

Fartein Lemjan Færøy

Expanding the Capabilities of Cyber Range Attack Agents

Master's thesis in 5-year MSc in Communication Technology and Digital Security

Supervisor: Basel Katt

Co-supervisor: Muhammad Mudassar Yamin

July 2022



NTNU – Trondheim
Norwegian University of
Science and Technology

Expanding the Capabilities of Cyber Range Attack Agents

Fartein Lemjan Færøy

Submission date: July 2022

Supervisor: Basel Katt, NTNU, IIK

Co-supervisor: Muhammad Mudassar Yamin, NTNU, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title: Expanding the Capabilities of Cyber Range Attack Agents
Student: Fartein Lemjan Færøy

Problem description:

Penetration testing describes the attempt of hacking a computer system in order to uncover potential vulnerabilities, and thus assess its level of security. Because a penetration test is mostly done manually, it is costly, time consuming, and the results will depend on the incline and expertise of the tester. A fully automatic penetration test would be fast, inexpensive and deterministic. In addition, it would be highly useful for training purposes in a cyber range. However, research has proven autonomous penetration testing to be a complex task, and especially challenging for devices in the Internet of Things (IoT) domain.

Yamin and Katt¹ developed a new modelling plan to subdivide the task of autonomously attacking and defending systems in a cyber range network. In the study, they demonstrated how agents on both sides could employ the modelling plan with a selected number of offensive and defensive tactics. This thesis will try to expand the capabilities and usability of the attacking agent. This will be done by employing the modelling plan on an exploit targeting a vulnerability in an IoT device. Finally, the new attack agent will be tested on a running IoT device, and compared to current automated vulnerability scanners and exploitation systems.

Date approved: 2022-02-23
Responsible professor: Basel Katt, NTNU, IIK
Supervisor(s): Muhammad Mudassar Yamin, NTNU, IIK

¹Yamin, M. M., & Katt, B. (2022). Use of Cyber Attack and Defense Agents in Cyber Ranges: A Case Study. NTNU.

Abstract

Internet of Things (IoT) devices are becoming a part of our daily life; from health monitors to critical infrastructure, they are used everywhere. This makes them ideal targets for malicious actors to exploit for nefarious purposes. Recent attacks like the Mirai botnet are just examples in which default credentials were used to exploit thousands of devices. This raises a lot of questions about IoT device security. Keeping that in mind, we aimed to investigate security of IoT devices in this thesis. A penetration test is a way of ensuring the security of a system, but manually testing the billions of IoT devices existing is infeasible at best.

This thesis has therefore examined autonomous penetration testing on IoT devices. In a recent study, a method named Execution Plan (EP) was developed for modelling automated attack agent decision making in a cyber range. We have (1) investigated how the EP model can be applied for modelling an autonomous IoT penetration testing agent. Furthermore, we have (2) investigated if some well known and severe Wi-Fi related vulnerabilities still exist in critical infrastructure. Through the methodology of Design Science Research and a case study we have shown that the EP model is sufficient for the purpose of modelling autonomous penetration testing agents. In addition, we have demonstrated that the vulnerabilities are in fact present in deployed and currently sold products used in critical infrastructure, and that they can be both autonomously revealed through a penetration test, and exploited.

Sammendrag

Enheter knyttet til Tingenes Internett (IoT)-domenet er i ferd med å bli en del av vårt daglige liv; fra kroppsmonitører til kritisk infrastruktur brukes de overalt. Dette gjør dem til ideelle mål for aktører som ønsker å utnytte dem til ondsinnede formål. Nylige angrep som Mirai-botnettet er bare eksempler der lett gjettbare passord og brukernavn ble brukt til å utnytte tusenvis av enheter. Dette fører til mange spørsmål om IoT-enhetssikkerhet. Med det som utgangspunkt satte vi som mål å undersøke sikkerheten til IoT-enheter. En penetrasjonstest er en måte å undersøke sikkerheten til et system på, men manuell testing av milliarder av IoT-enheter som eksisterer er umulig å gjennomføre i praksis.

Denne oppgaven har derfor undersøkt autonom penetrasjonstesting på IoT-enheter. I en fersk studie ble en metode kalt Execution Plan (EP) utviklet for å modellere automatisk beslutningstaking av programvare for dataangrep i et en cyber range. Vi har (1) undersøkt hvordan EP-modellen kan brukes for å modellere autonome IoT-penetrasjonstester. Videre har vi (2) undersøkt om noen velkjente og alvorlige Wi-Fi-relaterte sårbarheter fortsatt eksisterer i kritisk infrastruktur. Gjennom metodikken til Design Science Research og en casestudie har vi vist at EP-modellen er tilstrekkelig for formålet om å modellere autonome penetrasjonstester. I tillegg har vi demonstrert at sårbarhetene er tilstede i produkter som selges og brukes i kritisk infrastruktur, og at de både kan avsløres gjennom en autonom penetrasjonstest, og utnyttes.

Preface

This is the conclusion of a 5+ year MSc at NTNU.

Firstly, I would like to thank Muhammad Mudassar Yamin and Basel Katt for continuous support throughout the last year. Their vast knowledge and expertise within the topic has been inspiring and essential. Without them, this project and thesis would not have been possible.

Secondly, my gratitude must be directed to Mujahid Islam Peal for the great advice on the infamous topic of writing theses, and to Ahmed Walid Amro for valuable and kind support during the case study development.

Lastly, I would like to thank UKEkoret Pirum for all the fun, memories and friends I have made there, and Milena for emotional support during an existential crisis imposed by this masters thesis.

Contents

List of Figures	vii
List of Acronyms	ix
1 Introduction	1
1.1 Motivation	1
1.2 Problem Description and Research Objectives	2
1.3 Thesis Outline	3
2 Background and Related Work	5
2.1 Internet of Things and Cybersecurity	5
2.1.1 Threats and Risks	5
2.1.2 Threat Actors	6
2.1.3 Attack Surface and Security Issues	7
2.2 Penetration Testing	8
2.2.1 Testing of IoT Devices	10
2.2.2 Autonomous Penetration Testing	10
2.3 Wi-Fi	11
2.3.1 Encryption Standards and WPA2 Personal	11
2.3.2 The 4-Way Handshake	12
2.3.3 Deauthentication Frames	13
2.3.4 Evil Twin Attack	14
2.4 The EP Model and Formal Specification	14
2.4.1 Formal Specification and Verification	15
2.4.2 TLA+	16
2.5 Related Work	16
3 Methodology	19
3.1 DSR Life Cycle and Conditions	19
3.2 DSR Applied to the Project	20
4 Case Study and Environment	21
4.1 Hardware	21

4.2	Software	21
4.2.1	The Aircrack-ng Suite	22
4.2.2	Hostapd	24
4.3	Target IoT Device	24
4.4	Physical setup	25
5	System Design	27
5.1	The Attack Procedure	27
5.2	EP Models of Attacks	28
5.3	Formal Implementation and Verification	29
5.4	Implementation	35
5.4.1	Tool Interface	35
5.4.2	Agent Decision Making	36
6	Attack and Results	39
6.1	Attack 1: DoS	39
6.2	Attack 2: Evil Twin	39
7	Discussion	41
7.1	Authenticity of Attack	41
7.1.1	Network Interface Range	41
7.1.2	Time to Crack Network Password	41
7.2	Research Objectives	43
7.2.1	Research Objective 1	43
7.2.2	Research Objective 2	43
8	Conclusion and Future Work	45
	References	47

List of Figures

2.1	The 4-way handshake	13
2.2	Warning issued when attempting connection to a rogue AP with no encryption scheme	14
3.1	Steps in the DSR life cycle [24, p. 27]	20
4.1	Airodump-ng running in terminal	23
4.2	Aircrack-ng cracking a WPA2 key	23
4.3	The A200 connected to a battery and a VHF antenna	25
4.4	The Wi-Fi settings menu in A200	25
4.5	Diagram of the case study setup	26
4.6	OpenCPN debug window with received NMEA packets	26
5.1	Sequence diagram of DoS and Evil Twin attack	28
5.2	EP model of the DoS attack	29
5.3	EP model of the Evil Twin attack	30
5.4	The formal specification of the DoS attack EP model written in TLA+	32
5.5	State transition diagram showing all possible states of the DoS attack EP model	32
5.6	The formal specification of the Evil Twin attack EP model written in TLA+	33
5.7	State transition diagram showing all possible states of the Evil Twin attack EP model	34
5.8	Class diagram of the agent	35
6.1	YAML file for a DoS attack	39
6.2	Performing the DoS attack	40
6.3	DoS attack in OpenCPN debug window	40
6.4	Performing the Evil Twin attack	40
6.5	Debug view of Evil Twin connection	40
7.1	Aircrack-ng password cracking speed on Kali Linux machine	42
7.2	Aircrack-ng password cracking speed on the desktop computer	42

List of Acronyms

AES Advanced Encryption Standard.

AI Artificial Intelligence.

AIS Automatic Identification System.

AP Access Point.

APT Advanced Persistent Threat.

BDI Belief-Desire-Intention.

BSSID Basic SSID.

CCMP Counter Mode with Cipher Block Chaining Message Authentication Code Protocol.

DDoS Distributed Denial of Service.

DoS Denial of Service.

DSR Design Science Research.

EAPOL Extensible Authentication Protocol over LAN.

EP Execution Plan.

ESSID Extended SSID.

GPS Global Positioning System.

GTK Group Temporal Key.

IDS Intrusion Detection Systems.

IEEE Institute of Electrical and Electronics Engineers.

IGTK Integrity Group Temporal Key.

IIoT Industrial Internet of Things.

IoMT Internet of Medical Things.

IoT Internet of Things.

IPS Intrusion Protection Systems.

LAN Local Area Network.

MIC Message Integrity Code.

MitM Man-in-the-Middle.

NCR Norwegian Cyber Range.

NIC Network Interface Card.

NMEA National Marine Electronics Association.

PMK Pairwise Master Key.

PSK Pre-Shared Key.

PTES Penetration Testing Execution Standard.

PTK Pairwise Transient Key.

RL Reinforcement Learning.

SSID Service Set Identifier.

VHF Very High Frequency.

VTS Vessel Traffic Services.

WEP Wired Equivalent Privacy.

WLAN Wireless LAN.

WPA Wi-Fi Protected Access.

Chapter 1

Introduction

There are many definitions of the Internet of Things (IoT). Oracle describes it as a "network of physical objects—"things"—that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet" [48]. IoT devices are employed in anything from refrigerators and health equipment, to sensors and actuators in industrial networks. Some categorize them by the four subdomains Industrial Internet of Things (IIoT), Internet of Medical Things (IoMT), Smart Cities and Smart Homes, but IoT is applicable in many other fields and we can expect it to be an even more prevalent term in the time to come [56]. It is estimated that more than 12 billion devices are currently connected to the Internet, and by 2025, this number is expected to reach beyond 27 billion [60]. We can safely conclude that IoT handle systems and data that influence lives and economies all over the globe. Ensuring its security is a crucial task for both manufacturers and consumers in the time to come.

1.1 Motivation

Devices that belong in the IoT domain are typically designed and used for a specific application. Compared to a general purpose home computer, they are limited in terms of hardware and computational power, but benefit from being cheaper to produce, smaller in size and consuming less power. This ensures their broad applicability. With IoT expanding in terms of number of devices and applications, its appeal among cyber criminals increases accordingly. The physical presence and influence the devices have on their environment make them valuable targets. Additionally, they are highly suitable as pawns in botnets where they can be used for cryptomining or Distributed Denial of Service (DDoS) attacks [34]. In 2021 an average IoT device was attacked more than 5000 times each month according to Symantec [61]. This is a grave concern in the industry. Through a survey among their customers, a cybersecurity firm found that more than half of the organizations asked had discontinued a new project in 2021 because of security concerns [33]. Furthermore, three out of four organizations

had been in one or more situations where they considered the level of security to be inadequate, and 57% reported that cybersecurity breaches was their biggest concern regarding IoT implementation.

A way of testing and assuring the security of a device is through a process called *penetration testing*. This is an audit that attempts to reveal the potential security flaws and vulnerabilities of a device by attacking it in any way a hacker, bot or malware would [64]. The test is usually performed manually by one or more security experts with the help of hardware and software hacking tools. These tools can scan for services, ports and vulnerabilities, and some are able to exploit the systems, extract data or navigate within a breached target. Even though many tools automate large parts of the penetration test, most tools require human interaction and expertise which can be time consuming and costly. A team of security experts may be required to work many hours to sufficiently test all attack vectors of a system. This makes the penetration test an expensive and tedious process with an end result that depends on the competence and incline of the tester. Consequently, the test may not be considered worth the time and resources invested for certain products - especially in the case of low-priced IoT devices.

Fully automated penetration tests would be fast and inexpensive, improving cybersecurity by ensuring audits are viable for more manufacturers and their products. The tests could be performed at any stage during the development without obstructing the process, while the results of each would be more deterministic, and thus quantifiable and comparable. Furthermore, they would be valuable for educational purposes when training students and security personnel within a learning environment. Moreover, autonomous penetration tests could assist during development and testing Intrusion Detection Systems (IDS) and Intrusion Protection Systems (IPS).

1.2 Problem Description and Research Objectives

With the aforementioned situation in mind, we established the initial goal of researching autonomous penetration testing of IoT devices. Recently, Yamin & Katt conducted experiments involving autonomous attack and defense agents in a cyber range [76]. A *cyber range* is an isolated environment for cyber security testing and learning. Within this environment, both students and software agents operated. For the agent decision making, the researchers developed a model named Execution Plan (EP). This model was described and verified using the logic programming language *Datalog* before the agent was developed. While their use case was emulated and educational, the cyber range emulates realistic scenarios, implying that the agents could operate outside the isolated environment as well. Because the actions of the attack agent can be interpreted as an automated penetration test, the work of Yamin & Katt lays an interesting foundation for further research within the field

of automated penetration testing. As we will discuss in Section 2.5, most current research within this field focus on Artificial Intelligence (AI) and machine learning which has some issues. The EP model is a novel approach to solving these issues which is worth further investigation.

Preliminary research on IoT devices encouraged further inquiry into the technologies in which they communicate and connect to the Internet. Wireless connection is often the most convenient option because it is inexpensive, mobile, and requires little to no setup. In homes and offices, Wi-Fi becomes a quick and simple solution because its coverage is close to everywhere. However, a notable caveat with Wi-Fi networks comes with its issues regarding security and availability. As we will discuss in Section 2.3.1, a substantial majority of these networks are still encrypted using a vulnerable generation of the Wi-Fi encryption scheme Wi-Fi Protected Access (WPA), even though a more secure solution has been available since 2018 [4]. It seems that neither producers nor consumers are able or willing to make a quick transition. This puts the network and its connected devices at risk, and should be a concern in the growing IoT market [36].

This project and thesis will involve a case study in which the EP model will be used to create an autonomous attack agent. We will follow the Design Science Research (DSR) methodology with the case study as a foundation. Using the artifacts developed for the case study, we will attempt to expand on existing knowledge within the concepts of autonomous penetration testing of IoT devices, IoT security in relation to Wi-Fi networks, and the EP model. Following are the two research objectives we are aiming to realize in this project:

- **Research objective 1: Investigate the use of the EP model to design an autonomous penetration test for an IoT device**

By expanding on the work of Yamin & Katt, we will examine how the EP model can be utilized in real world scenarios, how the autonomous agent behaves in accordance to the model, and what potential results we can expect from it.

- **Research objective 2: Investigate and highlight Wi-Fi related vulnerabilities in IoT devices**

In the case study, we will identify and focus on one or more vulnerabilities present in currently produced and employed IoT devices. From the results, we will discuss the severity of these vulnerabilities.

1.3 Thesis Outline

Chapter 1 has discussed the motivation behind this thesis and formulated the research goals. Chapter 2 will explain the concepts behind IoT and Wi-Fi security, penetration

testing, and the EP model, to support the artifact and the research goals. The DSR methodology and its implications on the project is described in Chapter 3. Software and hardware tools and devices used in the case study will be accounted for in Chapter 4. The system design and implementation are presented in Chapter 5. Chapter 6 will show the results, which will be discussed in Chapter 7. A final summary and conclusion will be given in Chapter 8.

Chapter 2

Background and Related Work

2.1 Internet of Things and Cybersecurity

One may think that the security breach of a smart light bulb would be inconsequential, but there are numerous risks associated with the successful attack on an IoT device. This section will address the importance of IoT security and why it is difficult, along with potential attack vectors, threat actors and existing counter measures.

2.1.1 Threats and Risks

If an IoT device is compromised, the perhaps most evident issue comes from the potential loss of availability as defined in the CIA Triad [55]. As mentioned, IoT devices are used for an extensive number of applications. In the case of smart homes, a breached device can imply a broken refrigerator, unknown access to home security cameras, or an open smart lock on a front door or a safe. A broken sensor or actuator in a factory can halt production or endanger workers and end-users. In terms of the IoMT, the negative implications on availability in equipment like modern pacemakers and insulin pumps can have fatal consequences [38].

Many IoT devices handle sensitive information. When used in commerce, this could be customer information, physical security information or industry secrets that could be valuable to criminals or competitors. Devices sold to consumers may handle and store sensitive information about the owner. For instance, data processed by a smart thermostat may reveal when a house is vacant [23], workout devices may handle sensitive health data about the owner, and home surveillance cameras may record continually, even when the owners are home. Stolen data from such devices may be used in profiling for criminal purposes, identity theft, blackmailing, spear-phishing, or weakening the security of a physical place of residence or business.

A security researcher also revealed that a smart light bulb stored network access keys in plaintext, which were retrievable even after the light bulb was disconnected

[44]. Similarly, a compromised device can act as an entry point into a system or network [23].

The IoT has also been the main target for different types of malware. In 2016, the largest DDoS attack ever recorded at the time was delivered by a botnet named Mirai [3]. It consisted of over 300'000 infected IoT devices, and the attack brought down several global services including GitHub, PayPal, Amazon.com, BBC, PlayStation Network and Spotify. Following the attack, the source code of the botnet malware was published on Hack Forums and GitHub allowing anyone to recreate the Mirai malware or use parts of it in their own malware [1, 22]. Because IoT devices are continually operative and connected to the Internet while often requiring minimal human interaction, they are highly suitable for botnets and the malware will rarely be revealed [34]. Botnet devices can be used for DDoS attacks, cryptomining, password hash cracking or other types of distributed computing. While it is unlikely that the owner is persecuted for the malicious acts of an infected device, the device itself may become slow, malfunction or take damage from overheating. In addition, the botnet administrator may extract sensitive information or in other ways misuse the device.

2.1.2 Threat Actors

As IoT devices are used in many different fields, from smart homes to billion dollar industries and government projects, the potential threat actors are similarly diverse. Background, funding, motivation, skills, target and scope will vary between different types which we can categorize under groups described below [35]. They are loosely sorted in ascending order based on their cyber security threat level for companies, governments and other high-value targets. The last two groups are the most likely to act as an Advanced Persistent Threat (APT) once compromising a device, posing as a major threat unbeknownst to the victim over weeks, months or even years [28].

Cyber-criminals is a term used for individuals that conduct illegal activities on the web that does not involve hacking. This includes drug dealing, human trafficking, sharing or downloading child pornography, and conducting financial fraud. While they are not directly a cyber security threat, they are criminals within the cyber realm and included in this list for the sake of completeness.

Script kiddies and **cyber-punks** have limited knowledge and skills, and use existing tools to exploit low hanging fruit. Fame among peers, small gains or simply entertainment are usually their motives.

Hactivists are the digital equivalent to activists. They consist of anonymous groups that target private organisations and governments to publicize a political agenda.

Cyber-terrorists are terrorists using the web to recruit new members and share information. They may also conduct attacks in the cyber domain with the same motives as terrorist attacks in the physical world.

Black hat hackers are mostly individual hackers with knowledge and expertise in hacking and the tools used. Their targets may be specific companies or individuals, or arbitrary devices found by means like the search engine Shodan. Their motives are usually reputation or financial gains.

Malware- and hacking tools coders are highly skilled adversaries that create tools and malware used to target different types of systems. They may work alone or in a criminal organization. They may sell the tools, use them in ransomware attacks or to create botnets. This is one of the most prevalent threats for IoT devices [17].

State-sponsored attackers are groups with extensive expertise and resources. They target corporations or governments in order to reveal trade or state secrets, plans or ideas, or in other ways harm the victim. Their attacks are sophisticated and may utilize zero-days, making them difficult to avert.

Knowing where the threats come from is important in order to understand how the security of a system should be adequately tested. The motivation and skill of an adversary will determine its targets and attack vectors. Due to the diverse set of threat actors, it is important that security audits test the breadth of potential attacks as they may come from any of these adversaries.

2.1.3 Attack Surface and Security Issues

To map the attack surface of a system, one should consider the various parts of the system, and how it operates. Due to the numerous applications and heterogeneous environments of IoT, researchers and developers have not been able to universally agree on a single architecture for describing IoT devices [29]. The approach described in this thesis is a 3-layer model which differentiates between the perception, network and application layer [53, 13]. The model can be applied to most IoT systems due to its high generality. Because the attack vectors used on each of the three layers are highly different from each other, the model is especially useful when investigating topics related to penetration testing [13].

Perception Layer

The physical part of the device is represented by the perception layer. The layer gathers information from and interacts with the physical world around it. To achieve this, the device can use sensors, actuators, GPS, RFIDs, or other similar technologies.

IoT devices may often have limited computing powers, storage and battery capacity. This limits the complexity of its encryption schemes and key lengths [26]. Furthermore, the devices may need to be small and lightweight which limits the possible physical hardening options. In IIoT the devices may be situated in places where maintenance is difficult, or neglected because they rarely receive human interaction. Such devices may be left untouched simply because they work, contributing to the growing concern of orphaned devices [54].

Network Layer

To control the actuators or process the information gathered in the perception layer, data must be transmitted between the physical and the application layer. The network layer connects the end nodes to network devices, servers or other IoT objects, and handles the corresponding data flow. The connection is often wireless due of cost, coverage and mobility. Examples of wireless technologies used within this layer include ZigBee, Bluetooth, WiFi, 4G, 5G, satellites, and combinations of them.

The network layer is prone to jamming attacks, access point spoofing, data sniffing, Man-in-the-Middle (MitM) attacks and more [26]. The devices may be used as an entry point into their connected networks which makes their security increasingly important. The various technologies used, and especially the combinations of them, presents a notable cyber security risk [13].

Application Layer

The application layer provides user interaction and management of the service provided by the IoT device. This may be presented as a smart home hub, a web or mobile application, or a machine-to-machine interface. Depending on the field, service and user, there are numerous technologies and applications which can be used on this layer.

Because the application layer often presents an interface to the Internet, it has the same security issues as most other computing devices with an Internet connection. While this layer can be harder to exploit, it will often be accessible from anywhere in the world, substantially increasing the potential threat actors. Examples of common attacks include credential guessing, SQL injection, buffer overflow and social engineering attacks [5].

2.2 Penetration Testing

A penetration test is a form of security audit which can be performed to ensure an appropriate level of security of a system [45, 64]. It is a form of stress test which usually attempts to bypass or break the authentication mechanism of the device,

or in other ways compromise its integrity, availability or confidentiality. This is accomplished by discovering vulnerabilities in hardware, software or communication protocols. A penetration tester may exploit the discovered vulnerabilities as a proof of concept, or elevate its privileges within the system to reveal more vulnerabilities. The test can be defined as either *black box* or *white box* [45]. The former suggests the tester knows nothing about the underlying systems and acts as an external attacker. In a white box penetration test, the attacker has some kind of inside knowledge or access, like a software source code. Within this thesis, we will only consider black box penetration testing because this method emulates hacking attempts as they are realistically performed.

Penetration tests are commonly performed by one or more experienced penetration testers who employ a broad set of hardware and software tools depending on the target system [64]. The software tools are used to scan the target in order to both map the system and discover potential vulnerabilities. Some tools are capable of exploiting vulnerabilities. Essentially, the software tools are automating parts of the process, but they require interaction to both run and interpret the output.

A penetration test on a system will have a large number of potential attack vectors and surfaces to test. The action space of possible steps to scan and exploit is vast, while the outcomes may have severe implications for the system and people compromised by the test. Because of this, a structured procedure should be followed to ensure the outcomes are correct and the interests of all afflicted parties are accounted for. To standardize the procedure, several methodologies have been suggested [59]. A commonly acknowledged and utilized methodology is the Penetration Testing Execution Standard (PTES) [49]. This standard describes the penetration test in seven steps:

1. **Pre-Engagement Interactions**

This step involves and emphasises the importance of clearly defining the target, scope and potential boundaries before interacting with a system.

2. **Intelligence Gathering**

Before attacking the system, the tester must know how it functions, is structured and can be interacted with. As mentioned above, automated software tools can often be used for this purpose.

3. **Threat Modelling**

To properly analyze the security of a system, the tester should know who could attack it and why. Thus, PTES threat modelling focuses on the assets that can be targeted and the liable threat actors.

4. **Vulnerability Analysis**

This step is where potential vulnerabilities, from misconfigurations to faulty designs, are uncovered. Many software tools, but also human interaction with the system is important to properly examine it.

5. **Exploitation**

To analyze the discovered vulnerabilities, the tester will attempt to exploit them. This should reveal their potential implications and may be used to uncover more vulnerabilities.

6. **Post-Exploitation**

If a component of the system has been successfully compromised, the value of the component should be evaluated with regards to its usefulness in further exploitation of the system.

7. **Reporting**

The value of the penetration test comes from reporting the discovered and exploited vulnerabilities. These should be evaluated according to their severity and risk.

2.2.1 **Testing of IoT Devices**

Penetration testing IoT devices is not much different from penetration testing of larger computer systems. The audit must inquire all three layers described in Section 2.1.3 which would be the case regardless of the target system. However, the tests performed on the network layer and particularly the perception layer may be different depending on the device tested. Due to the vast number of different applications and utilities provided by the IoT, there is a corresponding diversity in the various technologies utilized. The devices may have particular vulnerabilities related to their services, location, sensors, communication protocol, and so on.

The steps of the PTES discussed above can be applied to the penetration testing of IoT systems as well. The threat actors described in Section 2.1.2 are an important part of the threat modeling step, while the IoT architecture and security issues from Section 2.1.3 are essential when evaluating step 2, 3 and 5 of the PTES model. Other models and methodologies designed specifically for penetration testing within the domain of IoT have recently been proposed [13, 12, 31, 51].

2.2.2 **Autonomous Penetration Testing**

As discussed, penetration tests are traditionally performed manually which makes them time consuming, expensive and potentially unreliable results. Autonomous penetration tests would solve these issues and be beneficial for development, production, certification, education and research. While autonomous tools exist, they are

either used for a specific purpose within penetration testing, require some level of human expert interaction or are not as capable as a human penetration tester. The main reason for automated penetration testing still being an open challenge comes from the large action space in which such an agent would operate [75]. The task becomes even more difficult when accounting for IoT devices due to their different applications and heterogeneous environments. The current state of art and related work will be discussed in more detail in Section 2.5.

2.3 Wi-Fi

Wireless Fidelity, commonly abbreviated to Wi-Fi, is a wireless communication technology based on the IEEE 802.11 standards for Local Area Network (LAN) [14]. Wi-Fi allows devices equipped with a wireless Network Interface Card (NIC) to act as clients, connecting to a local Access Point (AP) over the air interface. The AP will often provide Internet access to its clients, but it may also simply create a local network in which the stations can communicate. The clients and APs in a LAN are called *stations*.

To distinguish individual networks in a Wireless LAN (WLAN), the term Service Set Identifier (SSID) is often used [32]. This describes the network name which is usually how a Wi-Fi user will address and identify networks. In this thesis we will refer to the term Extended SSID (ESSID) for a network name to clearly separate it from the Basic SSID (BSSID) which describes individual APs MAC addresses. ESSID technically describes the set of all BSSIDs in a network, but for the practical context of this thesis, the terms SSID and ESSID can be considered equivalent. The term ESSID is also employed by the case study software tools discussed in Chapter 4.

Some IoT devices are connected to the Internet through a Wi-Fi connection. Other devices may run their own Wi-Fi network to provide Internet access, like wireless routers, mobile hotspots, or to transmit data to its clients, like the IoT device that will be used in this project.

2.3.1 Encryption Standards and WPA2 Personal

There are several generations of encryption schemes which can be employed on a Wi-Fi connection. WPA version 3 is the most recent and secure [52]. Still, many APs run on WPA2 which has several vulnerabilities associated to it [42, 69, 62]. WPA2 is based on Institute of Electrical and Electronics Engineers (IEEE) 802.11i and was ratified in 2004 [41, 16]. It can be further differentiated between its modes *Personal* and *Enterprise*, where the latter has a few additional security features. Enterprise mode utilizes an authentication server which administers individual session keys, and is designed for larger networks of upwards of 10 stations [50]. The WPA2 Personal is

sometimes referred to as Pre-Shared Key (PSK) mode because the AP authenticates the user and encrypts the connection based on a single key known by all stations in the network [42].

A 2022 study suggested that private WLANs are predominantly encrypted using WPA2 or less secure protocols, four years after the publication of WPA3 [20]. When probing 21,345 WLANs in Cyprus, they found that only 13 networks employed WPA3. 74.7% used WPA2 or WPA2/WPA in mixed mode meaning that devices will choose WPA2 if they support it. Only a 0.2% used WPA or the older Wired Equivalent Privacy (WEP), while 25.1% had no encryption. Similar studies from 2019 show comparable results in Romania and Bulgaria where WPA2 was distinctly the most popular protocol [40, 71]. While these results may be slightly different in other countries, they show that WPA2 is still very common and that we can expect millions of devices to communicate over this encryption scheme for several years forward.

2.3.2 The 4-Way Handshake

WPA2 Personal encryption is based on Counter Mode with Cipher Block Chaining Message Authentication Code Protocol (CCMP) with the Advanced Encryption Standard (AES) block cipher [42, 41]. Unicast messages are encrypted using a Pairwise Transient Key (PTK) while a Group Temporal Key (GTK) is used for multicast and broadcast messages on the network. The process of authenticating and negotiating these keys is known as the *4-way handshake*. Its name comes from the four Extensible Authentication Protocol over LAN (EAPOL) messages which constitute the process, transmitted between the client and AP as seen in 2.1. The roles are often labeled *supplicant* and *authenticator* during the handshake [41], but we will stick to the terms client and AP as they are used throughout the context of this project.

Upon finishing the initial connection process, the handshake authentication is begun. Both parties derive the Pairwise Master Key (PMK) from the PSK which is distributed out of band. The first message of the handshake contains a random 128-bit value called *ANonce* which is sent from the AP to the client. The client generates its own random value called *SNonce*. The client now knows the two random values, the PMK and the BSSIDs which are the MAC addresses of the two stations. With this information, the client can calculate the PTK. The client then transmits the SNonce with a Message Integrity Code (MIC) created using the PTK. The AP can now also calculate the PTK, and then utilize it to verify the MIC. If validated, a third EAPOL message is sent from the AP. This contains a confirmation of the PTK, a GTK encrypted with the PTK, along with a MIC. Finally, the client replies with a MIC, completing the 4-way handshake.

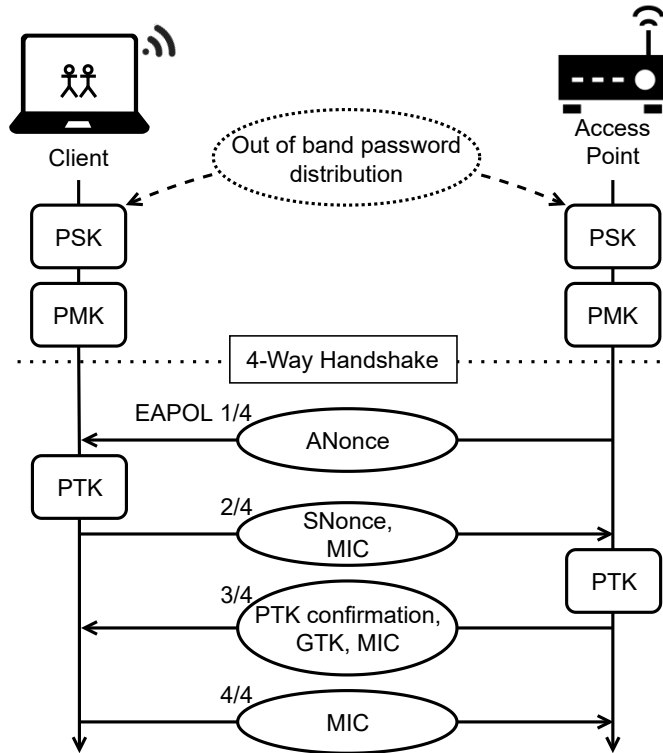


Figure 2.1: The 4-way handshake

2.3.3 Deauthentication Frames

A type of management frames called the *deauthentication* frame is used to instruct a station to drop its network connection [36]. Usually, the station will automatically attempt reconnect which initiates a new handshake process. However, if a client continuously receives deauthentication frames from the AP, it will not be able to establish a new connection. This constitutes a serious concern because in the IEEE 802.11i standard, which WPA2 is based on, the stations do not authenticate or encrypt management and control frames [16]. The frames are sent in plain-text, meaning that an adversary can forge a deauthentication frame which appears to be from an AP in order to kill the connection to a client [36]. The adversary can then capture the handshake process upon reconnection, or continue to send the deauthentication frames. The latter would act as a Denial of Service (DoS) attack by withholding the client from reconnecting.

In 2009, the IEEE published the 802.11w version which addressed the lacking protection of management frames [15]. With these changes, management frames

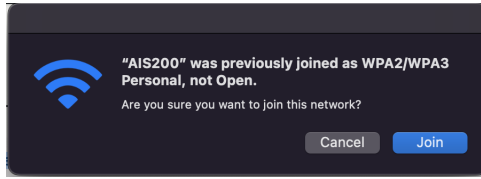


Figure 2.2: Warning issued when attempting connection to a rogue AP with no encryption scheme

should be protected if possible. To facilitate these security aspects, the third messages of the WPA handshake would contain an encrypted Integrity Group Temporal Key (IGTK) along with the GTK. The changes to the 802.11 version allows a station, i.e. a client, to drop the deauthentication frame if an IGTK check fails. Essentially, this means that an adversary that does not know the IGTK can not deauthenticate a station by simply spoofing the BSSID of the other connected station.

2.3.4 Evil Twin Attack

If a client is disconnected from its AP, it will usually attempt to reconnect automatically. Originally, the client would only check if the ESSID of the network it disconnected from matches any networks within range, and attempt to reconnect if it finds a match[6]. If there are more than one network with the same ESSID, it will connect to the one with the strongest signal. This is a severe vulnerability because the ESSID is visible to anyone within range, making it trivial to create a matching Wi-Fi network. Impersonating an AP to target unaware clients is called an *Evil Twin attack* and it can be classified under the serious category of MitM attacks.

Because of severity of this attack, many new devices will now examine the encryption scheme of the AP. If the ESSID of the network matches a previously associated network, but the encryption scheme is different or removed, it will not connect automatically. Furthermore, if attempted to connect manually, the device will issue a warning as shown in Figure 2.2. Because of this, a successful Evil Twin attack on an encrypted network will usually require knowledge of the encryption key used.

2.4 The EP Model and Formal Specification

The Execution Plan model was developed by Yamin & Katt to specify and describe the decision making process of autonomous agents within a cyber range [76]. The model has a tree structure with three levels. The tree translates the high-level goals of the agent into concrete commands for the agent to perform. The results

when running EP model are either *plan fulfilled*, *plan not fulfilled*, or *plan maybe impractical*.

Level 1 contains the root node of the tree. This describes the main goal of the agent. Connected to the main goal and within the first level are one or more branches describing sub-goals. These are separated by the logical operators \wedge (*and*) and \vee (*or*), which determine whether one or all sub-goals must be achieved in order for its parent goal to succeed.

Level 2 describes *actions* and *conditions* which will eventually decide what commands will be executed. These conditions are Boolean, meaning they can only be answered with *yes/true* or *no/false*. As a consequence, the \vee operator is the only allowed operator within this level. Each root node in the second level has its own sub tree which corresponds to a sub-goal in the upper level. Each of the second level leaf nodes are either a "Not fulfilled"-node or an actions-node. The former has no children and implies that the conditions to fulfill the plan have not been met, while the latter has one or more children in the next level.

Level 3 contains a single layer of nodes which describes concrete commands. These commands execute the actions described by the parent nodes in the layer above. The siblings within the third layer are separated by either the \wedge operator, which implies that all commands must be executed for the plan to be fulfilled, or the \vee operator, implying that either of them is sufficient. If a command is not executed successfully, the model indicates that the plan may be impractical.

2.4.1 Formal Specification and Verification

Formal specification within computer science is used to describe system requirements and function through abstraction and mathematics [25]. This is useful to aid with the design and testing of the system before, during or even after development. Formal specification can reduce redundant and ambiguous specifications and facilitate the development of more effective code with less errors [9]. While code verification like unit tests can demonstrate that code modules work as intended, the tests may not detect logical flaws in the system design. By abstracting the software and its elements, developers can verify that the proposed design works as intended before writing the code and its details. The EP model can and should be formally verified to ensure that its logic is correct and the final states can be reached. For the purpose of verifying the EP model within this project, the formal specification language called TLA+ will be used.

2.4.2 TLA+

A formal model written in TLA+ is called a *specification* [37]. The specification is a mathematical interpretation and abstraction of the discrete events of a system, which in this case will be the decision making process of the agent. The fundamental elements of a specification are the *states*. A state describes the variables of a system at a specific point in time during system operation. The transition from one state to the next is often what would be described as a program event. In TLA+, this transition is called a *step*. The full system execution will be represented by a specific sequence of states where the first state is the *init* state. The init state represents the initial condition, and all successive steps must adhere to the rules described by the *next-state relation*. By verifying a specification, the TLC Model Checker evaluates all possible states that can be reached beginning in the init state. The state transition diagram displays all reachable states as nodes, while their sequence is described by directed edges between them.

2.5 Related Work

As mentioned in Section 2.2, a number of tools which automates parts or most of the penetration testing already exists. Examples of partial automation tools include Nmap which can be used to scan open ports and identify services on a given system [39], and SQLmap which can test for SQL injection vulnerabilities and more [19]. Proprietary tools like Metasploit Pro [43] and Nessus [65] have automated vulnerability scanning and exploitation capabilities. However, these tools require some level of interaction with a human penetration tester.

Numerous attempts have been made at fully automating penetration testing. In the current state of research, many are attempting to solve the issue by employing AI and specifically machine learning techniques. The first hurdle to overcome when developing such solutions is the vast action space in which the agents must train and operate.

In 2021, a cyber security researcher and former penetration tester developed and trained an autonomous penetration testing agent using deep Reinforcement Learning (RL) algorithms [10]. Modules within a penetration testing software called Metasploit were used to create an abstraction of the action space. Within a test bed containing a highly vulnerable machine, the agent was able to acquire root privileges in all test runs. However, a control agent using the same modules were able to reach this access level in more than 20% of the test runs when randomly selecting attack vectors.

Similar work has been done by Schwartz & Kurniawati [57], Zennaro & Erdod [77] and Hu et al. [27], which involved various Q-learning algorithms to train RL penetration testing agents. Another suggestion proposed by Tran et al. investigates

an algebraic approach to structure the action space hierarchically [68]. Within this abstraction, they trained the penetration testing agent using deep RL algorithms. Their experiments showed positive results when compared to the deep Q-learning approach.

While the machine learning methods have shown positive results, even in large action spaces, the environments in which they have been trained and tested are still limited in scope of all computing systems. Especially, when accounting for all the various IoT devices and their applications. Moreover, new services and vulnerabilities are discovered every day. Limited research has been done on how these agents adapt and perform in volatile environments.

Another issue comes from the risk of damaging systems as penetration tests are often performed on live services and systems. A human penetration tester would both consider the implications of an exploit, and only execute it on a single device as a proof of concept if it is considered safe to perform. An autonomous agent will have less understanding of potential consequences. When a RL agent learns that exploiting a certain vulnerability provides a reward, it will attempt to perform this exploitation whenever possible.

Some work has been done related to the automated penetration testing of IoT devices specifically. Chu & Lisitsa proposed the use of a Belief-Desire-Intention (BDI) model to map the goals and plans of a penetration test to concrete actions and perceptions of the target system [13]. An autonomous agent will then make the decisions based on an AI framework called *procedural reasoning system*. Rak et al. developed an automatic threat modeling system which would describe concrete actions to manually perform the penetration test [51]. The actions and instructions were intended to be easy enough for a smart home owner without training to test the vulnerabilities. Considering IoT devices often are part of a larger network, Yadav et al. developed a penetration testing framework for analyzing both the IoT devices and the their connected network in its entirety [75].

There have been several studies related to WLAN and IoT security. Hossain et al. presented and discussed various security issues and challenges in the IoT, including WLAN connectivity [26]. In 2020, Kristiyanto et al. analyzed a deauthentication attack on a Wi-Fi connected IoT camera [36] equivalent to the DoS attack presented in Section 5.1. Verma et al. demonstrated several serious security risks for IoT devices on connected to IEEE 802.11ah WLAN networks [74]. Vanhoef & Piessens demonstrated that an adversary can force a reinstallation of the key generated during the handshake process, effectively resetting the nonce and replay counters [73]. In WPA2 PSK-CCMP, this would allow the adversary to decrypt and replay frames, but not forge new ones. In 2021, Vanhoef discovered more vulnerabilities in both design

and common implementations of all WPA versions [72]. With user interaction, an adversary uses the design vulnerability to steal sensitive data. Furthermore, Vanhoef demonstrated how the vulnerabilities could be exploited to attack IoT devices.

Over the last years, some tools that automate attacks on the Wi-Fi related vulnerabilities have been developed. Specifically, we know of the "WiFi Exploitation Framework - WEF" [18], "Airedgeddon" [70] and "Wifiphisher" [63]. These are all capable of running several attacks on WPA2 including versions of the Evil Twin attack. Yet, to the best of our knowledge, we do not think they are able to fully automate the process of the second attack presented in Section 5.1.

Chapter 3

Methodology

Hevner & Chatterjee explains DSR as the invention and creation of artifacts in order to provide knowledge concerning human issues [24, p. 5]. Furthermore, the artifacts themselves should provide value when attempting to solve the issues at hand. This project has followed the DSR principles and methodology. In this chapter, we will explain DSR methodology and how it is applied to our project.

3.1 DSR Life Cycle and Conditions

Any design follows a cyclic life cycle, and DSR is no exception. Each DSR cycle can be described by a set of steps as displayed in Figure 3.1 [24, p. 26-27]. Firstly, one has to be *aware of the problem* that should be solved. The problem itself, its reasoning, stakeholders and circumstances constitute what Hevner & Chatterjee define as the *environment* condition. The concerning theoretical aspect, related work and research significance depicts the *knowledge base*.

From the first step, a proposal should have been formed. The proposal should be worked with and refined in the second step called *suggestion*. This is where the proposed idea is described by a concrete, yet *tentative design* which presents the output of the suggestion step. Then, the tentative design is created or implemented in the *development* step which results in an *artifact*. The next step is evaluating the artifact based on the original problem and predefined requirements. The output is *performance measures* which is used in the *conclusion* step to produce the final *results*.

The results answer or solve some issue related the original problem described by the environment, while conforming to the scientific rigors and related theory within the knowledge base. Without the latter, the designed artifact does not contribute to research, and without the environment, it provides no value to human concerns. If this is not the case, the method should not reach the conclusion step, but rather

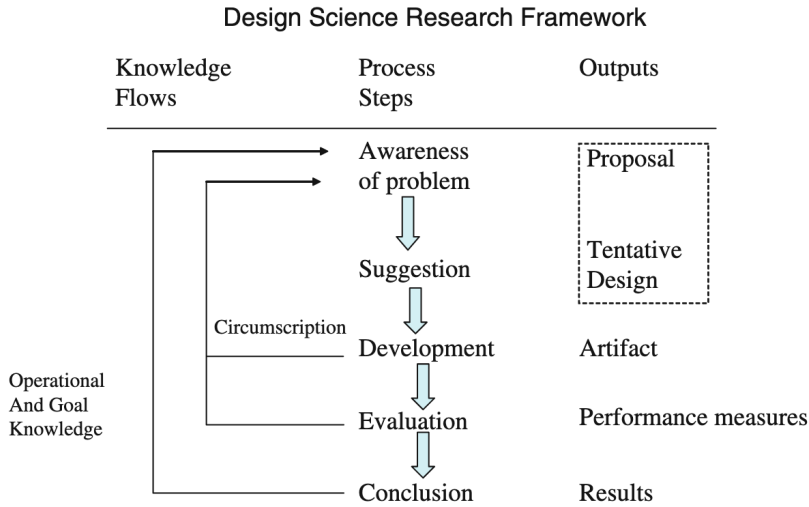


Figure 3.1: Steps in the DSR life cycle [24, p. 27]

reiterate which is cited as a *circumscription*. As implied by the nature of a cycle, the steps should be repeated until an adequate artifact is produced.

3.2 DSR Applied to the Project

This project has been through several iterations of the DSR life cycle. Regarding the research objectives, we have addressed both of the aforementioned conditions in the previous two chapters. The environment is the main topic of Chapter 1, while Chapter 2 has a greater focus on the knowledge base. From this, we first proposed an attack on the WPA2 encryption used by the IoT device provided in the case study. Through the suggestion and development steps we eventually defined the attack process as described in section 5.1. In the next iterations we developed the EP model and formal verification artifacts described in sections 5.2 and 5.3. Based on these, the agent and final artifact was implemented as described in Section 5.4. The evaluation according to the research objectives is discussed in Chapter 7, while a conclusion is found in the final chapter.

Chapter 4

Case Study and Environment

A laptop with the Kali Linux operating system was used as a basis of the attacks. Kali Linux is an open-source Debian based distribution of Linux, made as a platform which can facilitate advanced security audits and penetration tests [58]. Several software tools, including those used in the final python script, come pre-installed with Kali Linux. In addition, the distribution has a number of word lists containing commonly used passwords, which were employed to crack the WiFi keys.

4.1 Hardware

In addition to the hardware presented by a regular laptop itself, a few additional requirements must be satisfied for the attack to succeed. Most importantly a wireless NIC capable of being set to "monitor" mode is required. This NIC will serve as an interface from which most of the steps in the python script will be launched. In the final step of the client-takeover, another NIC is required to run the rogue AP. While this could be run on the wireless card with monitoring capabilities, this card is used to kill the connection between the original AP and the client while the rogue AP is running.

The laptop used was a "Lenovo ThinkPad T430 2349-EH9" with a Intel Core i5-3320M 2.60GHz CPU, and an Intel Centrino Advanced-N 6205 Dual Band NIC. The additional external USB NIC was a Linksys AE1200.

4.2 Software

All software used in the Python program, including the Python libraries which were mostly from the Python Standard library, comes pre-installed with Kali Linux. Following is an overview of the software tools used.

4.2.1 The Aircrack-ng Suite

Aircrack-ng is a suite of programs written in shell code that can be used to test the security of the most common encryption methods used in Wi-Fi today. The suite is a free, open-source, terminal run set of tools that is continuously maintained and utilized. The suite consists of several tools including Airmon-ng, Airodump-ng, Aireplay-ng, Aircrack-ng and more.

The tools can be used together to perform various actions including hardware analysis, packet monitoring and injection on the wireless interface, and performing brute-force and dictionary attacks against WEP and WPA-PSK encryption keys. It can also spoof an AP, and trick a client computer into connecting to its own rogue AP imitating the target Wi-Fi. The specific tools mentioned above were those of the Aircrack suite used in this project. Below is a description of their applications in the scope of this project.

Airmon-ng

This tool gathers and displays information about the computer network cards, interfaces and network processes. It can change the mode of the computer wireless cards to "monitor" mode which enables them to read and inject packets on the air, regardless of their source and destination. It can also kill the running network process that may interfere with the wireless interfaces while they are used by other tools. While it is not essential to run for most of the other tools to work, it creates a more stable foundation for them to operate on.

Airodump-ng

When wireless cards are operating in monitor mode, Airodump-ng is capable of sniffing any packets transmitted within range over the air interface. As a baseline, this can be used to get an overview over all APs and probes in the area. The information gathered on each node will include the ESSID, the BSSID, the encryption scheme, the channel on which it operates, its relative signal strength and its connected stations. Airodump can also be configured to narrow its scope to a single AP and its clients, for instance to capture initialization vectors or WPA handshakes. The captured data is displayed in the terminal as shown in Figure 4.1, or can be written to files.

Aireplay-ng

Aireplay-ng can be used to inject packets and spoof their source. One particular useful application in the scope of our project is to transmit deauthentication packets to clients. These packets claims to be from their connected AP, resulting in the clients dropping their connection. By continuously transmitting these packets, the client will not be able to reconnect with the AP, effectively executing a DoS attack.

```
(kali@kali)-[~]
└─$ sudo airodump-ng wlan0mon

CH 1 ][ Elapsed: 36 s ][ 2022-03-15 07:18

BSSID                PWR Beacons  #Data, #/s  CH  MB  ENC  CIPHER  AUTH  ESSID
1C:D1:E0:80:F9:22    -1      0           0  0  11  -1                <length: 0>
88:6B:0F:AB:75:83   -32     90           0  0  1  65                AIS200
1C:D1:E0:97:80:64   -55     25           0  0  1  260               WPA2 CCMP PSK <length: 0>
1C:D1:E0:97:80:61   -55     26           0  0  1  260               WPA2 CCMP MGT <length: 0>
1C:D1:E0:97:80:60   -55     26           0  0  1  260               WPA2 CCMP MGT eduroam
1C:D1:E0:97:80:62   -55     26           0  0  1  260               OPN          ntnuguest
1C:D1:E0:97:80:63   -55     24           0  0  1  260               OPN          <length: 0>
1C:D1:E0:96:93:64   -57     27           0  0  6  260               WPA2 CCMP PSK <length: 0>
1C:D1:E0:96:93:63   -57     28           0  0  6  260               OPN          <length: 0>
1C:D1:E0:96:93:62   -57     26           0  0  6  260               OPN          ntnuguest
```

Figure 4.1: Airodump-ng running in terminal

```
Aircrack-ng 1.6
[00:00:01] 197/222 keys tested (325.52 k/s)
Time left: 0 seconds                                     88.74%
KEY FOUND! [ 12345678 ]

Master Key       : FD 29 70 36 CE 56 75 F1 CD 2A F8 26 33 19 70 02
                  F5 0D 82 EA 6D BE 26 3C 90 83 0F 8C CB 51 C2 C2

Transient Key    : 10 E4 DB DF 27 8C EC DA D0 34 92 0C 68 C4 48 A3
                  73 42 02 03 99 B1 EF 00 00 00 00 00 00 00 00
                  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                  00 00 00 00 00 00 00 00 00 00 00 00 00 00

EAPOL HMAC      : 90 A2 23 C2 27 D7 C8 0E DD 09 98 AC 5C E8 68 99
```

Figure 4.2: Aircrack-ng cracking a WPA2 key

If no further deauthentication packets are transmitted, the clients will attempt to reestablish their connection through a new authentication process. During this process, a wireless sniffer like Airodump-ng may capture the handshake.

Aircrack-ng

With the same name as the tool suite itself, Aircrack-ng is used to crack WEP and WPA-PSK encryption keys. To retrieve WEP keys, several methods can be used. In terms of cracking a WPA-PSK key, Aircrack can use information captured from the four-way handshake, along with the BSSID of the AP. With this data, the tool performs a dictionary attack, testing several thousand keys each second, and outputs its result in the terminal or to a file. When a WPA2 is cracked, the output will look similar to the screenshot shown in Figure 4.2.

4.2.2 Hostapd

Hostapd, or "Host Access Point Daemon", enables communication between various 802.11 wireless access points when operating in Host AP mode [46]. It allows us to run our own AP from a wireless NIC with the name, encryption scheme and password that we specify, in addition to numerous other parameters¹. The daemon reads its configuration from a text file specified upon launch. This tool allows us to create a rogue AP that resembles the target AP enough that the client will unknowingly reconnect to it if the original connection is temporarily lost.

4.3 Target IoT Device

The IoT device used in the project is the Automatic Identification System (AIS) displayed in Figure 4.3. It is the A200 AIS Class A produced by Em-Trak [67]. An AIS is a autonomous monitoring and tracking system that communicates positional data and vessel information with nearby harbors and ships [8]. The AIS makes an operator able to see the location of other vessels in vicinity, aiding with navigation and collision avoidance. It is used on larger ships and utilities ashore like Vessel Traffic Services (VTS) for tracking, monitoring and identification. The AIS transmits data continuously and is required by international treaties to be installed and operational for larger vessels.

When mounted on a ship, the A200 AIS is connected to several sensors on the ship along with a Very High Frequency (VHF) antenna for communication with other AIS systems. It has a wireless NIC which it uses to run its own Wi-Fi AP. The Wi-Fi configuration menu is displayed in Figure 4.4. From there, the user can set the ESSID, choose internet protocol, select channel, and more. A ship operator can access and read data from the device by connecting to it with a laptop, tablet or similar device. The data is transmitted using National Marine Electronics Association (NMEA) 0183 messages which are continuously sent to all clients on the network. To read the NMEA messages, the client needs some kind of chart plotter software. There are many alternatives, but OpenCPN was used for this project [47]. The connection is protected with WPA2 Personal by default, which is the highest level of encryption the device supports. There is no password complexity validation.

¹Airbase-ng may seem to be a sufficient tool for this purpose. Unfortunately, it is unable to host a functional WPA2 encrypted AP.



Figure 4.3: The A200 connected to a battery and a VHF antenna



Figure 4.4: The Wi-Fi settings menu in A200

The Em-Trak A200 AIS Class A is an expensive AIS device used for larger vessels like freight and passenger ships, and is priced at almost £2,000 [67]. The tested device is the current version in production and market, shipped with a three year global warranty. Thus, we can expect this model to be operative in its current state in many years to come.

4.4 Physical setup

In the facilities of the Norwegian Cyber Range (NCR) we set up the testing environment. The A200 device was enabled in Wi-Fi mode with an Apple MacBook running the operating system MacOS Monterey as the client. They were located one meter apart from each other, with no physical objects between them. The laptop with Kali Linux was placed about one meter away from both the A200 and the MacBook. The AP options were left as displayed in Figure 4.4, which are the default values.

Because the AIS was not connected to all of its sensors, the messages it transmitted had no geographical data which could be displayed in the chart plotter. However, the debug window within OpenCPN displays all received NMEA messages, which was sufficient to prove the wireless connection and possible interruptions. Figure 4.5 displays a diagram of the case study setup, while Figure 4.6 shows the debug window of OpenCPN while receiving packets over a Wi-Fi connection.

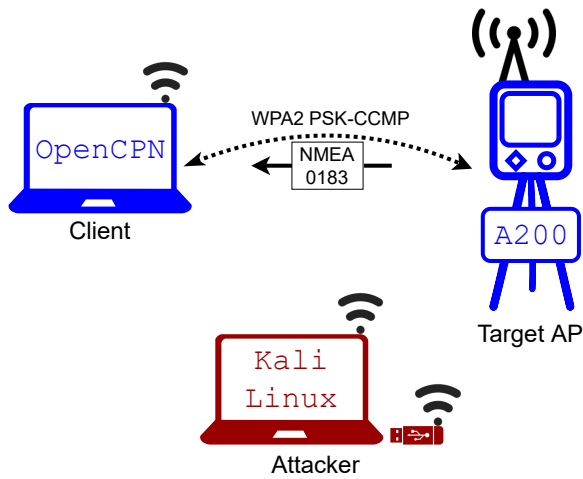


Figure 4.5: Diagram of the case study setup

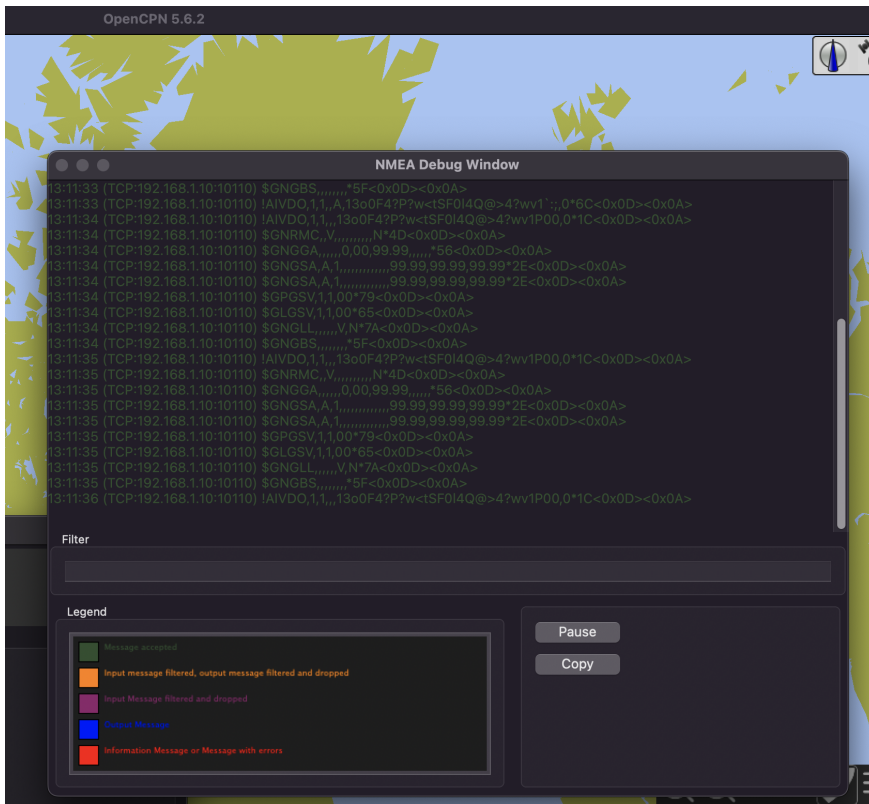


Figure 4.6: OpenCPN debug window with received NMEA packets

Chapter 5

System Design

Two similar attacks on the A200 IoT device were designed and implemented. Both are based on vulnerabilities in the WPA2 encryption scheme, targeting the network layer of the device. We have chosen these attacks because they affect all implementations of Wi-Fi using WPA2, and can be performed without extensive knowledge, expertise or equipment. This chapter will describe the attacks, define their EP models, and verify them using TLA+ and the TLC Model Checker. Finally, the verified design is implemented in Python.

5.1 The Attack Procedure

The first attack is a DoS attack using deauthentication frames, and the second is an Evil Twin attack. Figure 5.1 displays a sequence diagram of the two attacks. As evident from the diagram, the first two steps in both procedures are equal.

Attack 1: DoS

1. Capture APs

Firstly, the agent will capture all APs within range. This can be done within a couple of seconds using Airodump-ng, but should be performed on a monitoring network interface. To change the mode of NIC, the agent will use Airmon-ng.

2. Capture clients

If the target AP is found in previous step, the agent will capture the clients connected to the AP using Airodump-ng. Depending on the traffic on the network, this may take few more seconds than capturing the APs.

3. Launch attack

Provided that there are clients on the network, the agent should perform the DoS-attack. Each client will continuously receive deauthentication frames that appear to be from the AP, withholding them from the reconnecting.

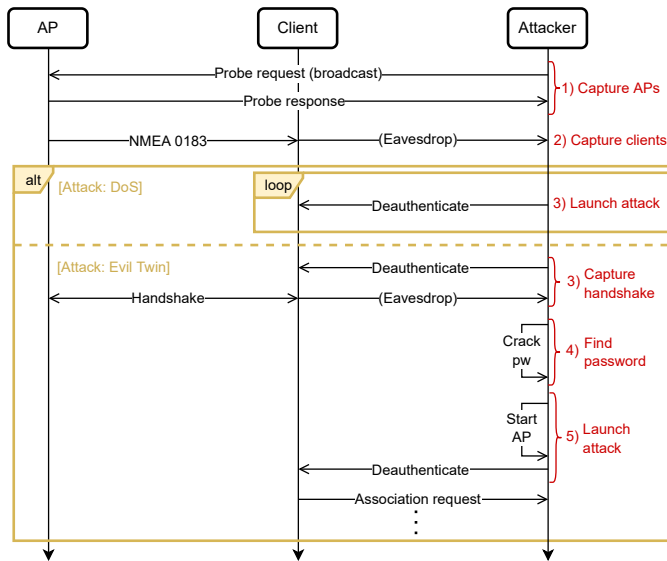


Figure 5.1: Sequence diagram of DoS and Evil Twin attack

Attack 2: Evil Twin

3. Capture handshake

This step involves capturing the handshake process between a client and the AP. By deauthenticating the client using Aireplay-ng, the agent can capture the handshake upon reconnection using Airodump-ng.

4. Find password

The password can be cracked using a dictionary attack if the nonces of the handshake was captured in the previous step. To perform the dictionary attack, Aircrack-ng will be used.

5. Launch attack

If the password is cracked the agent will spoof the network of the target AP using Hostapd. When deauthenticating the client again using Aireplay-ng, the client should automatically reconnect to the Evil Twin AP if its signal strength is stronger than that of the true AP.

5.2 EP Models of Attacks

The agent was designed based on the EP modeling discussed in Section 2.4. The diagrams displaying the high level abstraction of the DoS and Evil Twin attack models are shown in Figures 5.2 and 5.3 respectively. When running the attack,

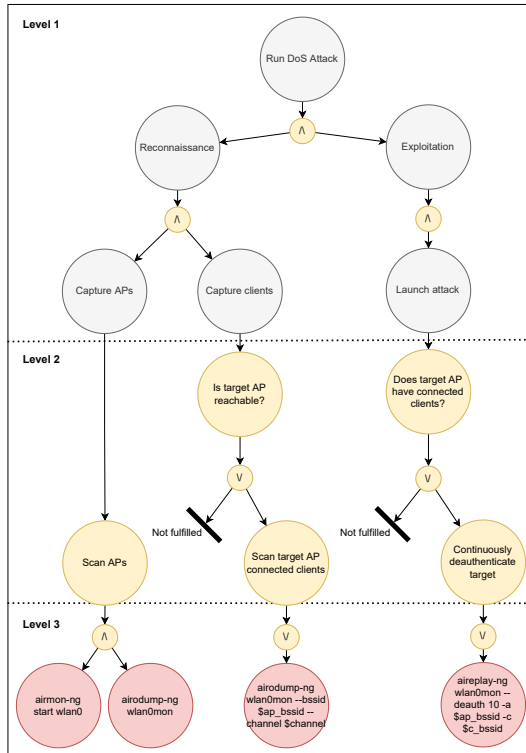


Figure 5.2: EP model of the DoS attack

the agent should analyze the main goal which is either Evil Twin or DoS, and then attempt to fulfill the sub-goals subsequently read from left to right in the model. Because both attacks involve sequential tasks where either a plan is fulfilled or the program terminates, this specific EP model is relatively simple with a single condition in the second level of each sub-goal.

5.3 Formal Implementation and Verification

In the work of Yamin & Katt, Datalog was used for formal modeling and verification of the EP. In this project, TLA+ were chosen as the formal language because it presents an approach which is based on theoretical mathematics, and thus more expressive. While Datalog is a language mainly used for database querying capable of verifying models, TLA+ was designed for modeling hardware and software at an abstract level [11, 37]. Furthermore, a software called *TLA+ Toolbox* provides both the *TLC Model Checker* which can verify the formal model, as well the possibility to create state transition diagram of the results.

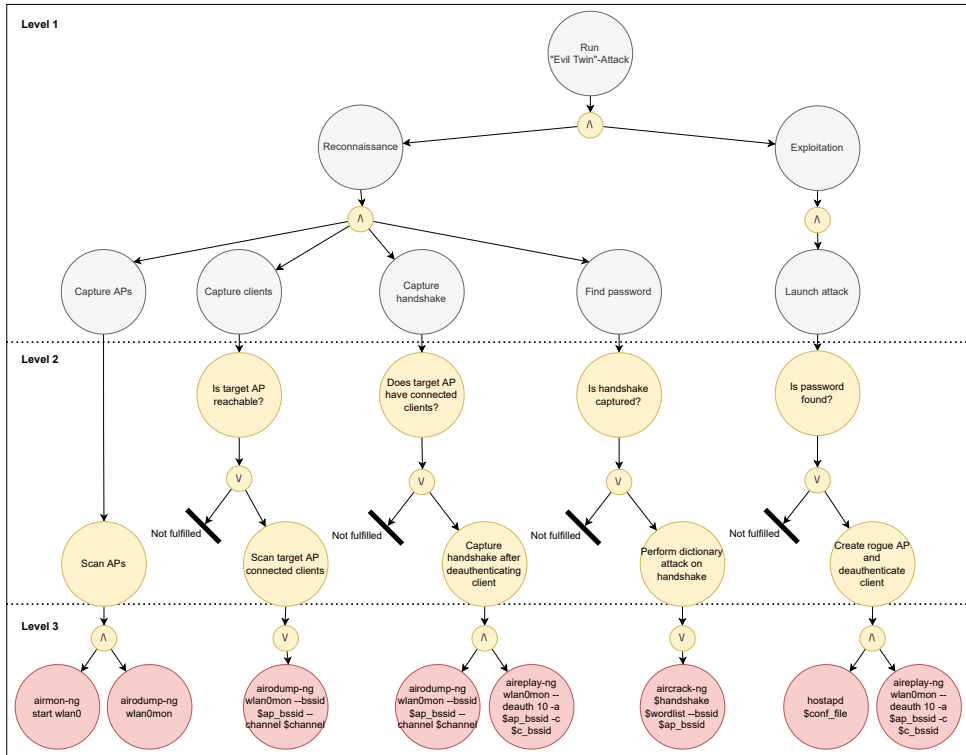


Figure 5.3: EP model of the Evil Twin attack

The model complexity for a single Evil Twin attack in this setup is trivial to the point where the formal model verification is practically unnecessary. However, its value becomes evident in complex systems where one or more agents are capable of performing many different attacks and intelligently choosing between them based on the reconnaissance phase. The formal models of the attacks written in TLA+ are presented in Figure 5.4 and Figure 5.6. Each next-state relation definition within the specification translates to a sub-goal branch of the EP model. The actions represented by the bottom node within the second layer of the model defines the variable changes of the state. For instance, the "Capture clients" sub-goal node in Figure 5.3 is described by the next-state relation "*Capture_clients*" in Figure 5.6. Within this relation definition, the condition "Is target AP reachable" of the EP model is tested by the operation "*IF found_ap = TRUE*". If this is found true, the action "Scan target AP connected clients" is performed by giving the variable "*clients*" an integer from 0 to 2, indicating the number of captured connected clients¹.

¹The max limit of clients should in theory be equal to the maximum number of possible clients

The level three commands of the EP model is not described by the specification, but by the abstraction defined in the action node of the level above. This corresponds to the TLA+ concept of abstracting the functionality, and conforms to explaining *whats* of the system, and leaving the *hows* to the lower level code implementation. In the final state of an EP where the last plan and sub-goal was fulfilled, the specification prints "Launching attack..." to illustrate a successful run. In the cases where the plan was successful until the very last state, but failed the last condition, the specification prints either "Not able to crack password" or "Not able to capture any clients" to indicate that the plan was not fulfilled.

Firstly, we wanted to formally verify that the logic of the TLA+ specifications, and consequently the EP models and agent decision making processes, were not flawed. This includes situations where the program terminates without reached a "Done" state, or where it enters an infinite loop. Secondly, we wanted to verify that the final states of the models could be theoretically reached. For verification, we used the TLC Model Checker. The model checker was able to compile and run the specifications without errors which ensures there are no logical flaw in the specifications. We could see from both the printed output of the model and generated state transition diagrams that the final states were reachable. The state transition diagrams for the DoS and Evil Twin attacks are displayed in Figures 5.5 and 5.7 respectively. When running model checker for the DoS attack, the following output was produced:

```
<<"Launching attack...">>
<<"Not able to capture any clients">>
<<"Launching attack...">>
```

These strings represent the final three possible states seen in the corresponding state diagram where the agent found the target AP. There are three of them because there are three possible values for the number of clients found in the previous state. The states in which the model found one or two clients were able to launch the attack, while the third did not. Thus, the specifications and EP models were verified.

connected to the AP, but was set to 2 because it would limit the state diagram to where it was possible to display within the thesis. During testing we also set the limit to 100 devices to ensure the validity of the model did not change.

```

MODULE DoSFormal
EXTENDS Naturals, TLC
VARIABLES found_ap, clients, pc

Init  $\hat{=}$   $\wedge$  found_ap = BOOLEAN
       $\wedge$  clients = 0
       $\wedge$  pc = "Capture_aps"

Capture_aps  $\hat{=}$   $\wedge$  pc = "Capture_aps"
              $\wedge$   $\vee$  found_ap' = TRUE
              $\vee$  found_ap' = FALSE
              $\wedge$  pc' = "Capture_clients"
              $\wedge$  UNCHANGED (clients)

Capture_clients  $\hat{=}$   $\wedge$  pc = "Capture_clients"
                  $\wedge$  IF found_ap = TRUE
                     THEN  $\wedge$  clients'  $\in$  0 .. 2
                          $\wedge$  pc' = "Launch_attack"
                     ELSE  $\wedge$  UNCHANGED clients
                          $\wedge$  pc' = "Done"
                  $\wedge$  UNCHANGED (found_ap)

Launch_attack  $\hat{=}$   $\wedge$  pc = "Launch_attack"
                  $\wedge$  IF clients > 0
                     THEN  $\wedge$  PrintT(("Launching attack..."))
                     ELSE  $\wedge$  PrintT(("Not able to capture any clients"))
                  $\wedge$  pc' = "Done"
                  $\wedge$  UNCHANGED (found_ap, clients)

Next  $\hat{=}$  Capture_aps  $\vee$  Capture_clients  $\vee$  Launch_attack

/* Modification History
/* Last modified Thu Jul 21 13:52:08 CEST 2022 by fartein
/* Created Thu May 19 13:48:07 CEST 2022 by fartein

```

Figure 5.4: The formal specification of the DoS attack EP model written in TLA+

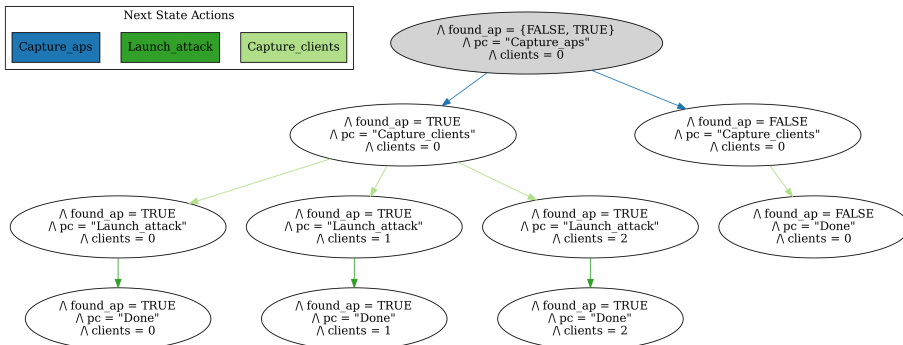


Figure 5.5: State transition diagram showing all possible states of the DoS attack EP model


```

MODULE EvilTwinFormal
EXTENDS Naturals, TLC
VARIABLES found_ap, clients, handshake, found_pw, pc

Init ≜ ∧ found_ap = BOOLEAN
      ∧ clients = 0
      ∧ handshake = BOOLEAN
      ∧ found_pw = BOOLEAN
      ∧ pc = "Capture_aps"

Capture_aps ≜ ∧ pc = "Capture_aps"
             ∧ ∨ found_ap' = TRUE
             ∨ found_ap' = FALSE
             ∧ pc' = "Capture_clients"
             ∧ UNCHANGED ⟨clients, handshake, found_pw⟩

Capture_clients ≜ ∧ pc = "Capture_clients"
                 ∧ IF found_ap = TRUE
                   THEN ∧ clients' ∈ 0..2
                       ∧ pc' = "Capture_handshake"
                   ELSE ∧ UNCHANGED clients
                       ∧ pc' = "Done"
                 ∧ UNCHANGED ⟨found_ap, handshake, found_pw⟩

Capture_handshake ≜ ∧ pc = "Capture_handshake"
                  ∧ IF clients > 0
                    THEN ∧ ∨ handshake' = TRUE
                        ∨ handshake' = FALSE
                        ∧ pc' = "Find_password"
                    ELSE ∧ UNCHANGED handshake
                        ∧ pc' = "Done"
                  ∧ UNCHANGED ⟨found_ap, clients, found_pw⟩

Find_password ≜ ∧ pc = "Find_password"
               ∧ IF handshake = TRUE
                 THEN ∧ ∨ found_pw' = TRUE
                     ∨ found_pw' = FALSE
                     ∧ pc' = "Launch_attack"
                 ELSE ∧ UNCHANGED found_pw
                     ∧ pc' = "Done"
               ∧ UNCHANGED ⟨found_ap, clients, handshake⟩

Launch_attack ≜ ∧ pc = "Launch_attack"
               ∧ IF found_pw = TRUE
                 THEN ∧ PrintT(⟨"Launching attack..."⟩)
                 ELSE ∧ PrintT(⟨"Not able to crack password"⟩)
               ∧ pc' = "Done"
               ∧ UNCHANGED ⟨found_ap, clients, handshake, found_pw⟩

Next ≜ Capture_aps ∨ Capture_clients ∨ Capture_handshake
      ∨ Find_password ∨ Launch_attack

```

```

\* Modification History
\* Last modified Wed Jul 20 17:37:51 CEST 2022 by fartein
\* Created Wed May 18 18:06:29 CEST 2022 by fartein

```

Figure 5.6: The formal specification of the Evil Twin attack EP model written in TLA+

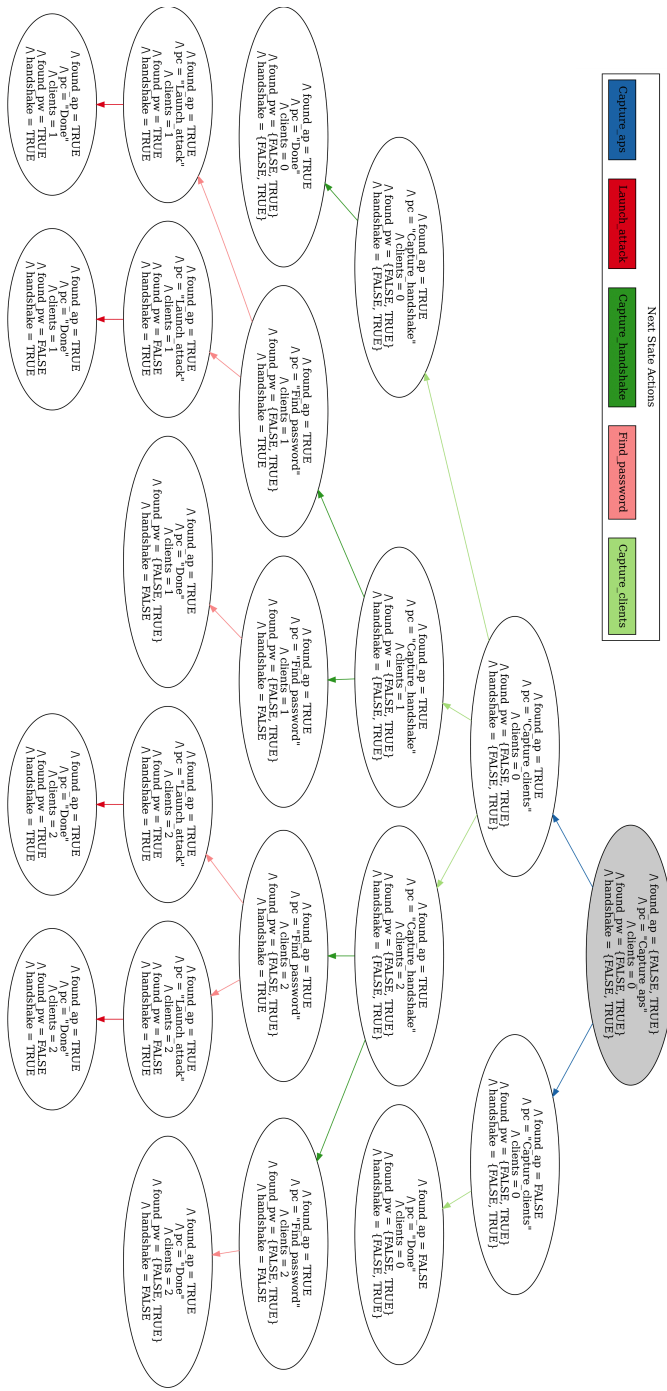


Figure 5.7: State transition diagram showing all possible states of the Evil Twin attack EP model

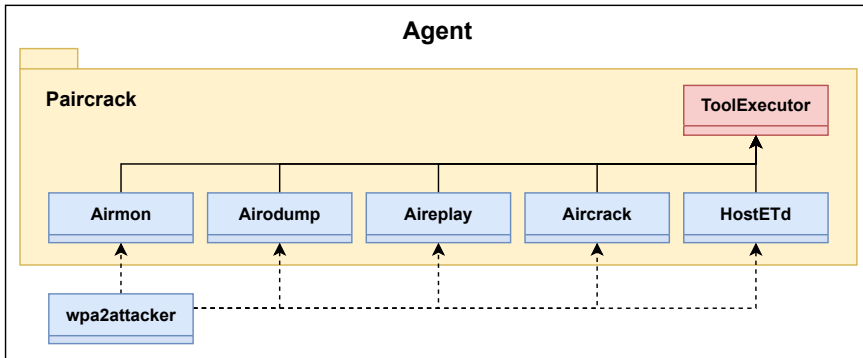


Figure 5.8: Class diagram of the agent

5.4 Implementation

The agent was implemented in Python 3.10. An essential part of the program is the *Subprocess* module from the Python Standard library [2]. The module was used to create and communicate with the processes running the software tools presented in Chapter 4.2. To ensure modifiability and applicability in other implementations, the program was written with high modularity following an object oriented approach. Figure 5.8 shows a simple class diagram of the program.

5.4.1 Tool Interface

A custom package named *Paircrack* was created and used as an interface between the agent decision making process and the software tools used. Each class within the package corresponds to a tool from the Aircrack-Ng suite or HostAPd. Because of common functionality between the package classes, an abstract class named *ToolExecutor* was created. Most importantly, this class runs and interprets the outputs of the individual tool processes. In Figure 5.1, the function for scanning and capturing APs is presented. This function uses the helper function `__capture` which calls the `run` function of the *ToolExecutor* class. All classes in the *Paircrack* package call the `run` function to start its processes. A part of this function which uses the *Subprocess* module is displayed in Figure 5.2. Because some processes run until they are manually interrupted, like those started by Airodump-Ng, a process timeout ensures their eventual termination.

There already exists a Python package for running Aircrack-Ng programs named *Pyrcrack* [21], and it serves many of the purposes we have implemented in *Paircrack*. While originally intended to be used as the tool interface and foundation for the agent, the package was discarded due to limited documentation and difficulties of use. *Pyrcrack* is still under development, which may make it applicable in future

Source code 5.1 The function for capturing all APs within range

```
def capture_aps(self, interface : str, proc_timeout=2) -> str:
    """Captures all access points within range

    Parameters
    -----
    interface : str
        Interface to capture packets on
    proc_timeout : int, optional
        Amount of seconds to run the capture, by default 2

    Returns
    -----
    str
        filepath to xml file containing AP data
    """

    self.logger.debug('Capturing all APs...')

    fsuffix = []
    flags = {
        '--write-interval': '1',
        '--output-format': 'netxml'}
    return self._capture(interface, fsuffix, flags, proc_timeout)
```

generations of the agent. The custom Paircrack package used in this project has some features inspired by Pycrack.

5.4.2 Agent Decision Making

The agent decision making process discussed in Chapter 5.2 was implemented in the class *wpa2attacker*. This class creates instances of the classes within the Paircrack package, and use their functionality to execute the actions described in the EP model. To determine what type of attack to run, i.e. DoS or Evil Twin, the attack type is given as program input in a YAML file. This file also contains a string which can be either the BSSID or the ESSID of the target AP. The YAML file is the only input required for the agent to run either attack. Providing the initial configuration data in a YAML file is for compatibility reasons with the framework and setup first created in the work of Yamin & Katt. By matching the interface with that of the autonomous agents on which this thesis is based, the developed agent can be used to expand on their research and work.

Source code 5.2 Part of the "run" function within the abstract ToolExecutor class

```
self.logger.debug(f'Running command: <{command}>')
if self.verbose:
    self.logger.debug(f'\tkeywords: <{proc_flags}>')

try:
    output = subprocess.run(command, **proc_flags)
except subprocess.TimeoutExpired as e:
    self.logger.debug(f'Process timeout')
    return True
else:
    if self.verbose:
        self.logger.debug(f'Captured stdout: <{output.stdout[:-1]}>')
        self.logger.debug(f'Captured stderr: <{output.stderr}>')
    return output
```

Chapter 6

Attack and Results

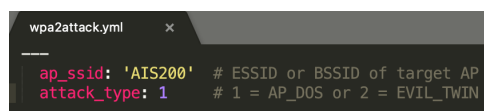
To launch an attack, the agent must know the ESSID or BSSID of the target AP. This information is provided using a YAML file. Figure 6.1 displays the contents of a YAML file for launching a DoS attack on a network named "AIS200". Below are the results of each attack.

6.1 Attack 1: DoS

The MacBook was connected to the A200 AP as described in Section 4.4, continuously receiving NMEA 0183 packets every second which were interpreted using OpenCPN. The ESSID of the network was "AIS200" which is the default setting, and the client had the BSSID "86:32:FC:1A:7C:25". The DoS attack was launched and sustained for 45 seconds, with the terminal output displayed in Figure 6.2. The agent used 18 seconds to gather the information required to launch the attack, resulting in a complete program run time of 1 minute and 3 seconds. In Figure 6.3, we can see a sudden time gap between received NMEA packets of 53 seconds. The additional 8 seconds comes from the time it took for the client to reestablish its connection.

6.2 Attack 2: Evil Twin

Again, we followed the setup previously described. The network ESSID remained unchanged, but the client BSSID was "8C:85:90:61:1F:1A" for this experiment. "12345678" was used as the network encryption password. As seen in Figure 6.4, the agent was able to successfully impersonate the A200 AP and trick the client into



```
wpa2attack.yml x
ap_ssid: 'AIS200' # ESSID or BSSID of target AP
attack_type: 1 # 1 = AP_DOS or 2 = EVIL_TWIN
```

Figure 6.1: YAML file for a DoS attack

```
(kali@kali)~[/master/code]
└─$ sudo ./wpa2attacker.py wpa2attack.yml
INFO:Attacker:Running automated WPA2 attack: AP_DOS
INFO:Attacker:Searching for target AP for 2 seconds ...
INFO:Attacker:Target AP found!
INFO:Attacker:Capturing connected clients for 10 seconds ...
INFO:Attacker:1 connected clients found!
INFO:Attacker:DoS-ing 86:32:FC:1A:7C:25
INFO:Attacker:DoS-ing 86:32:FC:1A:7C:25
INFO:Attacker:DoS-ing 86:32:FC:1A:7C:25
INFO:Attacker:DoS-ing 86:32:FC:1A:7C:25
INFO:Attacker:DoS-ing 86:32:FC:1A:7C:25
```

Figure 6.2: Performing the DoS attack

```
13:21:03 (TCP:192.168.1.10:10110) $GLOGSV1,100*65-0x0D-><0x0A>
13:21:03 (TCP:192.168.1.10:10110) $GLOGSV1,VN*7A<0x0D>-<0x0A>
13:21:03 (TCP:192.168.1.10:10110) $GLOGSV1,98<0x0D>-<0x0A>
13:21:03 (TCP:192.168.1.10:10110) $GLOGSV1,120*442*W*5F04Q@*47*W*1P00,0*1C<0x0D>-<0x0A>
```

Figure 6.3: DoS attack in OpenCPN debug window

```
File Actions Edit View Help
(kali@kali)~[/master/code]
└─$ sudo ./wpa2attacker.py wpa2attack.yml
INFO:Attacker:Running automated WPA2 attack: EVIL_TWIN
INFO:Attacker:Searching for target AP for 2 seconds ...
INFO:Attacker:Target AP found!
INFO:Attacker:Capturing connected clients for 10 seconds ...
INFO:Attacker:1 connected clients found!
INFO:Attacker:Attempting to capture WPA2 handshake
INFO:Attacker:Handshake captured!
INFO:Attacker:Attempting to crack password ...
INFO:Attacker:Password cracked: <12345678>
INFO:Attacker:Deauthenticating client to force reconnection. Attempt 1/10 ...
INFO:Attacker:Client takeover success!
```

Figure 6.4: Performing the Evil Twin attack

connecting to it. The reconnection was done automatically and transparently by the client MacBook operating system without interaction from the user. When running the attack in verbose mode, we can see the client connection in the debug logging messages as displayed in Figure 6.5. The command for deauthenticating the client using Aireplay-ng is also visible in the screenshot. The run time from start to client connection takeover was 39 seconds. A notable variable considering the result is the time it took to crack the password. Aircrack-ng used less than one second to find "12345678" in the provided wordlist. This due to the size of the wordlist which contained less than 200 words. A discussion of this matter is found in Section 7.1.

```
DEBUG:Hostetd:Launching Evil Twin AP ...
DEBUG:Hostetd:wlan1: interface state UNINITIALIZED→ENABLED
DEBUG:Hostetd:wlan1: AP-ENABLED
INFO:Attacker:Deauthenticating client to force reconnection. Attempt 1/10 ...
DEBUG:Aireplay:Running command: <['aireplay-ng', 'wlan0mon', '-deauth', '10', '-a', '88:6B:0F:AB:75:83', '-c', '8C:85:90:61:1F:1A']>
DEBUG:Hostetd:wlan1: STA 8c:85:90:61:1f:1a IEEE 802.11: associated
DEBUG:Hostetd:wlan1: AP-STA-CONNECTED 8c:85:90:61:1f:1a
DEBUG:Hostetd:wlan1: STA 8c:85:90:61:1f:1a RADIUS: starting accounting session 8B75C62FC54E70E1
DEBUG:Hostetd:wlan1: STA 8c:85:90:61:1f:1a WPA: pairwise key handshake completed (RSN)
DEBUG:Hostetd:wlan1: EAPOL-4WAY-HS-COMPLETED 8c:85:90:61:1f:1a
DEBUG:Hostetd:Client is connected: True
INFO:Attacker:Client takeover success!
```

Figure 6.5: Debug view of Evil Twin connection

Chapter 7

Discussion

This chapter will discuss the results of the case study experiment, both in terms of realism and authenticity, and in terms of the research objectives.

7.1 Authenticity of Attack

The case study was performed in a controlled environment. Because of this, we will address some important factors that could change the results of the Evil Twin attack if performed in a realistic scenario.

7.1.1 Network Interface Range

A potential limitation of the attack efficiency comes from the distance between attacker, client and AP. The client will only connect to the rogue AP if the AP signal strength is stronger than that of the authentic AP. If the signal strength of the impersonated network is weaker, the client will simply reconnect its original AP. Unless the attacking device is onboard the ship, it is likely to be further away from the client than the AIS is.

On the other hand, this issue can be addressed by either jamming the authentic AP [7], or increasing the strength of the wireless signal [30]. In this project, an old and simple network adapter was used to host the rogue AP. Even in its default signal strength configuration the client connected to it. With a new and better adapter the wireless signal strength would be stronger, which could be further increased by modifying the signal strength.

7.1.2 Time to Crack Network Password

In the case study, we used a wordlist with 200 words where we knew the AP password was one of them. Because of this, the time it took to crack the password was less than one second, which would certainly not be the case in a real scenario. The number of

Table 7.1: Comparison of key test speeds on different wordlists using Aircrack-ng and Hashcat

	Aircrack-ng, Kali machine	Aircrack-ng, Desktop	Hashcat
Crack speed (k/s)	2300	22500	1.1 million
Dictionary attack on 1.4M key wordlist (s)	10m 6s	62 seconds	1 second
Brute-force 8 integers	13 hours	75 minutes	2 minutes
Brute force 8 lowercase letters	3 years	4 months	53 hours
Brute force 8 integers and lowercase letters	40 years	5 years	30 days
Brute force 12 integers and lowercase letters	66234755 years	6770664 years	138491 years

keys tested per second on the case study computer using Aircrack-ng amounts to approximately 2300. Testing the cracking speed on desktop computer with an AMD Ryzen 5 3600 6-Core processor, we saw an increase to about 22500 keys per second. However, these numbers could be outperformed by software tool for hash cracking called Hashcat using GPU acceleration. According to a speed test on Hashcat using GPU services provided by Google Cloud, the tool can test 1.1 million WPA2 keys per second when running on Nvidia-Tesla-a100 [66].

Using these password cracking speeds to calculate the time it takes to exhaust different wordlists, we get the results as presented in Table 7.1. By using the Aircrack-ng on the Kali Linux machine, a wordlist with the same size as the famous "rockyou.txt", can be exhausted within minutes. Even a random eight integer password is feasible to crack. However, Aircrack-ng would not be able to crack a random password of integers and letters of eight characters. By using Hashcat, attacks on wordlists with millions of passwords or even random letter passwords would be possible.

```
(kali@kali)-[~/master/code]
└─$ aircrack-ng -S
2288.683 k/s
```

Figure 7.1: Aircrack-ng password cracking speed on Kali Linux machine

```
fartein@bedroompc:~$ aircrack-ng -S
22534.193 k/s
```

Figure 7.2: Aircrack-ng password cracking speed on the desktop computer

7.2 Research Objectives

In this work we investigated vulnerabilities in IoT devices, we identified Wi-Fi related vulnerabilities present in critical infrastructure. We developed an autonomous agent to test and verify them. This work will help to automatically detect Wi-Fi related vulnerabilities and create a more secure IoT empowered world. Below is a discussion of the specific research objectives based on the developed DSR artifacts and case study results.

7.2.1 Research Objective 1

We designed an autonomous agent whose decision making process was based on the EP model described. The design and model were verified using the formal language TLA+. By successfully launching the two attacks in a case study involving a currently employed IoT device, we have demonstrated that the EP model can be used to automate penetration testing. Furthermore, the script can and will be used to expand the capabilities autonomous attack agents used in the NCR cyber range.

7.2.2 Research Objective 2

The results show that we were not only able to put the target device out of service, but also spoof the connection to the client. Only considering the fact that we could easily and automatically kill all wireless connections to the device proves its security to be inadequate. As mentioned, the device is used on large vessels like yachts, cruise ships and freight ships, and international maritime law requires it to be operative at all times. Hourly docking fees, labour and maintenance costs are high for these kinds of ships. If the navigational device is put out of service for only a few hours, the associated extra costs can amount to thousands of pounds, not to mention that many of these ships are considered critical infrastructure. The particular device tested is a high end, off-the-shelf product with three year warranty, and may be employed for many more years as discussed in Section 2.1.3.

Furthermore, by being able to make the client automatically and transparently connect to a rogue AP, the issue increases in severity. In the case of AIS devices, sending fake Global Positioning System (GPS) data can lead to collisions with harbors, underwater reefs or other ships. The connection can also be misused to send malware or in other ways attack the client. In the case of routers or other devices providing Internet access, the connection can be used for MitM attacks. As IoT devices are used in anything from private homes to critical infrastructure, the fact that many of them are using insecure methods of communication and may continue to do so for several years to come, should be an alarming conclusion.

As we have discussed and seen, WPA2 is inherently vulnerable. To mitigate the vulnerabilities presented in this thesis, Wi-Fi networks should utilize the more secure WPA3. If WPA2 must be used for compatibility or other reasons, strong password policies should be employed. As displayed in Table 7.1, if the password length is sufficiently long and hard to guess, it is infeasible to crack, even with cloud provided GPU accelerated hash cracking tools.

Chapter 8

Conclusion and Future Work

This research aimed to expand and investigate the capabilities of autonomous attack agents created in the work of Yamin & Katt by both introducing new types of attacks, and using the agents to automate penetration testing on IoT devices. Based on the results of a case study following the DSR methodology, it can be concluded that the attack agents following the EP model can indeed be used for autonomous IoT penetration testing. Furthermore, we have shown that old and insecure Wi-Fi encryption protocols are still used critical infrastructure today, and likely will be for several years to come.

The agent is developed in a modular form which allows future work to easily integrate new attacks like those discussed in Section 2.5. This will make the agent able detect more vulnerabilities, increasing its usefulness for testing and verifying IoT and Wi-Fi security. Moreover it can be used in teaching and training activities in the Norwegian Cyber Range for raising awareness about IoT related attack vectors.

References

- [1] “Anna-senpai”. *Mirai forum post*. <https://hackforums.net/showthread.php?tid=5420472>. 2016.
- [2] Python 3.10. subprocess - Subprocess management. 2022. URL: <https://docs.python.org/3/library/subprocess.html> (last visited: July 20, 2022).
- [3] Manos Antonakakis, Tim April, et al. «Understanding the Mirai Botnet». In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. URL: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>.
- [4] Maximilian Appel and Ing Stephan Guenther. «WPA 3-Improvements over WPA 2 or broken again?». In: *Network 7* (2020).
- [5] Aileen G Bacudio, Xiaohong Yuan, et al. «An overview of penetration testing». In: *International Journal of Network Security & Its Applications* 3.6 (2011), p. 19.
- [6] Kevin Bauer, Harold Gonzales, and Damon McCoy. «Mitigating evil twin attacks in 802.11». In: *2008 IEEE International Performance, Computing and Communications Conference*. IEEE. 2008, pp. 513–516.
- [7] John Bellardo and Stefan Savage. «802.11 {Denial-of-Service} Attacks: Real Vulnerabilities and Practical Solutions». In: *12th USENIX Security Symposium (USENIX Security 03)*. 2003.
- [8] Shilavadra Bhattacharjee. Automatic Identification System (AIS): Integrating and Identifying Marine Communication Channels. 2021. URL: <https://www.marineinsight.com/marine-navigation/automatic-identification-system-ais-integrating-and-identifying-marine-communication-channels/> (last visited: June 12, 2022).
- [9] Marc Bourgois. «Advantages of Formal Specifications: A Case Study of Replication in Lotus Notes». In: *Formal Methods for Open Object-based Distributed Systems*. Ed. by Elie Najm and Jean-Bernard Stefani. Boston, MA: Springer US, 1997, pp. 231–244. URL: https://doi.org/10.1007/978-0-387-35082-0_17.
- [10] Sharon Caldwell. «Training an Autonomous Pentester with Deep RL». In: *Strange Loop Conference 2021*. St. Louis, MO: Strange Loop, Oct. 2021. URL: <https://www.youtube.com/watch?v=EiI69BdWKP&t>.
- [11] Stefano Ceri, Georg Gottlob, Letizia Tanca, et al. «What you always wanted to know about Datalog(and never dared to ask)». In: *IEEE transactions on knowledge and data engineering* 1.1 (1989), pp. 146–166.

- [12] Chung-Kuan Chen, Zhi-Kai Zhang, et al. «Penetration testing in the iot age». In: *computer* 51.4 (2018), pp. 82–85.
- [13] Ge Chu and Alexei 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)*. IEEE, 2018, pp. 1479–1484.
- [14] Cisco. What is Wi-Fi. 2022. URL: <https://www.cisco.com/c/en/us/products/wireless/what-is-wifi.html> (last visited: June 4, 2022).
- [15] IEEE Computer Society LAN/MAN Standards Committee et al. «IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications». In: *IEEE Std 802.11[^]* (2007).
- [16] IEEE Computer Society LAN/MAN Standards Committee et al. «IEEE Standard for information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements». In: *IEEE Std 802.11i-2004* (2004), pp. 1–190.
- [17] Andrei Costin and Jonas Zaddach. «Iot malware: Comprehensive survey, analysis framework and case studies». In: *BlackHat USA 1.1* (2018), pp. 1–9.
- [18] D3Ext. WiFi Exploitation Framework. <https://github.com/D3Ext/WEF>. 2022.
- [19] Bernardo Damele and Miroslav Stampar. SQLmap. 2006. URL: <https://sqlmap.org/>.
- [20] Victor Ojong Etta, Arif Sari, et al. «Assessment and Test-case Study of Wi-Fi Security through the Wardriving Technique». In: *Mobile Information Systems 2022* (2022).
- [21] David Francos. Pyrcrack Python package. <https://github.com/XayOn/pyrcrack>. 2020.
- [22] Jerry Gamblin. Mirai GitHub upload. <https://github.com/jgamblin/Mirai-Source-Code>. 2016.
- [23] Grant Hernandez, Orlando Arias, et al. «Smart nest thermostat: A smart spy in your home». In: *Black Hat Briefings USA 2015*. Las Vegas, NV: Black Hat, Aug. 2014. URL: <https://blackhat.com/docs/us-14/materials/us-14-Jin-Smart-Nest-Thermostat-A-Smart-Spy-In-Your-Home-WP.pdf>.
- [24] Alan Hevner and Samir Chatterjee. «Design science research in information systems». In: *Design research in information systems*. Springer, 2010, pp. 9–22.
- [25] Robert M Hierons, Kirill Bogdanov, et al. «Using formal specifications to support testing». In: *ACM Computing Surveys (CSUR)* 41.2 (2009), pp. 1–76.
- [26] Md Mahmud Hossain, Maziar Fotouhi, and Ragib Hasan. «Towards an analysis of security issues, challenges, and open problems in the internet of things». In: *2015 IEEE world congress on services*. IEEE, 2015, pp. 21–28.

- [27] Zhenguo Hu, Razvan Beuran, and Yasuo Tan. «Automated penetration testing using deep reinforcement learning». In: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2020, pp. 2–10.
- [28] Imperva. Advanced Persistent Threat (APT). 2022. URL: <https://www.imperva.com/learn/application-security/apt-advanced-persistent-threat/> (last visited: May 10, 2022).
- [29] Jabraeil Jamali, Bahareh Bahrami, et al. *Towards the internet of things*. Springer, 2020.
- [30] JavaRockstar. What is Wi-Fi. 2017. URL: <https://hackingvision.com/2017/02/18/increasing-wifi-tx-power-signal-strength-in-linux/> (last visited: July 23, 2022).
- [31] Rahul Johari, Ishveen Kaur, et al. «Penetration Testing in IoT Network». In: *2020 5th International Conference on Computing, Communication and Security (ICCCS)*. IEEE. 2020, pp. 1–7.
- [32] Juniper. Understanding the Network Terms SSID, BSSID, and ESSID. 2018. URL: https://www.juniper.net/documentation/en_US/junos-space-apps/network-director4.0/topics/concept/wireless-ssid-bssid-ssid.html (last visited: June 17, 2022).
- [33] Kaspersky. Pushing the limits: How to address specific cybersecurity demands and protect IoT. 2022. URL: <https://www.kaspersky.com/blog/iot-report-2022/> (last visited: May 29, 2022).
- [34] Constantinos Kolias, Georgios Kambourakis, et al. «DDoS in the IoT: Mirai and other botnets». In: *Computer* 50.7 (2017), pp. 80–84.
- [35] Nicholas Kolokotronis and Stavros Shiaeles. «Cyber-Security Threats, Actors, and Dynamic Mitigation». In: (2021).
- [36] Yogi Kristiyanto and Ernastuti Ernastuti. «Analysis of deauthentication attack on ieee 802.11 connectivity based on iot technology using external penetration test». In: *CommIT (Communication and Information Technology) Journal* 14.1 (2020), pp. 45–51.
- [37] Leslie Lamport. A High-Level View of TLA+. 2021. URL: <http://lamport.azurewebsites.net/tla/high-level-view.html> (last visited: June 28, 2022).
- [38] Neal Leavitt. «Researchers fight to keep implanted medical devices safe from hackers». In: *Computer* 43.8 (2010), pp. 11–14.
- [39] Gordeon Lyon. Nmap. 1997. URL: <https://nmap.org/>.
- [40] Máté Marácz. «Wardriving in Eger». In: *2019 IEEE 13th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. IEEE. 2019, pp. 000127–000130.
- [41] Moffat Mathews and Ray Hunt. «Evolution of wireless LAN security architecture to IEEE 802.11 i (WPA2)». In: *Proceedings of the Fourth IASTED Asian Conference on Communication Systems and Networks, AsiaCSN*. Vol. 7. 2007, pp. 292–297.
- [42] Kyle Moissinac, David Ramos, et al. «Wireless encryption and WPA2 weaknesses». In: *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE. 2021, pp. 1007–1015.

- [43] H. D. Moore. Metasploit. 2003. URL: <https://www.metasploit.com/>.
- [44] Daniel Oberhaus. This Hacker Showed How a Smart Lightbulb Could Leak Your Wi-Fi Password. 2019. URL: <https://www.vice.com/en/article/kzdwp9/this-hacker-showed-how-a-smart-lightbulb-could-leak-your-wi-fi-password> (last visited: Oct. 13, 2021).
- [45] US Department of the Interior Office of Chief Information Officer. Penetration Testing. 2022. URL: <https://www.doi.gov/ocio/customers/penetration-testing> (last visited: May 20, 2022).
- [46] OpenBSD. Hostapd. 2022. URL: <https://man.openbsd.org/hostapd.8> (last visited: July 12, 2022).
- [47] OpenCPN. OpenCPN Official Webstie. 2022. URL: <https://www.opencpn.org/> (last visited: July 10, 2022).
- [48] Oracle. What is IoT? 2022. URL: <https://www.oracle.com/in/internet-of-things/what-is-iot/> (last visited: Mar. 26, 2022).
- [49] Penetration Testing Execution Standard. 2009. URL: http://www.pentest-standard.org/index.php/Main_Page (last visited: Nov. 10, 2021).
- [50] Tamara Radivilova and Hassan Ali Hassan. «Test for penetration in Wi-Fi network: Attacks on WPA2-PSK and WPA2-enterprise». In: *2017 International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo)*. IEEE. 2017, pp. 1–4.
- [51] Massimiliano Rak, Giovanni Salzillo, and Claudia Romeo. «Systematic IoT Penetration Testing: Alexa Case Study.» In: *ITASEC*. 2020, pp. 190–200.
- [52] B Indira Reddy and V Srikanth. «Review on wireless security protocols (WEP, WPA, WPA2 & WPA3)». In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* (2019), pp. 28–35.
- [53] Ian J.H. Reynolds. IoT Architecture. 2020. URL: <https://www.zibtek.com/blog/iot-architecture/> (last visited: Mar. 5, 2022).
- [54] Chris Rose et al. «The Security Implications Of The Internet Of Things». In: *Journal of Cybersecurity Research (JCR)* 2.1 (2017), pp. 1–4.
- [55] Spyridon Samonas and David Coss. «The CIA strikes back: Redefining confidentiality, integrity and availability in security.» In: *Journal of Information System Security* 10.3 (2014).
- [56] Eryk Schiller, Andy Aidoo, et al. «Landscape of IoT security». In: *Computer Science Review* 44 (2022), p. 100467.
- [57] Jonathon Schwartz and Hanna Kurniawati. «Autonomous penetration testing using reinforcement learning». In: *arXiv preprint arXiv:1905.05965* (2019).
- [58] Offensive Security. What is Kali Linux. 2022. URL: <https://www.kali.org/docs/introduction/what-is-kali-linux/> (last visited: June 10, 2022).
- [59] Aleatha Shanley and Michael N Johnstone. «Selection of penetration testing methodologies: A comparison and evaluation». In: (2015).

- [60] Satyajit Sinha. State of IoT 2021. 2021. URL: <https://iot-analytics.com/number-connected-iot-devices/> (last visited: Sept. 29, 2021).
- [61] Rob Sobers. 134 Cybersecurity Statistics and Trends for 2021. 2021. URL: <https://www.varonis.com/blog/cybersecurity-statistics> (last visited: May 22, 2022).
- [62] Mukhtar Ahmad Sofi. «Bluetooth Protocol in Internet of Things (IoT), Security Challenges and a Comparison with Wi-Fi Protocol: A Review». In: *International Journal of Engineering and Technical Research* 5 (2016).
- [63] Open source. Wifiphisher. 2017. URL: <https://wifiphisher.org/> (last visited: July 1, 2022).
- [64] Deris Stiawan, Mohd Yazid Idris, et al. «Cyber-Attack Penetration Test and Vulnerability Analysis». In: *International Journal of Online Engineering* 13.1 (2017).
- [65] Inc. Tenable. Nessus. 2005. URL: <https://www.tenable.com/products/nessus>.
- [66] David Tomaschik. GPU Accelerated Password Cracking in the Cloud: Speed and Cost-Effectiveness. 2021. URL: <https://systemoverlord.com/2021/06/05/gpu-accelerated-password-cracking-in-the-cloud.html> (last visited: July 14, 2022).
- [67] Em-Trak. A200 AIS Class A. 2022. URL: <https://em-trak.com/products-a200/> (last visited: July 7, 2022).
- [68] Khuong Tran, Ashlesha Akella, et al. *Deep hierarchical reinforcement agents for automated penetration testing*. 2021.
- [69] Achilleas Tsitroulis, Dimitris Lampoudis, and Emmanuel Tsekleves. «Exposing WPA2 security protocol vulnerabilities.» In: *Int. J. Inf. Comput. Secur.* 6.1 (2014), pp. 93–107.
- [70] v1s1t0r. Airgeddon. <https://github.com/v1s1t0r1sh3r3/airgeddon>. 2022.
- [71] H Valchanov, J Edikyan, and V Aleksieva. «A study of Wi-Fi security in city environment». In: *IOP Conference Series: Materials Science and Engineering*. Vol. 618. 1. IOP Publishing, 2019, p. 012031.
- [72] Mathy Vanhoef. «Fragment and Forge: Breaking Wi-Fi Through Frame Aggregation and Fragmentation». In: *Proceedings of the 30th USENIX Security Symposium*. USENIX Association, Aug. 2021.
- [73] Mathy Vanhoef and Frank Piessens. «Key reinstallation attacks: Forcing nonce reuse in WPA2». In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1313–1328.
- [74] Shikhar Verma, Yuichi Kawamoto, and Nei Kato. «A network-aware Internet-wide scan for security maximization of IPV6-enabled WLAN IoT devices». In: *IEEE Internet of Things Journal* 8.10 (2020), pp. 8411–8422.
- [75] Geeta Yadav, Kolin Paul, et al. «IoT-PEN: an E2E penetration testing framework for IoT». In: *Journal of Information Processing* 28 (2020), pp. 633–642.
- [76] M. Mudassar Yamin and Bass Katt. «FUse of Cyber Attack and Defense Agents in Cyber Ranges: A Case Study.» In: NTNU, 2022.

- [77] Fabio Massimo Zennaro and Laszlo Erdodi. «Modeling penetration testing with reinforcement learning using capture-the-flag challenges: trade-offs between model-free learning and a priori knowledge». In: *arXiv preprint arXiv:2005.12632* (2020).

