendation appears in the generated pop-up, specialized in presenting the report. Each recommendation is composed by their respective name and a list of the penetration tools suggested, as can be seen in figure 4.5. If the user wishes more information about these tools and a brief tutorial for installation and usage, he/she needs to click on the `Read More` button, which launches a markdown file with the referred content. Simultaneously, the user will be prompted to download the reports produced. If the user agrees, the main report will be converted to Portable Document Format (PDF). The report and all guide files will be then downloaded to the computer of the user. To make this association, the logic will query the database that stores all the recommendations given in one session, and collect all the names of the threats outputted after the completion of the TMS module, so as to make a direct association between the threat name and an attack that can be done by exploiting it. Furthermore, each attack will be associated with a set of

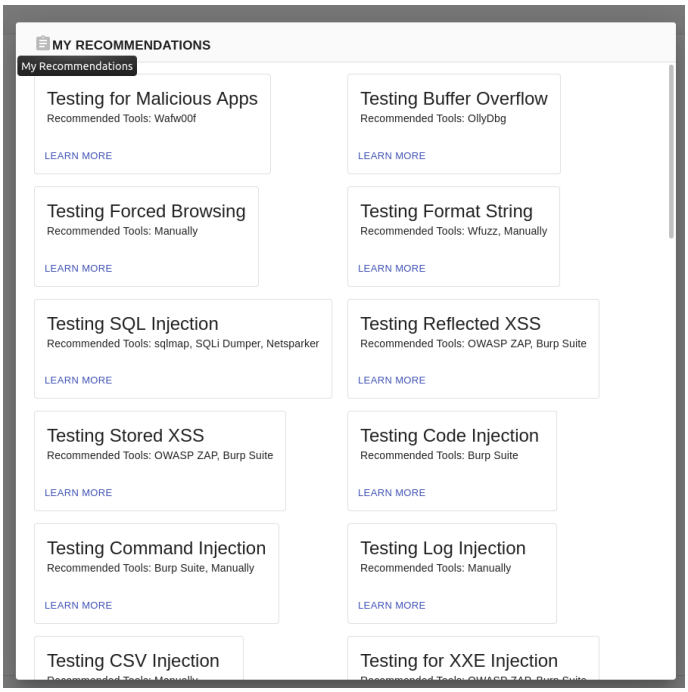

Figure 4.5: Screenshot with an example of a report produced by the ACISM module.

penetration tools compatible with the IoT environment.

To map the association between the answers and the provided report, it is required to implement a so-called *logic* file (implemented in python). Even-tough each module is different and tackles its own problem, there are functions common to every plugin, as shown in listing 4.5: function `get_dependency_recommendation` returns a list of recommendations of the last session for each one of the dependencies of the plugin in question; function `get_recommendation_id` returns the database identifier associated with each given recommendation; function `get_recommendation_content` returns, specifically, the name/content of each recommendation.

Listing 4.1: Functions common to all plugins.

```
...
def get_dependency_recommendations(session, dependency_number):
...
def get_recommendation_id(recommendations, recommendation_name):
...
def get_recommendation_content(recommendations):
...
```

Regarding the code used to create the logic behind this module, listing 4.5 presents the header (prototypes) of the function used to perform this task, along with some pseudo-code to exemplify the associations made in this logic file. As can be seen, a critical part of this work was on researching the literature and state-of-the-art and devising this map.

Listing 4.2: Header of the function specifically used in the ACISM module.

```
...
def get_recommendations(threats, ciphers):

    if "vulnerabilityX" in threats :
        recommendations.append["Testing for attack x"]

    if "vulnerabilityY" in threats :
        recommendations.append["Testing for attack y"]
...
```

This function receives as input all the information about the threats recommended by TMS module to associate the vulnerability with the correct testing file and tools. It also receives the output from the LWCAR module (ciphers) with the intention to use the lightweight cryptographic algorithms recommended to output (if applicable) a specific `ProVerif` file for each.

4.6 User Interaction and Flow

When the user successfully logs in into SAM, after filling the necessary fields with the respective email and password (figure 4.6), they are presented with the modules available on the platform. When choosing the ACISM module (figure 4.7) the system will verify if the user has already completed the dependencies necessary. If the user has not answered the LWCAR Module and the TMS Module, SAM will ask the user to first complete the mentioned modules (figure 4.8). ACISM can only be executed once this pre-requisite is met.

The vulnerabilities inputted into this module logic were taken from Common Weakness Enumeration and, when possible, the ACISM module will associate the vulnerabilities with attacks and tools to simulate said attacks. The association between vulnerability and attack was made with the use of the correspondence of said vulnerability and the attack patterns associated with it in Common Attack Pattern Enumeration and Classification and OWASP Attacks.

Correspondence between each testing file, vulnerability and recommended tools are provided in table 4.1. For example, the vulnerability **CWE-79** – *Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')* is associated with XSS attacks; because of this, the tool will output two different recommendations, one for Reflected XSS attacks and one for Stored XSS attacks. Each one of these recommendations will describe the attack and suggest tools that aid in performing penetration tests of the attack. A brief and simple tutorial on how to use and install the recommended tools is also presented. For both these attacks, the tools recommended are OWASP ZAP and Burp Suite. On the other hand, the vulnerability **CWE-404** – *Improper Resource Shutdown or Release* is directly associated with Denial-of-Service (DoS)/DDoS and, in this case, the tool will recommend the use of `hping3`, HULK, GoldenEye and Wireshark to be able to

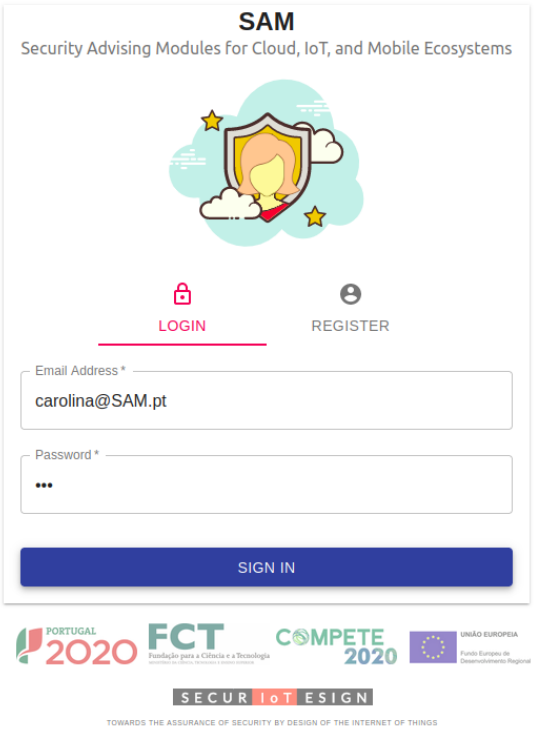

Figure 4.6: Screenshot showing a user filling in the fields necessary to sign-in.

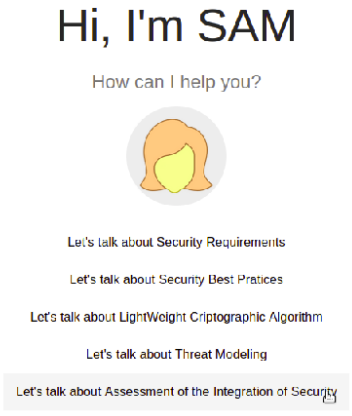

Figure 4.7: Screenshot showing a user selecting the ACISM module.

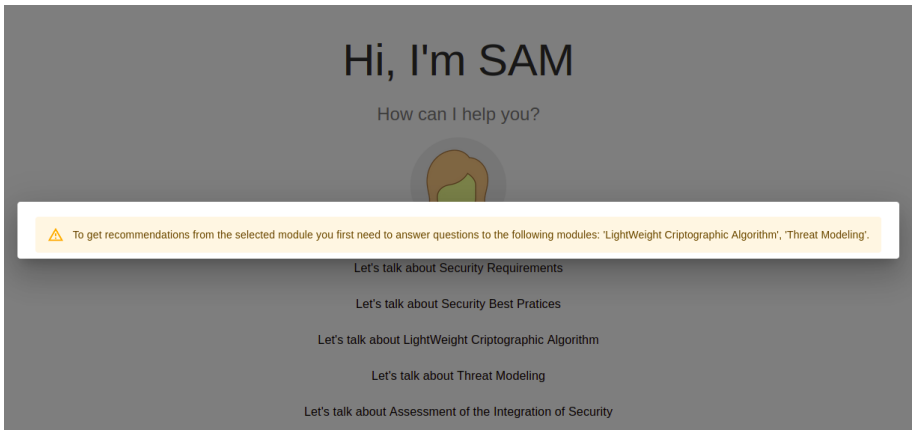

Figure 4.8: Screenshot exemplifying how the platform requests for the user to complete the dependencies of the ACISM module before trying to execute it.

observe the traffic generated by the other tools. The vulnerability **CWE-269** – *Improper Privilege Managemen* is associated with Access Control problems and it can lead to CSRF attacks. For these reasons, the tool will recommend Testing the Access Control of the system with the OWASP ZAP scanner and test the system against CSRF attacks with OWASP ZAP and Burp Suite. If the vulnerability **CWE-327** – *Use of a Broken or Risky Cryptographic Algorithm* is possible in the system under analysis, the tool will advise testing the implementation of the cryptographic protocols. In this scenario, the module will suggest the use of the tool *ProVerif*, but since this is a formal cryptographic protocol verifier and the user may not have the theoretical knowledge to implement correctly the protocols following the syntax specific to *ProVerif*, the ACISM module will check the suggested lightweight cryptographic algorithms from the LWCAR module and it will, also, output a *ProVerif* script that illustrates the type of protocols suggested such as HASH, MAC, HMAC, etc.

Testing File	CWE Identifier	Recommended Tools
Testing the Access Control	269, 306, 1220, 1224, 1242, 1244, 1262, 1267, 1268, 1280, 1311, 1326	OWASP ZAP
Testing if Assumed-Immutable Data is Stored in Writable Memory	1282	Manually
Testing Brute Force	287, 1220, 1224, 1244, 1273	Cain & Abel, THC Hydra, John the Ripper
Testing Buffer Overflow	787, 119, 573, 170, 190	OllyDbg
Code Injection	74, 573, 416, 476, 913, 434, 469, 479, 502, 22, 94	Burp Suite
Code Signing	1326	Manually
Command Injection	74, 668, 476, 434,22, 78	Burp Suite, Manually
Assessment of Cryptographic Protocols	327	ProVerif

CSRF	668, 269, 306, 352, 732	OWASP ZAP, Burp Suite
CSV Injection	74	Manually
DDoS/DoS Attacks	668, 416, 476, 4040, 705, 913, 625, 502, 170, 252, 248, 400, 22	hping3, HULK, Golden-Eye, Wireshark
DNS Spoofing/Poisoning	441	Arpspoof & DNSspoof, Cain & Abel, Ettercap, Wireshark
Error Handling	544	Manually
Testing Expired Domain	287	Manually
Firewall Implementation	Always Recommended	Wafw00f
Firmware Cracking	1278	IoT Inspector, Binwalk, QEMU, FIRMADYNE, Firmware Analysis Toolkit (FAT)
Forced Browsing	787, 732, 1242	Manually
Format String	119, 74, 668	Wfuzz, Manually
Testing for Hardware Overheating	1338	Manually
Hardware Fault Injection	1319	Manually
Heartbleed bug	125	Openssl
Log Injection	74	Manually
Testing Man in the Middle Attacks	287, 515, 522	Bettercap, Ettercap, Cain & Abel
Testing Path Traversal	200, 668, 913, 434, 22	OWASP ZAP, Burp Suite
Testing Race Conditions	479, 1298	CHESS
Testing Random and Pseudo-random Number Generators for Cryptographic Applications	330	Statistical Test Suite for Random and Pseudo-random Number Generators for Cryptographic Applications (NIST)
Testing Reflected/Stored XSS	119, 74, 20, 125, 625, 79, 1242,	OWASP ZAP, Burp Suite
Remote File Inclusion	98	Manually
Replay Attack	330, 732	Tcpreplay, Wireshark
Avoid Incorrect Selection of Fuse Values	1253	Manually
Shared Resource Manipulation	1331	OWASP ZAP
SQL Injection	119, 74, 20, 125, 611, 625, 89	sqlmap, SQLi Dumper, Netsparker

Insecure Transport	200	Testssl, OWASP O-Saft
Testing for Insecure Direct Object References	862	Manually
XML external entity (XXE) Injection	20, 611	OWASP ZAP, Burp Suite
SEU Sensitivity	1261	Particle accelerator facility, Chamber shielded for radiation

Table 4.1: Correspondence between each vulnerability and the respective tests.

4.7 Conclusion

This chapter presented an analysis of the requirements that the module developed aims to meet and introduced the high-level system design. This chapter intends to provide the user with a better understanding of the tool to be designed. The details of the implementation of the ACISM module were presented and discussed. It contains a brief introduction of the SAM platform (in which this module is inserted), followed by a brief explanation of the ACISM module, its functioning and respective dependencies. Finally, an overview of how the user interacts with the module was provided together with associations made between each threat and respective output from the ACISM module.

Chapter 5

Testing and Module Validation

5.1 Introduction

In this chapter, the steps taken to test the developed module are described. The process of deploying SAM in order to make it accessible to the test subjects is described in section 5.2, followed by the procedure used to test the platform in section 5.3, which is divided into two phases: automatic tests on the platform (to guarantee that it is ready to be presented to the public); and user tests. To conclude, the results of the user tests will be presented and briefly discussed.

5.2 SAM Deployment

To facilitate the testing phase of the developed module, test subjects should have easy access to ACISM without any kind of installation or setup needed. This is achieved by hosting SAM in a server with a public domain.

A server was set up inside Universidade da Beira Interior (UBI). The server is running CentOS at the time of writing of this dissertation. To deploy SAM, Nginx was required as an HTTP server and thus the front-end was deployed on Nginx. The SAM-API was first deployed on *gunicorn* to handle the requests. To maintain the *guincorn* server running in the background and automatically restarting, when needed, a custom purpose Linux service was created. The service, described in a brief manner, defines the location of the project and executes the command to start the server. Next, Nginx was configured to redirect the request to the correct location. After the installation of this server, the only steps left were configuring on which ports the server should be listening. The server was set to always listen to port 80 when receiving requests from the defined Uniform Resource Locator (URL). If the server receives any request started with `api/`, then it should redirect to port 81, from which the request would be further redirected to the *gunicorn* service.

The final step was to generate a valid X.509 certificate to provide HTTPS communication to the users. To generate the certificate, *certbot*, one of the certificate generators approved by *Let'sencrypt*, was used. SAM and all its functionalities can be accessed in `https://securiotesign.di.ubi.pt/`.

5.3 Testing

This section presents the procedure taken to successfully validate the usefulness and correct functioning of the SAM platform and the ACISM module. Firstly, the process of automatically testing the modules of SAM will be discussed. Afterwards, the presentation of the opinions of the test subjects about the developed module will be included.

5.3.1 Automatic Testing

Before advancing to user testing, it was necessary to ensure that the system had no evident (implementation) flaws in the modules to be tested. To guarantee the correct execution of the implemented modules, automatic testing was performed using *Selenium*. The testing was then divided amongst team members to be more efficient (the development team of the platform had five team members at the time these experiments were performed), and the modules tested by the author were the SRE and the SBP modules.

Selenium [Hug] is a portable framework that is used to achieve test automation of Web Apps. In the context of this work, it was used to automate the testing of the modules made available within SAM. To create a comprehensive set of tests, various scripts were created to cover a wide combination of possible answers for each of the tested modules. Recall that other modules of the security framework use a set of answers to specifically crafted questions as inputs.

After several iterations and improvements, and once all of the executed tests were passed, there were no errors found inside the modules and therefore the system was considered ready for user testing.

5.3.2 User Testing

In order to guide the users through their experience while testing tool, a PDF file, available on Appendix A, with the descriptions of possible IoT scenarios was created and hosted on the server (and provided to those users). The purpose of this file is for the user to choose one of the available scenarios and use this scenario as guidance to follow through with the completion of the modules. There were seven scenarios described in this document, one for each big IoT application domain:

- **Perishable Cargo and Transportation Monitoring** – this example implies that perishable goods and their transit methods are being monitored (in this specific case, a truck). The system is made up of sensors that monitor the cargo in terms of humidity, temperature and positioning, as well as an On-board diagnostics (ODB) scanner that connects to the Electronic Control Unit (ECU) of the truck and feeds data to the main sensors, as well as a variety of other sensors that monitor most components, fluids, tire pressures, and GPS tracking. All of the truck sensors will communicate with a hub via 4G technology, which will send all of the data to a cloud service in real-time, where it will be analyzed and monitored by the end-user.

Bluetooth will be used to communicate between the sensors, ODB scanner, and hub. The cargo sensors can immediately communicate their condition to the infrastructure since each container has its own 4G connection, allowing cargo to be monitored at all times, even while in storage or changing modes of transportation;

- **Optimization of Infrastructure and Energy Usage for Electric Vehicles Home Charging** – This scenario depicts the usage of IoT in an electric grid to ensure that electric vehicle charging at individual houses does not put undue strain on the power infrastructure. This will ensure that charging occurs primarily during periods of low demand and, where possible, with the use of predominantly renewable energy sources. This will result in increased efficiency, decreased total usage, and reduce costs. Wall chargers will need to interact with the infrastructure, which will include smart metering across the grid, from production to distribution and consumption, in order for this system to operate;
- **Smart Lamp** – These lamps need to be connected to the home wireless network, and users can manage them by installing the software, creating an account, and connecting their mobile phones to the same network. The router will function as a bridge between the user and the lamp. The user issues orders to the lights via the app, and these commands are delivered to a coordinator via wireless communication. The coordinator decides which lights will be impacted by the command and sends it to the appropriate lamps. This command is sent to the affected lights through a microprocessor, which then performs the desired operation;
- **Pacemaker with Mobile Device Interface** – This scenario depicts the employment of a smart pacemaker in conjunction with a mobile interface that allows a user to obtain heart-related data. The communication will be done using an external device that gathers information from the pacemaker when it is close by (near field) and connects to a phone through Bluetooth. The user simply has to launch the app and log in to view the information after it has been synchronized. Sensors on the pacemaker will collect data related with the health of the patient as well as the working conditions of said device. Doctors will also be able to obtain information from the patient through the app. Notifications may also be generated using Wi-Fi technology to alert a local hospital if the condition of the user deteriorates;
- **Smart Irrigation System** – This scenario depicts a smart irrigation system that is used in agriculture to conserve resources and increase crop quality. Sensing, processing, front-end, actuation, and persistence are the five components of this IoT system. Several wireless nodes with soil moisture and temperature sensors make up the sensing unit. A receiver node is serially linked to a Raspberry Pi to form the processing unit. The Message Queuing Telemetry Transport (MQTT) communication protocol is used to send data from the receiver to the gateway. To allow the user to monitor data in real-time, a graphical Web interface and a mobile application are created. The user is notified via a smartphone notification or through the Web interface when the moisture levels in the soil reach a specific threshold level. In order

to save water and energy, quick action may be made to regulate the engine. Finally, the persistence unit stores data straight from the publisher and may be accessed through the Web or a mobile app;

- **Generic Smartwatch** – This case portrays a smartwatch, which is one of the most prevalent IoT devices. This gadget will connect through Bluetooth to the smartphone of the user and will be linked with a specific app that the user will need to download to his or her smartphone. The smartwatch will be able to receive user-relevant events, such as text messages, emails, or phone calls, after a successful pairing. Any additional notifications may be activated to appear on the wristwatch as well. It is also feasible for the user and the wristwatch to communicate, such as answering calls, reading or responding to messages or emails;
- **Smart Factory** – This scenario depicts a smart factory that employs a variety of technologies to cut costs, downtime, and waste. The factory floor is made up of linked machinery and gadgets that allow data to flow between humans and machines. One example is a plastic-producing machine with an integrated display that is used to track temperature, air pressure, plastic consistency, and time spent working. Even though this machine generates the data, it requires sensors to gather it. These sensors are built into the machine itself, and their data is shown on the gadgets. The gadgets previously mentioned are solely utilized to collect data from the manufacturing floor. This data is transmitted to the Cloud or to owned servers, where it is analyzed and used to offer the intelligence part of the smart factory. It is critical to use Big Data Analytics on the server/Cloud to ensure cost savings, downtime reduction, and waste minimization. Gates are used in the smart factory to communicate across numerous equipment, gadgets, and sensors. It also contains gateways that connect to the Internet and transfer data to the Cloud, when needed.

Starting from the scenarios described above, we presented SAM to 17 testers. Five of the testers were employees of a company located in Fundão named *Fruition*, to whom SAM was presented to obtain their feedback on this platform. To make the test subjects assessment of the tool easier to collect and analyze, a questionnaire was created in Google Forms. This questionnaire had a total of five questions addressing the clarity of the reports, the adequacy of the tools, user awareness, helpfulness and the sense of utility of the modules. None of the participants was a security specialist, and some of them were undergraduation of masters students on computer science and engineering. The majority of the participants were men.

5.3.2.1 Results and Conclusions

The module ACISM was evaluated by 17 participants, all with some expertise in the field of computer science. Each one of the participants answered to all five questions in the form. Bellow, the results of that evaluation will be presented and discussed.

The first question addressing the ACISM module asked the user to evaluate the clarity

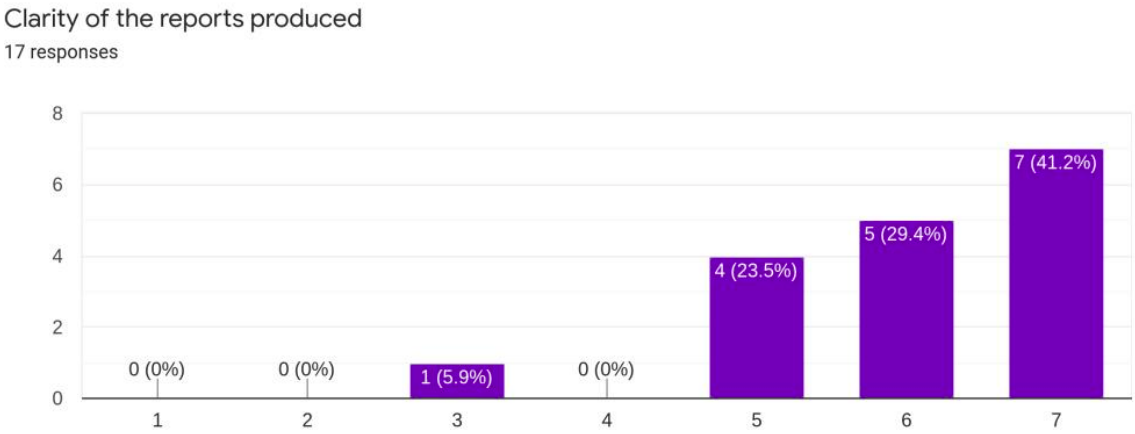

Figure 5.1: Bar chart representing the test results regarding the clarity of the tool.

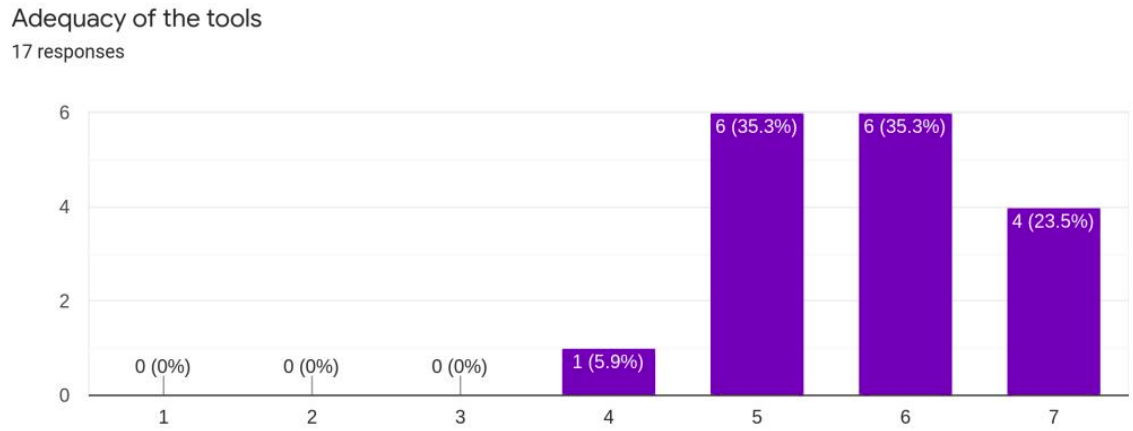

Figure 5.2: Bar chart representing the test results regarding the adequacy of the tools recommended.

of the reports that it produces. The test subjects could evaluate this topic with a number ranging from 1 (Extremely Unclear) to 7 (Extremely Clear). From the analysis of the results presented in figure 5.1 it is possible to conclude that the majority of respondents (71%) found the reports to be very clear or extremely clear, represented by 6 and 7 respectively. Only 6% of respondents found the reports not that clear.

The next question was focused on the adequacy of the recommended testing tools. Once again, the users could evaluate this aspect in a scale ranging from 1 (Extremely Inadequate) to 7 (Extremely Adequate). It can be concluded, namely by analysing figure 5.2, that most of the test subjects (59%) found the recommendations to be adequate, represented by the value 5 or very adequate, given by the value 6.

Next, the test subjects were asked about their awareness of the testing tools that were recommended in their run of the module. In other words, they were asked if they knew the recommended tools. The range of the evaluation was between 1 (Not Aware) and 7 (Extremely Aware). By observing the chart in figure 5.3, it can be stated that only 47% of the test subjects knew well (defined by the value 5) or very well (value 6) those tools. This means that almost half of the participants did not have knowledge of all the testing tools outputted. In a real-life scenario, this means that those 47% could possibly miss some

threats to their systems due to incomplete security testing.

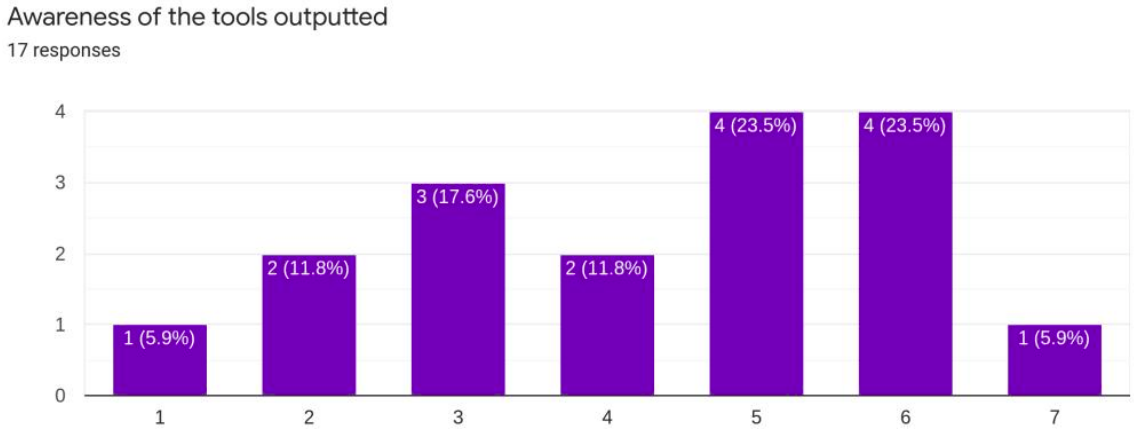

Figure 5.3: Bar chart representing the test results regarding the user awareness of the tools recommended.

The test subjects were then asked if they found the individual guides provided for each tool to be helpful for the first interaction with the tool. The subjects were allowed to evaluate this topic with values ranging from 1 (Not Helpful) to 7 (Extremely Helpful). Figure 5.4 summarizes the obtained results for this topics and it is possible to conclude that 71% of the respondents found the guides to be very helpful or extremely helpful, respectively represented by the values 6 and 7.

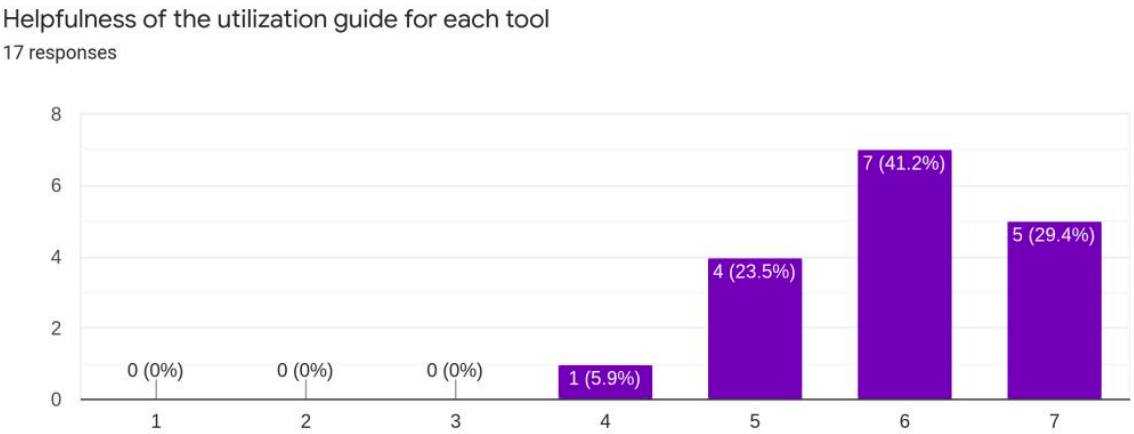

Figure 5.4: Bar chart representing the test results regarding the helpfulness of the individual guide provided for each tool.

Finally, the users were asked to evaluate their perception of the utility of the module in aiding through the testing phase of an IoT device (perception of utility of the tool). The possible evaluations could range from 1 (Not Useful) to 7 (Extremely Useful). From the perspective of the test subjects, and as shown in figure 5.5, it is possible to assert that 71% of users found the module developed to be very or extremely useful to assist in the task of security testing, represented bu the values 6 and 7 respectively.

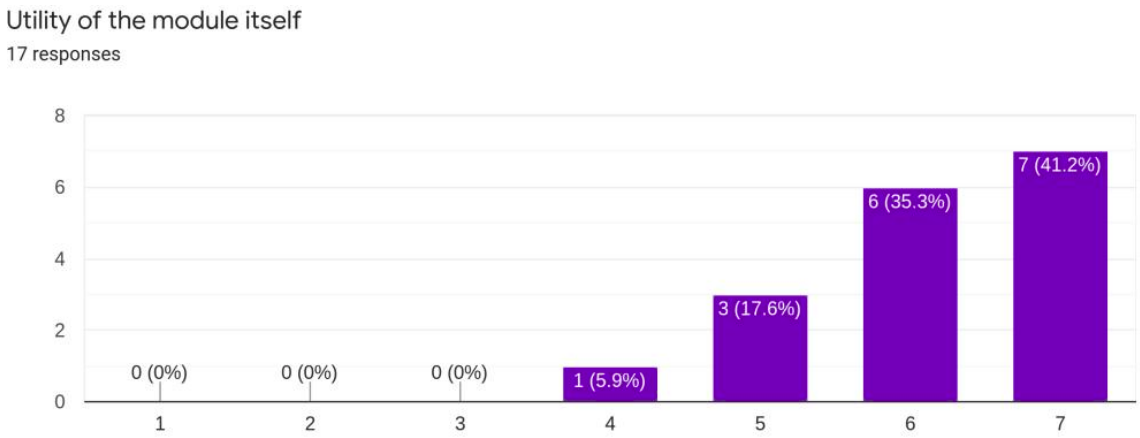

Figure 5.5: Bar chart representing the test results regarding the utility of the module.

After interpreting the results of the questionnaire presented to the test subjects, it is possible to conclude that the module is producing reports easy to understand, with suitable tools. The test subjects also found that the individual guides were very useful as a first introduction to the tools presented. Moreover, a large majority also found the module in question to be very, represented by the value 6, or extremely useful, represented by the value 7, in aiding the security test phase of an IoT device.

5.4 Conclusion

This chapter presents the steps taken in order to prepare SAM and the ACISM module for user testing. Firstly, the platform was hosted in a server located inside UBI, to allow for an easy access. Next, the various modules of the platform were automatically tested using *Selenium* to be sure that the tool was behaving as expected. The platform and all the modules worked as expected and, therefore, were considered to be ready for user testing.

Finally, the tool was tested by 17 users with computer science background. Specifically, the users found the ACISM module capable of producing easy to understand reports, the recommended testing tools were found to be adequate for the suggested tests, and the individual guides provided with each recommendation were found to be very helpful. A large majority of the test subjects has found the ACISM to be very or extremely useful for aiding in security testing of IoT devices.

Chapter 6

Conclusion and Future Work

The final chapter of this dissertation will present the main conclusions of the work developed. In section 6.2, some additional contributions made during the course of this dissertation will be described. Finally, section 6.3 will present the future work to be carried out, in order to improve the tool developed in the scope of this dissertation.

6.1 Main Conclusions

The objective of the work presented in this dissertation was to research and develop a tool that, according to the type of potential attacks that a system may suffer, generates a report advising the user on which security tests are required and which tools can test the system against a possible vulnerability. Chapter 2 provided the readers with some background knowledge in the area of IoT, IoT vulnerabilities and testing. In chapter 3, some relevant works in the area of testing were presented, followed by a set of tools used in security testing. After this research, it was concluded that efforts are being made to automate the tasks of testing but the majority of the existing tools require some level of knowledge in testing. Since not every company has the possibility of hiring a security expert, a tool that advised users on how to test their system for security flaws and what resources were available to aid in that task was needed. This further motivated the development of the ACISM module.

Chapter 4 presents the functional and non-functional requirements of the proposed tool, followed by an overview of the system and its architecture. Next a brief explanation of SAM, the platform in which the developed module was embedded. Within SAM, ACISM is classified as a plugin module since it does not present the user with a questionnaire and instead derives its inputs from answers submitted in other SAM modules. A brief introduction to the ACISM module was provided including its dependencies: to produce the report the ACISM module only needs to process the recommendations given by the TMS module and the LWCAR module. The flow behind the interactions of the user with this module and its dependencies was then presented.

Chapter 5 presents the steps taken to host SAM in a public server located in UBI. Afterwards, the platform was tested using Selenium to verify if it was running as expected. Since all the tests were successful, the tool and all the IoT modules it possessed were subjected to user testing. The tool was tested by 17 users with some experience in computer science, and the results were very positive towards the developed module, as the majority of the test subjects found the tool to be very or extremely useful in assisting security testing in the IoT domain. The remaining input collected from the test subjects was also very

positive and the detailed results are included in chapter 5.

In conclusion, and taking into account all that was presented above, it is possible to assert that all the objectives proposed for this dissertation project were successfully achieved.

6.2 Contributions and Achievements

In this section, some extra contributions and achievements done in the scope of this dissertation are discussed.

6.2.1 Tool for Production of Test Values for Conditional Coverage Testing

Another important step in developing high quality software, besides guaranteeing security, is to assure that the system behaves as expected. If the code of the system is not verified against the requirements proposed, errors may appear, causing the malfunctioning of the system and displeasing its users. However, code checking is a very daunting and time-consuming task and, as such, the need for automation in this area is also of utmost importance so as to produce quality software.

During this dissertation, a Python tool that analyzes the code of the user and searches for the conditions in the code was also developed. This tool then analyzes each condition and produces boundary values to test that condition. After the generation of test values for each condition, the tool creates a Python file in which it will write the tests generated in a syntax compatible with `pytest`. With this tool, the users will be able to perform conditional coverage testing in their code and verify the compliance of the code to the requirements established.

At the time of writting of this dissertation, this tool only has the capability of testing `if` conditions and it supports generating values for `int`, `float` and `strings`. It also supports the more common logical conditions such as: `<`, `>`, `==`, `!=`, `<=`, `>=`. Specifically, for strings it supports `in`, `len()`, `.isdigit()`, `.isdecimal()`, `.isidentifier()`, `.islower()`, `.isupper()`, `.isnumeric()`, `.isprintable()`. After the generation of the test files, the user can run the command `pytest --verbose`, `pytest` will automatically test the conditions with the values generated and will produce a list of all the tests that have been done with the respective results (PASSED, FAILED). Afterwards, the tester should analyse the results produced and evaluate if the conditions are working as planned. Condition testing can be automated via the usage of this tool and, therefore, it saves time and resources during the test phase of a system.

6.3 Future Work

During the development and testing of this tool, several improvements that would make this tool even more useful for developers were identified. For example, in response to

feedback from one of the test subjects (who noted that the guide files for each attack were in PDF format instead of *markdown*), all the guides should be converted to this format.

In the current version, the downloadable PDF report does not contain the same information included in the individual guides. These guides are also downloaded, and below each recommendation the name of the corresponding file is presented. This means that the user needs to manually find the file and open it to access the desired information. To make this process a little easier, the content of each individual guide should be automatically inserted into the PDF file that contains the recommendations.

Finally, even though this module outputs tools for each attack the system could be susceptible to, the installation and usage of the tools suggested is not automated. At the time of writing of this document, the user needs to manually install each tool and follow the usage guide to verify if the system under test is vulnerable to that specific threat. A significant improvement to the SAM platform would be to develop the means that automatically installs and runs the exemplified tests using the suggested tools. These means would potentially comprise a new module that takes the output given by the ACISM module and asks the user some extra information (e.g., about the system in which the tool is to be installed).

Bibliography

[AAM20] E. A. Asonye, I. Anwuna, and S. M. Musa. Securing ZigBee IoT Network Against HULK Distributed Denial of Service Attack. In *2020 IEEE 17th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET)*, pages 156–162, 2020. 23

[ACSC20] T. Alladi, V. Chamola, B. Sikdar, and K. R. Choo. Consumer IoT: Security Vulnerability Case Studies and Solutions. *IEEE Consumer Electronics Magazine*, 9(2):17–25, 2020. xix, 8, 10

[AHDR18] A. Aniculaesei, F. Howar, P. Denecke, and A. Rausch. Automated generation of requirements-based test cases for an adaptive cruise control system. In *2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests (VST)*, pages 11–15, 2018. ix, 18

[AYL+21] Y. An, F. R. Yu, J. Li, J. Chen, and V. C. M. Leung. Edge Intelligence (EI)-Enabled HTTP Anomaly Detection Framework for the Internet of Things (IoT). *IEEE Internet of Things Journal*, 8(5):3554–3566, 2021. xix, 9, 10

[BAF08] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *The Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008. Algebraic Process Calculi. The First Twenty Five Years and Beyond. III. Available from: `https://www.sciencedirect.com/science/article/pii/S1567832607000549`. 26

[Bla] B. Blanchet. ProVerif: Cryptographic protocol verifier in the formal model. [Online] `https://prosecco.gforge.inria.fr/personal/bblanche/proverif/` [Cited 31 March 2021]. 26

[CL18] G. Chu and A. Lisitsa. Penetration Testing for Internet of Things and Its Automation. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1479–1484, 2018. 5

[Dat18] M. Data. The Defense Against ARP Spoofing Attack Using Semi-Static ARP Cache Table. In *2018 International Conference on Sustainable Information Engineering and Technology (SIET)*, pages 206–210, 2018. xix, 8, 10

[DRG08] V. Darmaillacq, J. Richier, and R. Groz. Test generation and execution for security rules in temporal logic. In *2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, pages 252–259, 2008. ix, 19

[DV17] J. Deogirikar and A. Vidhate. Security attacks in IoT: A survey. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 32–37, 2017. xix, 7, 8, 9, 10

[ECG+15] J. S. Eo, H. R. Choi, R. Gao, S. Lee, and W. E. Wong. Case Study of Requirements-Based Test Case Generation on an Automotive Domain. In *2015 IEEE International Conference on Software Quality, Reliability and Security - Companion*, pages 210–215, 2015. ix, 17

[EMB15] R. Elghondakly, S. Moussa, and N. Badr. Waterfall and agile requirements-based model for automated test cases generation. In *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 607–612, 2015. 16

[Ett] Ettercap. Ettercap Home Page. [Online] `https://www.ettercap-project.org/index.html` [Cited 10 March 2021]. 24

[FJR+18] D. Freudenstein, M. Junker, J. Radduenz, S. Eder, and B. Hauptmann. Automated Test-Design from Requirements - The Specmate Tool. In *2018 IEEE/ACM 5th International Workshop on Requirements Engineering and Testing (RET)*, pages 5–8, 2018. ix, 17

[FMCS20] P. Ferrara, A. K. Mandal, A. Cortesi, and F. Spoto. Static analysis for discovering IoT vulnerabilities. In *International Journal on Software Tools for Technology Transfer*, number 23, page 71–88, 2020. 7

[GHM18] D. Gafurov, A. E. Hurum, and M. Markman. Achieving Test Automation with Testers without Coding Skills: An Industrial Report. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 749–756, 2018. ix, 20

[GS] B. Damele A. G. and M. Stampar. sqlmap Automatic SQL injection and database takeover tool. [Online] `http://sqlmap.org/` [Cited 10 March 2021]. 25

[HAS+20] J. Hwang, A. Aziz, N. Sung, A. Ahmad, F. Le Gall, and J. Song. AUTOCON-IoT: Automated and Scalable Online Conformance Testing for IoT Applications. *IEEE Access*, 8:43111–43121, 2020. ix, 22

[HEN+13] Y. Huang, M. Esmalifalak, H. Nguyen, R. Zheng, Z. Han, H. Li, and L. Song. Bad data injection in smart grid: attack and defense mechanisms. *IEEE Communications Magazine*, 51(1):27–33, 2013. xix, 7, 10

[Hug] J. Huggins. Selenium. [Online] `https://www.selenium.dev/` [Cited 17 July 2021]. 42

[JP16] P. M. Jacob and M. Prasanna. A Comparative analysis on Black box testing strategies. In *2016 International Conference on Information Science (ICIS)*, pages 1–6, 2016. ix, 10

[Kala] KaliTools. hping3 Package Description. [Online] https://tools.kali.org/information-gathering/hping3 [Cited 9 March 2021]. 23

[Kalb] A. Kalwan. What is Security Testing and how to perform it? [Online] https://www.edureka.co/blog/what-is-security-testing/ [Cited 9 March 2021]. 12

[KMJ18] T. Kakarla, A. Mairaj, and A. Y. Javaid. A Real-World Password Cracking Demonstration Using Open Source Tools for Instructional Use. In *2018 IEEE International Conference on Electro/Information Technology (EIT)*, pages 0387–0391, 2018. 23

[KSH20] N. M. Karie, N. M. Sahri, and P. Haskell-Dowland. IoT Threat Detection Advances, Challenges and Future Directions. In *2020 Workshop on Emerging Technologies for Security in IoT (ETSecIoT)*, pages 22–29, 2020. 5

[LCS+21] C. Lopes, J. C. Costa, J. B. F. Sequeiros, T. Simões, M. M. Freire, and P. R. M. Inácio. Machine learning applied to security requirements elicitation: Learning from experience. *INForum*, pages 0–12, 2021. 3

[LMY+19] M. Li, B. Meng, H. Yu, K. Siu, M. Durling, D. Russell, C. McMillan, M. Smith, M. Stephens, and S. Thomson. Requirements-based Automated Test Generation for Safety Critical Software. In *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, pages 1–10, 2019. ix, 17

[Lue18] K. L. Lueth. State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating, 2018. [Online] https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b [Cited 9 March 2021]. vii, 2

[MAM09] K. M. Mustafa, R. E. Al-Qutaish, and M. I. Muhairat. Classification of Software Testing Tools Based on the Software Testing Methods. In *2009 Second International Conference on Computer and Electrical Engineering*, volume 1, pages 229–233, 2009. 11

[Mat20] K. Matthews. 4 Statistics That Reveal Major Problems With IoT Security, 2020. [Online] https://www.channels.theinnovationenterprise.com/articles/4-statistics-that-reveal-major-problems-with-iot-security [Cited 9 March 2021]. vii, 1, 2

[MK15] Y. Makino and V. Klyuev. Evaluation of web vulnerability scanners. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 1, pages 399–402, 2015. 23

[MNA+19] M. Matić, E. Nan, M. Antić, S. Ivanović, and R. Pavlović. Model-Based Load Testing in the IoT System. In *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, pages 310–315, 2019. ix, 16

[MRV+18] Y. Mahmoodi, S. Reiter, A. Viehl, O. Bringmann, and W. Rosenstiel. Attack Surface Modeling and Assessment for Penetration Testing of IoT System Designs. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 177–181, 2018. ix, 16

[Nav17] S. Naveed. Automatic validation of UML specifications based on UML environment models. In *2017 4th IEEE International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, pages 1–6, 2017. ix, 15

[NSCC20] H. N. Noura, O. Salman, A. Chehab, and R. Couturier. DistLog: A distributed logging scheme for IoT forensics. *Ad Hoc Networks*, 98:102061, 2020. Available from: `https://www.sciencedirect.com/science/article/pii/S1570870519306997`. xix, 8, 9, 10

[Oll] OllyDbg. Main Page. [Online] `http://www.ollydbg.de/` [Cited 9 March 2021]. 23

[O'R] O'Reilly. ARPspoof. [Online] `https://www.oreilly.com/library/view/learn-kali-linux/9781789611809/1bb735da-180c-4178-890f-b7026e8ea6ec.xhtml` [Cited 10 March 2021]. 24

[OWA] OWASP. Format string attack. [Online] `https://owasp.org/www-community/attacks/Format_string_attack` [Cited 16 March 2021]. xix, 9, 10

[OWA18] OWASP. IoT Top 10, 2018. [Online] `https://owasp.org/www-project-internet-of-things/` [Cited 15 March 2021]. 6, 7

[PJB+17] S. Patil, A. Jangra, M. Bhale, A. Raina, and P. Kulkarni. Ethical hacking: The need for cyber security. In *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, pages 1602–1606, 2017. 22

[RKPR18] S. Rizvi, A. Kurtz, J. Pfeffer, and M. Rizvi. Securing the Internet of Things (IoT): A Security Taxonomy for IoT. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 163–168, 2018. xix, 9, 10

[ROC+20] S. Rizvi, RJ Orr, A. Cox, P. Ashokkumar, and M. R. Rizvi. Identifying the attack surface for IoT network. *Internet of Things*, 9:100162, 2020. Available from: `https://www.sciencedirect.com/science/article/pii/S2542660520300056`. xix, 6, 8, 10

[RRPB19] G. Rajendran, R. S. Ragul Nivash, P. P. Parthy, and S. Balamurugan. Modern security threats in the Internet of Things (IoT): Attacks and Countermeasures. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–6, 2019. xix, 7, 8, 9, 10

[SJK17] S. N. Swamy, D. Jadhav, and N. Kulkarni. Security threats in the application layer in IOT applications. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 477–480, 2017. viii, 1, 5

[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 *Global Internet of Things Summit, Endorsed by IEEE Global IoT Summit GIoTS*, pages –, June 2020. 30

[SM17] K. Sneha and G. M. Malle. Research on software testing techniques and software automation testing tools. In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pages 77–81, 2017. 10

[SMM10] Santosh Swain, Durga Mohapatra, and Rajib Mall. Test Case Generation Based on Use case and Sequence Diagram. *International Journal of Software Engineering*, 3, 01 2010. ix, 18

[SPPS17] R. Silva, P. Perera, I. Perera, and K. Samarasinghe. Effective Use of Test Types for Software Development. In *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 1–6, 2017. viii, 10

[SSS+20] M. G Samaila, J. B. F. Sequeiros, T. Simões, M. M. Freire, and P. 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. 30, 33

[SV11] M. S. Siddiqui and D. Verma. Cross site request forgery: A common web application weakness. In *2011 IEEE 3rd International Conference on Communication Software and Networks*, pages 538–543, 2011. xix, 9, 10

[SWJ13] R. Scandariato, J. Walden, and W. Joosen. Static analysis versus penetration testing: A controlled experiment. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, pages 451–460, 2013. 23

[Tcp] Tcpreplay. Tcpreplay - Pcap editing and replaying utilities. [Online] `https://tcpreplay.appneta.com/` [Cited 10 March 2021]. 24

[Tuta] TutorialsPoint. Big-Bang Testing. [Online] `https://www.tutorialspoint.com/software_testing_dictionary/big_bang_testing.htm` [Cited 9 March 2021]. 11

[Tutb] TutorialsPoint. Bottom Up Testing. [Online] https://www.tutorialspoint.com/software_testing_dictionary/bottom_up_testing.htm [Cited 9 March 2021]. 11

[Tutc] TutorialsPoint. Boundary Testing. [Online] https://www.tutorialspoint.com/software_testing_dictionary/boundary_testing.htm[Cited 9 March 2021]. 11

[Tutd] TutorialsPoint. Condition Coverage Testing. [Online] https://www.tutorialspoint.com/software_testing_dictionary/condition_coverage_testing.htm [Cited 9 March 2021]. 11

[Tute] TutorialsPoint. Equivalence Partitioning Testing. [Online] https://www.tutorialspoint.com/software_testing_dictionary/equivalence_partitioning_testing.htm[Cited 9 March 2021]. 10

[Tutf] TutorialsPoint. Integration Testing. [Online] https://www.tutorialspoint.com/software_testing_dictionary/integration_testing.htm[Cited 9 March 2021]. 11

[Tutg] TutorialsPoint. System Testing. [Online] https://www.tutorialspoint.com/software_testing_dictionary/system_testing.htm[Cited 9 March 2021]. 11

[Tuth] TutorialsPoint. Top Down Integration Testing. [Online] https://www.tutorialspoint.com/software_testing_dictionary/top_down_integration_testing.htm [Cited 9 March 2021]. 11

[Tuti] TutorialsPoint. White box Testing. [Online] https://www.tutorialspoint.com/software_testing_dictionary/white_box_testing.htm[Cited 9 March 2021]. 11

[VG16] Vishawjyoti and P. Gandhi. A survey on prospects of automated software test case generation methods. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 3867–3871, 2016. 18

[VS20] N. Varghese and R. Sinha. Can Commercial Testing Automation Tools Work for IoT? A Case Study of Selenium and Node-Red. In *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, pages 4519–4524, 2020. ix, 21, 27

[Wir] Wireshark. About Wireshark . [Online] https://www.wireshark.org/ [Cited 10 March 2021]. 24

[WZLH14] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou. DDoS Attack Protection in the Era of Cloud Computing and Software-Defined Networking. In *2014 IEEE 22nd International Conference on Network Protocols*, pages 624–629, 2014. xix, 9, 10

[YPAO20] G. Yadav, K. Paul, A. Allakany, and K. Okamura. IoT-PEN: A Penetration Testing Framework for IoT. In *2020 International Conference on Information Networking (ICOIN)*, pages 196–201, 2020. ix, 21

[YVY+19] S. Yatskiv, I. Voytyuk, N. Yatskiv, O. Kushnir, Y. Trufanova, and V. Panasyuk. Improved Method of Software Automation Testing Based on the Robotic Process Automation Technology. In *2019 9th International Conference on Advanced Computer Information Technologies (ACIT)*, pages 293–296, 2019. ix, 20

[Zho] Weilin Zhong. Command Injection. [Online] `https://owasp.org/www-community/attacks/Command_Injection` [Cited 16 March 2021]. xix, 8, 10

Appendix A

Document Created to Guide the Test Subjects

This appendix contains the document presented to the test subjects in order to guide their evaluation of SAM and the ACISM module. The document presents a detailed description of each one of the scenarios proposed. All the scenarios are accompanied by a diagram to better exemplify them. The contents of this appendix are briefly discussed in chapter 4.

Internet of Things Scenarios for Testing Proposes

SECUR IoT ESIGN Team

Towards the assurance of SECURity by dESIGN of the Internet of Things

June 2021

This document aims to describe several security related scenarios to be tested and demonstrated under the first alpha demonstration of the SAM platform. The user is asked to choose one or more scenarios and subsequently run several of the available SAM modules in order to achieve a desired set of security requirements, best practices, or security related algorithms.

Scenarios List

1 Smart Transportation and Logistics

Both the handling of logistics in moving cargo and monitoring of the transportation means can have their efficiency improved with IoT. Either in terms of locating either cargo or transport in real time, monitoring through IoT devices can aid in reducing costs, increase time efficiency, prevent mechanical issues, and predictive maintenance. The cargo itself, specially if sensitive, can also be similarly monitored (e.g., temperature-sensible cargo, such as perishable goods).

1.1 Perishable Cargo and Transportation Monitoring

This example assumes the monitoring of perishable cargo and its transportation means (in this specific case, a truck). The system is composed of sensors that

monitor the cargo, in terms of temperature, humidity, and positioning, while the truck is equipped with an OBD scanner, that connects to the truck's ECU and feeds data on the main sensors, and an array of other sensors that monitor most components, fluids, tire pressures, and also GPS tracking. All these truck sensors will communicate with a hub that has a data connection, via 4G technology, to report back all the data, in real time, to a cloud service, where it will be analyzed and monitored by the end user. The connection between the sensors, OBD scanner, and the hub will be made through Bluetooth. The cargo sensors are able to directly report their status to the infrastructure: each container has its own 4G connection, so that cargo can be continually monitored even when in storage, or changing transportation means. Figure 1 schematizes this system through components.

Some other more specific system characteristics:

- All system components should be able to detect if physical tampering is attempted, e.g., if someone gains physical access to any of them, they should not be able to access any information, or attack other components from any information gained on that component;
- The system should always be available and have a nearly optimal uptime;
- Any captured transmitted data should not be possible to be re-sent;
- Unauthorized third parties should not be able to access the system nor any of its communications and data;
- The system should automatically trigger warning if a deviation in the data is registered (e.g., sudden tire pressure change, or temperature increase in one of the cargo boxes).

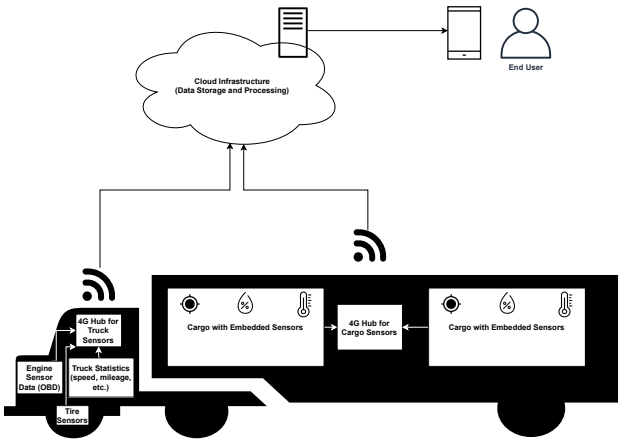

Figure 1: Component Diagram of the cargo and transportation monitoring example.

2 Smart Grids

Smart Grids pertain to the adaptation of IoT technologies to the electrical power grid management and distribution, to make it more efficient and failure-resistant. It can be applied in several points of the supply chain, from production to transmission, distribution and consumption.

2.1 Optimization of Infrastructure and Energy Usage for EV Home Charging

This scenario describes the use of IoT in an electric grid to ensure that EV charging at individual homes is used in a way that it does not stress the power grid to an unreasonable level. This will ensure that charging is mostly done during the times of least consumption and, if possible, using mostly renewable energy sources. EVs will also be used as temporary grid batteries, allowing the system to discharge them to offset extra load on the grid. This will bring better efficiency, lower overall consumption and reduce costs. For this system to work, wall chargers will need to communicate with the infrastructure, which will include smart metering in the entire grid, from production to distribution and consumption. Aggregators will take care of metering the usage and total available battery power, while load balancers will act as a medium between the production and consumption elements of the grid.

Figure 2 exemplifies the architecture of the system that is comprised by components.

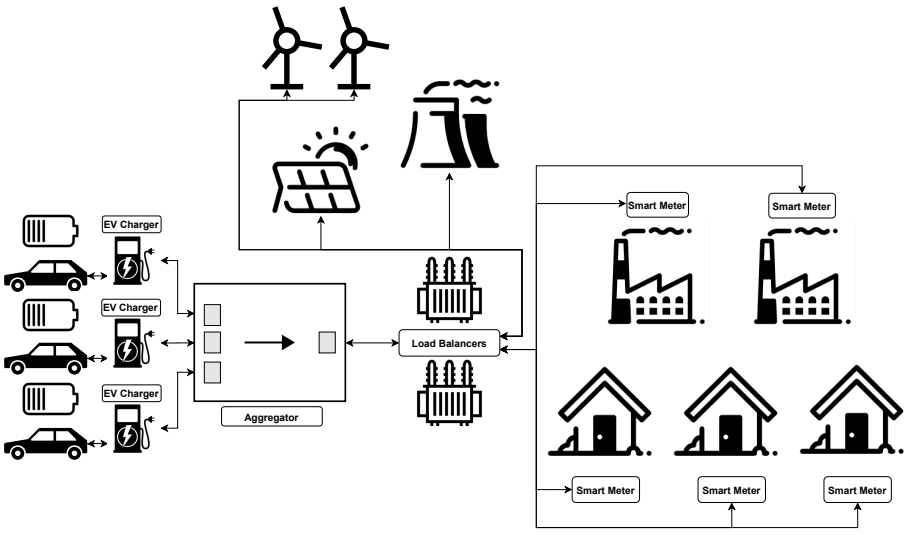

Figure 2: Component Diagram of the EV Home Charging Infrastructure example.

3 Smart Environments

Home automation, or domotics, are other terms frequently used to describe the automation and integration of IoT into common homes or offices. The premise here is to automate most common attributes, such as lighting, climate or appliances. Some common examples that can already be found in most houses include automated lighting control (either time sensitive, programmable, remotely accessible), climate control, energy spending control, air quality, smoke, CO2 and other types of detectors and sensors, automated cleaning systems (e.g., vacuum robots), security and intrusion detection, automated blinders, or occupancy detection (e.g., detecting if a division has occupants and automatically turning the lights on or off). The great advantage of IoT in these scenarios is that it gives users a fine control over their spaces, allowing for significant energy savings, comfort in terms of remote access and task automation, and increasing the livability and quality standards of that space.

3.1 Smart Lamp

This case is based on smart lamps that can be controlled by a smartphone belonging to the user. These lamps will be connected to the home wireless network, by downloading the app, creating an account, and connecting their mobile phones to the same network, the users will have control over these devices. This system its composed by the router, a coordinator, the lamps, and a smartphone. The router will act as a mediator between the user and the lamp: the user emits commands towards the lamps through the app, and these commands are sent using wireless communication to a coordinator. The coordinator determines which lamps will be affected by the command and will send the command to the respective lamps. The impacted lamps receive this command through a microprocessor that will perform the intended action.

The following security requirements should be considered when developing these systems:

- The users and their personal network information should be protected and impossible to access. For example, if a perpetrator gains access to user information, it will be able, based on the activity of the lamps (if they are on or off), to know if the user is in his home or not. That is why this confidentiality and privacy are a requirement in this type of systems;
- User data should not be modified by unauthorized persons. E.g., if the users login password was changed by a unauthorized third party, the user would lose the ability to control the smart lamp, which would invalidate the system. By having integrity and accountability, this type of situations could be prevented;
- Only authorized devices and users are allowed to access any information on the network and control the lamps. Without this property, the lamps

could be controlled by anyone with good or bad intentions, and user data could also be used in the same manner;

- The services of the smart lamps systems should be available to authorized entities whenever needed. I.e., if a system update is required, the smart lamp system should be always available to download and implement this requirement. This way the user will always be able to have the best software possible;
- Even if an attacker gains physical access into the home where the smart lamps are installed, it should be improbable for the attacker to have access to any meaningful data or to spread the attack to other components of the system, and if the attacker tries to alter the system components, it should be able to warn the user. With these requirements we assure that even by gaining physical access to the system user privacy will always be assured;
- The data sent to the system must always be the most recent data. In other words, if the attacker tries to resend the same information to the system, it should be able to detect that an old message is being sent.

Figure 3 presents the overall architecture of this system that is comprised by components.

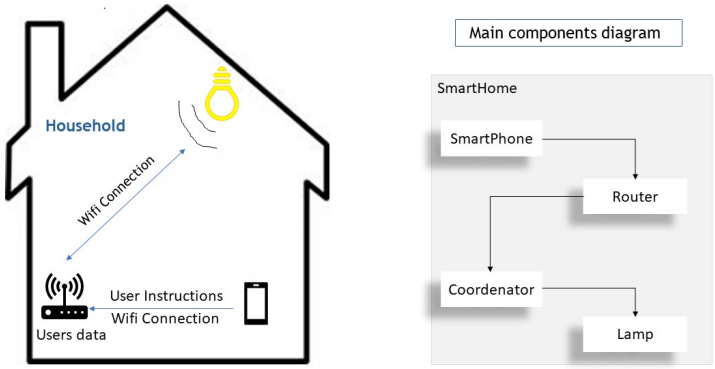

Figure 3: Component Diagram of a Smart Lamp.

4 Healthcare

Healthcare is an area where pervasive use of IoT systems can be of great use to gather real-time data to aid in the decision process, or simply have medical

aiding devices that can be remotely adjusted. Here, sensors play a major role, enabling paradigms such as telemedicine, by allowing remote monitoring of the status of a given patient, or correct prescription compliance, besides providing detection of events, enabling a faster response. Not only directly related to patient health, but application of IoT in healthcare can also increase hospital workflow and organization, reduce incidents to to wrong dosage or wrong patient identification, and automate several care related tasks.

4.1 Pacemaker with Mobile Device Interface

This scenario describes the usage of smart pacemaker and a mobile interface that enables a user to access heart-related data. The communication will be accomplished through an external device that, when close to the pacemaker, and connected to a phone via Bluetooth, collects information from it. Once the information is synced, the user only needs to open the app and log in to access it. The pacemaker will have sensors to collect data on the health state of the patient, as well as its own status. The app will also enable doctors to access information from the patient. Using Wi-Fi technology, notifications may also be triggered to warn a nearby hospital if the state of a user reaches a critical condition. This system is schematized in figure 4.

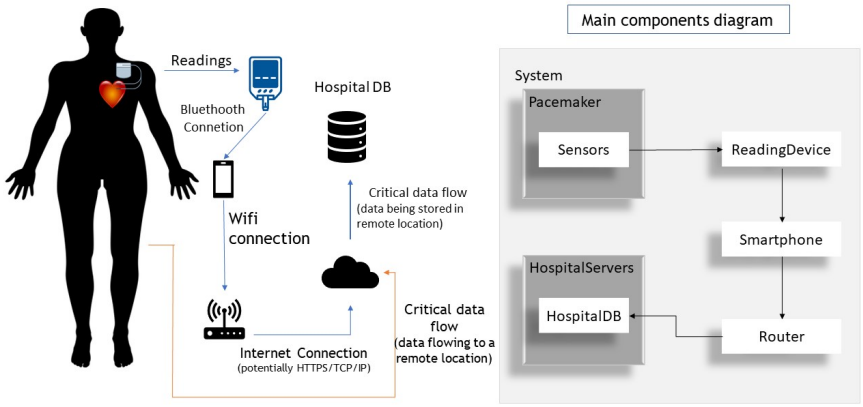

Figure 4: Component Diagram of a Smart Pacemaker.

5 Agriculture and Environmental Monitoring

IoT can bring several advantages when it comes to monitoring agricultural fields, crops, forests or other agricultural related environments. Through them, and with the aid of sensor arrays, it is possible, for agriculture, to, e.g., keep track

and monitor soil composition, humidity, temperature, or crop growth through a video feed. Another area of agriculture where IoT can be useful is with livestock, allowing to know the position of animals, their vitals and general health, and how they behave over time periods. Environmental monitoring has some similarities to agriculture, as it is done resorting to sensor arrays to monitor temperatures, humidity, or soil changes.

5.1 Smart Irrigation System

This scenario exposes a smart irrigation system used in agriculture to save resources and improve the quality of crops. This IoT system comprises five units: sensing, processing, front-end, actuation, and persistence. The sensing unit is composed of a number of wireless nodes integrating soil moisture and temperature sensors. The processing unit consists of a receiver node serially connected to a Raspberry Pi. Sensed data is sent from the receiver to the gateway using the MQTT communication protocol. A graphical Web interface and a mobile application are implemented to enable the user to monitor data in real-time. When the moisture levels of the soil reach a certain threshold level, the user is alerted via a mobile notification or through the Web interface. An immediate action can be taken to control the motor in order to reduce water and energy wastage. Finally, the persistence unit is used to store data directly from the publisher and can also be accessed via the Web or mobile application. This system is schematized in figure 5.

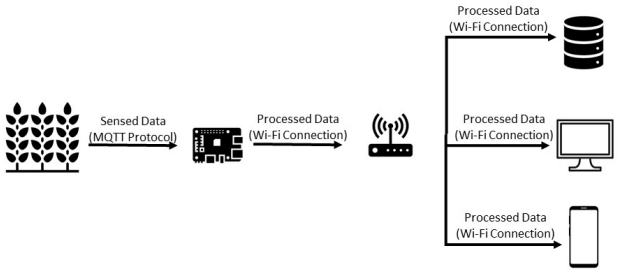

Figure 5: Component Diagram of a Smart Irrigation System.

6 Smart Wearables

Wearables are some of the most common IoT devices that users have had access over the last years. From fitness trackers to smart watches, wearables are simple devices that usually connect to a smartphone, and can incorporate several sensors. These sensors may allow the user to measure exercise intake, health data, and other related data. The information is later transmitted to the smartphone and displayed to the user. Wearables are mainly powered by the Bluetooth protocol for communication, but most are also capable of operating independently, having some processing capabilities. In some cases, these devices can also be included in the healthcare category, as some of them gathered data from body sensors (most commonly, heart rate).

6.1 A Generic Smartwatch

This case describes one of the most common IoT gadgets, a smartwatch. This device will connect to the user's phone through Bluetooth and will be paired with a specific app that the user needs to download to his/her phone.

After a successful pairing, this device will be able to receive user-relevant events, such as text messages, emails, or calls. Any other alerts can also be triggered to be displayed on the smartwatch. An interaction between the user and the smartwatch is also possible, for example, answer calls, reading or replying to messages or emails. In several of these devices, Google's assistant is included to enrich the interaction between the user and the device through the means of a built-in microphone and speaker. This system is schematized in figure 6.

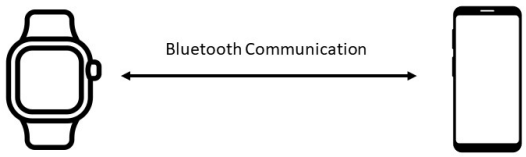

Figure 6: Component Diagram of a Generic Smartwatch.

7 Smart Manufacturing

The Industrial Internet of Things (IIoT) refers to the use of IoT in industrial applications, with the main purpose of increasing efficiency and reliability. It incorporates different application areas, such as robotics, medical devices, or production chains, and has a strong emphasis in M2M communication and big data. It also integrates well with industrial control systems, SCADA, CPSs, PLCSs and DCSs. It brings gains in productivity, improves maintenance costs and scheduling, reduces downtime and overall improves the production efficiency in industrial scenarios.

7.1 Smart Factory

This scenario describes a smart factory that uses multiple technologies to reduce costs, down-times, and waste. The factory floor consists of multiple machines and devices interconnected and facilitates data between people and machines. One example is a machine with an integrated display that produces plastic, used to monitor the temperature, air pressure, plastic consistency, and time spent working. Although this machine provides this data, it requires sensors to collect them. These sensors are directly embedded in the machine, and its information is displayed on the devices.

The previously described devices are only used to acquire data from the factory floor. This information is sent to the Cloud or owned servers to be processed and provide the intelligence associated with the smart factory. When on the server/Cloud, it is essential to use Big Data Analytics to assure cost reduction, downtime decrease, and waste minimization. The smart factory contains gateways used to communicate between multiple machines, devices, and sensors. It also has gateways that communicate with the Internet, that if necessary, sends information to the Cloud. This system is schematized in figure 7.

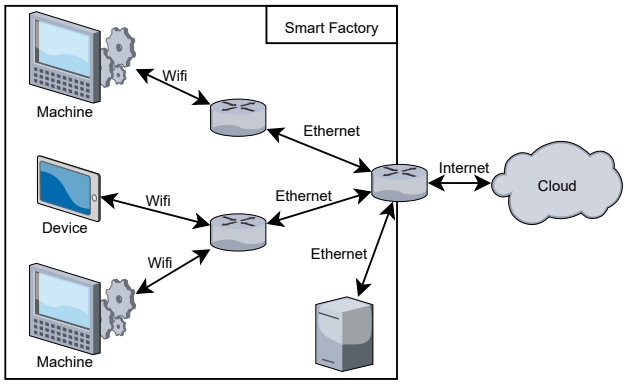

Figure 7: Overview of Smart Factory Components.