BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

PROTOTYPE OF INTRUSION SOLUTION FOR MOBILE NETWORKS

PROTOTYP INTRUSIVNÍHO ŘEŠENÍ PRO BEZDRÁTOVÉ SÍTĚ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **TIMOTEJ KAMENSKÝ**
AUTOR PRÁCE

SUPERVISOR **Ing. VLADIMÍR VESELÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2022

Brno University of Technology
Faculty of Information Technology

Department of Information Systems (DIFS) Academic year 2021/2022

Bachelor's Thesis Specification

Student: **Kamenský Timotej**
Programme: Information Technology
Title: **Prototype of Intrusion Solution for Mobile Networks**
Category: Networking

Assignment:

1. Learn about existing MitM proxy solutions (such as WiFi Pineapple, ESP32), IMSI catchers.
2. Study the currently applicable attacks on devices in wireless networks (e.g., Karma attack, SSL/TLS split) and applications to implement them (e.g., aircrack-ng, nmap, Wireshark).
3. Perform an analysis of selected vulnerabilities for mobile devices (respecting the versions of different operating systems or wireless network interface chips used).
4. Design a custom solution implementing the selected attacks with respect to its possible portable deployment.
5. Implement the prototype as recommended by the supervisor.
6. Test the prototype under realistic conditions and discuss its functionalities.

Recommended literature:

- Stehlík, R. "Útok na WiFi síť s využitím ESP32/8266." *Diplomová práce*. *Vysoké učení technické v Brně, Fakulta infromačních technologií.* Vedoucí práce Ing. Jan Pluskal. Brno, 2021.
- Dagelić, A., et al. "SSID oracle attack on undisclosed Wi-Fi preferred network lists." *Wireless Communications and Mobile Computing*. 2018.

Requirements for the first semester:

- Items 1 to 4 (included).

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Veselý Vladimír, Ing., Ph.D.**
Head of Department: Kolář Dušan, doc. Dr. Ing.
Beginning of work: November 1, 2021
Submission deadline: May 11, 2022
Approval date: October 13, 2021

Abstract

As the mobile networks rise, the importance of their security rises too. The vulnerabilities of the mobile protocols on L1-L3 are easily misused. Using a small and relatively cheap Software Defined Radio, it is possible to implement a variety of attacks aiming at mobile networks. Goal of this work is to create an intrusion tool, implementing the available attacks. These include jamming of the connection, service downgrade, Denial of Service, Location tracking, IMSI catcher.

Abstrakt

Spolu s rozvojom mobilných sietí stúpa aj dôležitosť ich zabezpečenia. Zraniteľnosti jednotlivých mobilných protokolov na vrstve L1 - L3 je možné ľahko zneužiť. S použitím malého a relatívne lacného Software Defined Radio je možné implementovať rôzne druhy útokov cieliacich na mobilné siete. Cieľom tejto práce je vytvoriť intruzívny nástroj, implementujúci dostupné útoky. Tie zahrňujú zarušenie spojenia, Zníženie kvality služieb, Denial of Service, Sledovanie polohy, IMSI catcher.

Keywords

Mobile networks interception, GSM, LTE, SDR, miniaturisation, Denial of Service, Service downgrade, radio signal jamming, location tracking, IMSI catcher

Klíčová slova

Intercepce mobilných sietí, GSM, LTE, SDR, miniaturizace, Denial of Service, Zníženie kvality služieb, zarušenie rádiového signálu, sledovanie polohy, IMSI catcher

Reference

KAMENSKÝ, Timotej. *Prototype of Intrusion Solution for Mobile Networks.* Brno, 2022. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Vladimír Veselý, Ph.D.

Rozšířený abstrakt

Spolu s rozvojom mobilných sietí stúpa aj dôležitosť ich zabezpečenia. Zraniteľnosti jednotlivých mobilných protokolov na vrstve L1 - L3 je možné ľahko zneužiť.

Na začiatku sme sa venovali prieskumu zraniteľností v rámci Wi-Fi a mobilných sietí na týchto vrstvách. Popísali sme útoky na Wi-Fi, ako sú evil twin attack či known beacons attack. Pri mobilných sieťach sme taktiež preskúmali existujúce a už popísané útoky v oblasti. Skúmali sme tiež existujúce intruzívne nástroje, ako sú napr. Wi-Fi pineapple.

V posledných dvadsiatich rokoch došlo ku veľkému posunu v oblasti tzv. Software Defined Radio (SDR). Toto rozhranie s rádiom nahrádza mnohé hardwarové súčiastky ich softwérovou implementáciou. To umožnilo ich zmenšenie, zníženie energetickej náročnosti a flexibilitu pri vývoji i prevádzke. S bežnou cenovkou okolo 1000 EUR sú aj relatívne dostupné. Pre tento typ periférii existuje široká komunita ktorá ich používa ako svoje hobby, či vo výskume. Jednou z najznámejších implementácii pre SDR sú implementácie mobilných stackov ako LTE (4G) či GSM. Pomocou nich je tak možné vytvoriť celú mobilnú sieť. Tieto riešenia sme preskúmali a porovnali sme OpenAirInterface, OpenLTE a srsRAN (srsLTE). Z týchto možností sme ako najlepšiu vyhodnotili srsRAN, s ktorej pomocou sme neskôr pokračovali vo vývoji.

Táto práca sa pokúša zreprodukovať útoky na LTE skrze vrstvy L1-L3. Cieľom bolo vytvoriť intruzívny nástroj, implementujúci dostupné útoky. Implementované útoky zahrňujú zarušenie spojenia, Zníženie kvality služieb, Denial of Service, Sledovanie polohy, IMSI catcher. Vedľajším cieľom bolo navrhnúť hardwarové riešenie, ktoré bude schopné poháňať tieto útoky a SDR, no zároveň bolo čo najviac kompaktné, napájateľné batériou po dlhú dobu, a zostavené z dostupných zariadení.

Prvým útokom je IMSI catcher, ktorý zistí International Mobile Subscriber Identity pripájajúcich sa zariadení. Na základe toho je možné teoreticky sledovať pohyb telefónu. Druhým je Denial of service. Ak sa nejaký telefón začne pripájať na našu vežu, dostane ako odpoveď zákaz sa pripájať na akúkoľvek sieť. Posedným je Downgrade attack. Ak sa nejaký telefón začne pripájať na našu vežu, dostane ako odpoveď zákaz používania tzv. EPS - čo v praxi znamená nutné použitie GSM.

Implementácia útokov bola uskutočnená vytvorením napodobeniny Mobility Management Entity (MME). Na tú sa potom pripája mobilná veža LTE (eNodeB). MME koná ako mozog celej operácie, zatiaľ čo veža zabezpečuje rádiové spojenie a preposielanie údajov MME. Ako eNodeB sme zvolili srsENB, ktoré je súčasťou srsRAN stacku. Implementované útoky fungujú vďaka skutočnosti, že na ich prijatie mobilnými zariadeniami nie je potrebné mať uskutočnenú výmenu kľúčov uložených na SIM karte. Ide o slabinu v protokole, ktorá bola v 5G sieťach zaplátaná.

Vykonali sme prieskum existujúcich hardwarových riešení v oblasti malých počítačov i SDR. Porovnali sme napríklad BladeRF, HackRF a USRP B210. Zistili sme tiež očakávanú náročnosť na PC a pripravili si sériu možností na výber. Nakoniec bola ako hardwarová platforma zvolená kombinácie počítača Raspberry Pi4B a SDR Blade-RF micro 2.0 ax9. Táto kombinácia bola otestovaná a fungovala znamenite aj pri behu na batériu (5V 3A). Táto kombinácia je tiež dostatočne kompaktná. Okrem práce na batériu sme tiež zistili, že na stabilný výkon je Raspberry Pi plne vhodné, a na prevádzku tohoto nástroja na Raspberry Pi OS stačí aj najnižšia verzia Raspberry s 2GB RAM.

Na komunikáciu medzi eNodeB a nami implementovaným MME sa používa SCTP protokol, na ktorom beží S1-Application Protocol (S1AP). S1AP tiež tuneluje správy medzi mobliným telefónom (UE) a MME pomocou NAS protokolu. Na prácu so všetkými troma

protokolmi sme použili knižnice od P1Sec, a to konkrétne Pysctp a Pycrate. Implementačným jazykom je Python.

Výsledná práca bola testovaná pomocou unit testov a predpripravených scenárov, ktorými zas simulujeme eNodeB. Po implementácii bola práca skúmaná pomocou záchytov skrze Wireshark. Hoci v priložených záznamoch vidíme priebeh správ ako sú popisované pri útokoch, mobilné telefóny na ne nedávajú ohľad. Teda hoci sme tieto útoky úspešne implementovali, intruzívny nástroj sa nám teda vytvoriť nepodarilo. Z pôvodne plánovaných útokov plne funguje len IMSI catcher. Zvolená architektúra útoku (mockovanie MME) nám tiež neumožňuje uskutočniť ďalšie existujúce útoky na mobilnú sieť.

Prototype of Intrusion Solution for Mobile Networks

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vladimíra Veselého Ph.D..

Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Timotej Kamenský
May 11, 2022

Acknowledgements

I would really like to thank my supervisor, mr. Veselý. He accepted me and my ideas, gave all the support I could have asked for, and led me in this research. Without him, I probably would not have been able to write this project topic; certainly not with this level of quality.

I would also like to present a recipe to my favourite food - pancakes. What is important to understand is that pancakes lie on a bread - cake scale. On this scale, it has very high degree of freedom to move around it. It is possible to make a mancake that is similar to a matzo (passover bread) - only water and flour. It is also possible to make pancakes that are filled with chocolate, multiple fruits and covered in whipped cream, with a topping of ground nuts. This freedom to make something the same and so different is what I really like about pancakes.

The pancakes I make most of the time are really simple. First, we mix together flour, milk, salt and a single egg. Milk-to-flour ratio is roughly 2:1, but if the mix is too dense, do not hesitate to add in a bit of water. Do NOT add sugar - this would ruin the dough. Sunflower oil is optional. After that, cook the pancakes thinly, until they are of a golden color. Serve with cut bananas, quark and nutella. Bon apetit!

Contents

Chapter 1

Introduction

In the age of information, wireless technologies form a backbone of civilisation as we know it. Mobile networks and Wi-Fi access points are ubiquitous in our everyday lives, and people rely on them with almost absolute trust. It is not just the comfort and options it brings; society relies on these networks even when lives and big money are at risk. From emergency services, railroad signalling, and high-frequency trading up to power and water utilities, internet and mobile services have become a critical infrastructure in and of themselves [quote?].

Not all wireless networks are used by the general public. The two mobile network types widely used by the public are governed by GSM and LTE standards, and the Wi-Fi network is governed by 802.11 standards respectively. While 802.11 (with applied WPA2/3 protocols) and LTE standards are widely considered safe for use (the attacker cannot read the content of encrypted messages), the fact that these networks are wireless opens the possibility for Man-in-the-Middle attacks (MitM attacks). Some methods choose passive listening to the network traffic. Others favour creating a rogue access point/base station, onto which they try to connect their victims. Nevertheless, the end goal is to secretly gain data from and about the users; location, internet traffic, phone calls, and messages.

Wireless MitM attacks are distinct from other network attacks in that they need equipment on the attack scene. The usual scenario involves the attacker personally coming to the target's proximity, with all the equipment required. This approach is not just highly time-consuming. Depending on the circumstances, this can even be dangerous or not possible at all. However, modern chips (e.g. ESP32 by Espressif) offer very high capabilities with comparably small size and low power needs. When we combine this with advances in drone technology in the last decade, the miniaturised equipment could be brought to the scene using a UAV, lowering the risk and requiring fewer person-hours from the attacking personnel.

This work strives to create a prototype of a miniaturised intrusive device for Wi-Fi and mobile networks. This device should be able to work independently for hours on end, applying a range of attacks. Based on the analysis, the device might be required to transmit results in real-time or store the data for further analysis. Moreover, the device and its batteries should be light enough to be carried by a reasonably sized drone.

In Chapter 2, we describe the state-of-the-art of MitM wireless attack, off-the-shelf solutions, usable hardware and practical software tools. We give special attention to an ESP32 by Espressif, Wi-Fi pineapple MK. V, supposed capabilities of Stingray II/Agata.

In Chapter 3, we investigate possible types of MitM attacks. We gather their prerequisites and capabilities and compare them to one another. Other than their pure functionality, we keep our attention on the weight and size of the components.

In Chapter 4, we synthesise Chapters 2 and 3, pick the hardware platform and implement the attacks. After that, we design a possible solution to the problem using the information available.

In Chapter 5, we implement the designed solution from Chapter 4. We describe its functionality and its components. Then,

In Chapter 6, we test our work, evaluate our results, work out the problems, and outline possible improvements to the prototype.

And at last, in Chapter 7 we summarise the bachelor thesis.

Chapter 2

Wi-Fi interception

This chapter will summarise the already existing solutions for wireless intrusion. We will go through the required hardware and types of attacks separately for Wi-Fi attacks and Mobile networks.

2.1 Wi-Fi Attacks

2.1.1 Required Hardware for Wi-Fi Attacks

Luckily for us, the technical requirements for Wi-Fi attacks are very affordable. The attacks that we encountered required a maximum of two adapters with a possibility of switching to monitor mode. Including the Wi-Fi card in virtually all modern computers and the cost of a secondary suitable Wi-Fi adapter at around 25€ (we used TP-Link Archer T3U Plus for 26€ [5]), the barrier for entry is almost non-existent.

Alternatively, one can obtain a Wi-Fi pineapple. With a cost of around 100€, this device has all the hardware required, with preprogrammed attacks that we can execute with just a single click.

2.1.2 Evil Twin Attack

The 802.11 standard sets out control frames. Their goal is, for example, to control the connection between Access Point and the client device. However, these control frames are usually not encrypted by any means.

The goal of the evil twin attack is to establish a MitM position. The attack goes as follows: the target client is already connected to a Wi-Fi network. The attacker creates the evil twin AP, i.e. network with the same SSID. After that, using the vulnerability of unencrypted control frames, the attacker forges deauthentication frames that look as if sent from the original network. These frames disconnect all devices connected to the network (if broadcast) or just a given device. Since the original network is de-facto jammed by the deauthentication frames, the only possibility is for the target to connect to the evil twin.

We verified the possibility of the given attack using the open-source tool EAPhammer[35] for the evil twin, combined with aireplay-ng for the deauthentication. Since EAPhammer took over the whole Wi-Fi antenna, aireplay-ng had to use a second antenna. However, a similar attack can also be replicated using Wi-Fi pineapple or other software and hardware solutions.

Source	Destination	Info
Attacker - MAC of real AP	Broadcast	Deauthentication
Attacker - MAC of real AP	Broadcast	Deauthentication
Client Device	real AP	Probe request for SSID=xyz
Attacker - MAC of real AP	Broadcast	Deauthentication
Client Device		Acknowledgement
real AP	Client Device	Probe response
Attacker - MAC of real AP	Broadcast	Deauthentication
Client Device		Acknowledgement
10.0.0.1	255.255.255.255	DHCP NAK
10.0.0.1	10.0.0.116	DHCP Offer
10.0.0.1	10.0.0.116	DHCP Ack

Table 2.1: Picked packets from Evil twin attack. The deauthentication are transmitted continuously throughout the attack.First, they disconnect the target from the original AP. Even if the device tries to reconnect to the original network, receives the deauthentication frame. That is interpreted as „cannot connect now“ and is acknowledged. The last three packets show started communication with evil twin AP.

2.1.3 KARMA Attack

When Access Point and client device want to connect, they can transmit different control frames. Both the AP and the client tried to reach each other in the past. The access point transmits Beacon frames. These are a way for AP to say, „hey, Wi-Fi is here!“. On the opposite side, the client device is actively used to probe for Wi-Fi. It transmitted Probe requests, which are a way of saying, „is there any Wi-Fi with this name?“. The probe requests are targeting the Wi-Fi networks that it has saved as known ones. Such a search of Wi-Fi by the client is called active search.

The 802.11 standard expected that only a Wi-Fi with that SSID would answer to probe requests. However, the KARMA attack exploited this baseless trust. The attacker answered every probe request while masquerading as the AP searched. After that, a connection in the MitM position could be established. It was also relatively fast since the target itself transmitted all the required information, and the attacker had just to collect it and retransmit it.

This attack has, however, lost traction since its heyday. The attack is dependent on the client devices actively transmitting probe requests. Instead, the manufacturers have switched their products to passively listening for beacon frames.

2.1.4 Known Beacons Attack

In response to an industry-wide shift from active search to a passive one, George Chatzisofroniou developed an attack against passive AP search in 2018.

The attacker first has to create a list of AP SSIDs, that the target might have used in the past and have it saved as a known network. These SSIDs can either be tailored for a target (name of home network or workplace network) or consist of popular Wi-Fi networks, usually in the services and transport. The fast-food chain, the national train carrier, or a supermarket are used by tens of thousands.

After having a collection of usable Wi-Fi SSIDs, the attacker starts transmitting beacon frames going through the list of SSIDs. If the target device has connected, the attack has been successful.

This attack can also reconstruct a list of saved networks and use this information in conjunction with any open Wi-Fi maps. In this way, we can pinpoint locations and institutions frequented by the target.

However, the attack does not prevent the target from seeing the Wi-Fi SSID to which it is connected. Depending on the circumstances, this can cause problems with the stealth of the attack. If, for example, the target is connected to Fast food Wi-Fi in the middle of a forest and notices it, every alarm bell should go off. Therefore, the attack is dependent on careful execution.

We were able to reproduce the following attack, again using a freely available tool EAPhammer [35]. The only extra parameters required were the SSIDs of the known beacons. We picked some of our area's most used Wi-Fi networks, but their choosing deserves further research.

```
./eaphammer -i wlan0 --mana -e CD_wifi \
    --known-beacons --captive-portal \
    --known-ssids ZSSK_free_wifi eduroam McDonalds_Free
```

The actions of the tool can be followed by Wireshark. What the software did was imitate the given Wi-Fi networks. The first three packets show the response of the EAPhammer to the Probe request. It answered with probe responses for each of the given networks. The last packet shows that Beacon Frames are sent for each of the known beacons. The Beacon frames are more spaced in time, but they are transmitted for each network still. The response of the attacked device was to connect to a known beacon.

Source	**Destination**	**Info**
Attacker	Device attempting to connect	Probe response SSID=ZSSK_free_wifi
Attacker	Device attempting to connect	Probe response SSID=eruroam
Attacker	Device attempting to connect	Probe response SSID=McDonalds_Free
Attacker	Broadcast	Beacon frame SSID=CD_wifi

Table 2.2: Wireshark recording showing the basic mechanic of known beacons attack. Notice that all the probe responses originated from the attacking device.

2.1.5 4-Way Handshake Capture

This attack is a classic. When there is a legitimate device connecting to the Wi-Fi network, we can capture its 4-way handshake and deduce from it the network password.

The attacker has to be in the range of the network, with monitor mode, eavesdropping on the network. If a device is connecting, the attacker saves the connection-establishing 4-way handshake. From it, the attacker can deduce every used variable for the hash calculation, except the password.

The whole thing works as follows: During the 4-way handshake, the client (supplicant) and AP (authenticator) are trying to establish encryption keys for data exchange. Notice that the protocol is built so that password is never transmitted over the air separately. It is only a part of the Pairwise Transit Key.

```
PTK = PMK + Anonce + Snonce + MAC(authenticator) + MAC(supplicant)
```

PTK = Pairwise Transit Key - final encryption key for unicast traffic (i.e. between client and AP)
PMK = Pairwise Master Key - key used for establishing PTK. When talking about WPA/WPA2 personal version, PMK is PSK (i.e. password)
Anonce = authenticator nonce, the single-use random key generated by authenticator
Snonce = supplicant nonce, the single-use random key generated by client

From the equation above, all of it is transmitted during the 4-way handshake except the PMK. Knowing all the other parameters and the resulting PTK, the attacker can brute-force the password after capturing this 4-way handshake.

Attack benefits from people using weak passwords, on which we can use dictionary attacks that significantly accelerate the process. However, the attack relies on recording the handshake of someone knowing the password. Waiting for someone to connect can take a long time. To speed it up, the attacker can de-authenticate the already connected users just so that they connect again. However, doing so can potentially expose the attack.

2.1.6 PMKID Attack

PMKID is one of the newest attacks on Wi-Fi networks. It uses roaming, a feature of networks composed of multiple Access Points. In handover from one access point to another, the 4-way handshake does not repeat. Instead, the handover uses cached values to simplify and speed it up. This data is cached in PMKSA (PMK security association). The key variable is PMK ID

```
PMKSA = PMKID + Lifetime of PMK + MAC addresses + other variables

PMKID = HMAC-SHA1-128( PMK, "PMK Name" + MAC (AP) + MAC(Supplicant) )
```

The used weakness is that the attacker can ask for a PMKID. Similarly to a 4-way handshake attack, from this value, we know all the unknown parameters except the PMK, which can be brute-forced to get the original password. Unlike the 4-way handshake, we do not need to wait for a legitimate client to connect to capture the data - this attack is „client-less“. [9]

2.2 Hardware Solutions

2.2.1 Wi-Fi Pineapple

Wi-Fi Pineapple is a purpose-built wireless auditing platform. The device comes as a pre-made package of hardware and software, intending to make penetration testing as easy as a button push. Pre-made attacks include deauth, MitM and WPS attacks, to name a few. Since all attacks are pre-made and even include GUI, it does not require much technical knowledge.[16]

However, there are some drawbacks. The main one is that new attacks are usually not developed on this platform. Instead, developers opt for more usual and versatile devices, such as Raspberry Pi. Therefore, it often takes time and work to make the attacks work on Wi-Fi pineapple.[10] As such, it is not a leading-edge device but rather a well-defined, easy to use and fast tool.

The device sells for 100-140 EUR, including three independent Wi-Fi antennas and the software.[16] It seems like a fair price compared to other hardware solutions, mainly when it includes the software and saves much time for penetration testers.

2.2.2 ESP32

ESP32 is a series of SoC[1] chips developed and manufactured by Espressif Systems. It offers low power consumption, integrated Wi-Fi @2.4GHz and dual-band Bluetooth, and some versions include GPIO or other features; all that for a meagre cost. Espressif itself promotes the device as ideal for Industrial applications, the Internet of Things, or as a means for getting Wi-Fi/Bluetooth functionality to other devices. [12]

ESP32 can be broadly divided into three categories.

The first is the System-on-a-chip alone. However, that is not useful for small-scale applications and is intended for other hardware manufacturers as a component. SoC alone is not certified, and therefore the certifications must be obtained by the customer.

The second category is the ESP32 module. It adds some components to a chip. Therefore, it is also certified and is a ready-made component that can be added to a product.

The third category is the ESP development kit. It usually has easy access to peripheral connectors and a way to program the device.

Processor used is a dual-core Tensilica Xtensa LX6 that operates on a frequency of 160-240MHz and can perform at up to 600 DMIPS[2]. It also offers an ultra-low-power co-processor, which can be used for rudimentary tasks while the main processor is in deep sleep mode. [12]

Supported wired interfaces include GPIO, I2C, I2S, SPI and UART.

When it comes to wireless applications, ESP32 offers integrated Wi-Fi and Bluetooth. Wi-Fi supports 802.11 b/g/n and therefore works only on 2.4GHz frequency. Bluetooth interface supports v4.2 or v5, depending on the version. It also supports Bluetooth Low Energy. It is certified for both Wi-Fi and Bluetooth. Wi-Fi and Bluetooth share a modem and therefore cannot operate simultaneously. The antenna is integrated as a PCB trace or a U.FL socket for an external antenna depending on the version. [12]

There are numerous real-world applications of this platform. For example, company Norvi uses ESP32 as a part of their industrial IoT products for industrial controllers. It includes added ports to connect to the machinery. One of their promoted applications is using it as a sensor platform, which can be set up as a wireless device with a very long battery life. [18]

2.3 Software Tools

2.3.1 ESP32 Wi-Fi Penetration Tool

Wi-Fi penetration tool for ESP32 was created by our fellow at Brno University of Technology in Czechia. It created a tool implementing some of the most used attacks on Wi-Fi networks: Handshake Capture with the possibility of de-authenticating already connected devices (to capture their handshake on demand); Denial-of-Service attack; and PMKID attack. [39]

[1] SoC = System on a Chip - a computer building architecture where many major components are inside of processor chip instead of on the motherboard

[2] DMIPS = Dhrystone MIPS (Million Instructions per Second) - a standard measurement of computing power

We decided to check out this tool to see how we can use ESP32 in practical applications.

To **Build and flash** the program, we followed the most standard way on ESP32. The prerequisite is just standard ESP-IDF. First, we built the program using the command

`idf.py build`

in the directory of the program, then we used the command

`idf.py flash -p <COM_NUMBER>` to flash it onto ESP. That is all.

To **Start the program**, we connected to the Wi-Fi network created by ESP32 and went to a predefined web page. There, we were greeted by the selection menu.

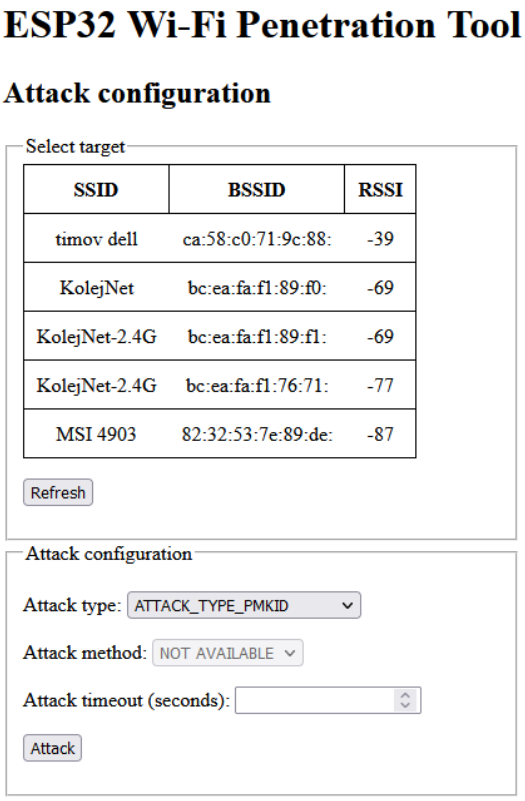

Figure 2.1: menu inside of ESP32 Wi-Fi penetration tool. It allows for picking a target and a mode of attack.

In this menu, we are greeted by the selection of attack method and selection of the targeted network. We ran some of the attacks, all targeting a pre-configured network with two client devices, to see the behaviour of these scripts.

When running the script *deauth_combine_all*, it did something rather interesting. When sending the deauthentication frames, it transmitted both deauthentication targeting broadcast (i.e., targeting all devices) and deauthentication of each individual connected device.

When running the script *deauth_rouge_ap*, it first baited the devices into connecting, and after that, it spammed them with disassociate and deauthenticate frames; and repeating this cycle over and over again. Notably, it retransmitted the same packet tens of times in just a few milliseconds, possibly to ensure that it was delivered.

When running the script *handshake_deauth_broadcast*, the attack tried to capture the handshake of connecting devices. This exact script runs a deauth attack spaced out in time

to allow connecting of the devices to speed up the process. After the time had run out, the script offered us to download the captured data in a text string, PCAP or HCCAPX file. The spacing of transmitting of deauth frames was just right.

All in all, this work is still fully functional as of December 2021. In our humble opinion, the repository [38] offers an excellent start for an ESP32 project.

Chapter 3

Software Defined Radio

3.1 Overview

When it comes to Wi-Fi, the hardware options are all over. There is almost certainly a device that meets the criteria, and the scale brings the cost down to a reasonable level. Mobile network radios lack both. Yes, everyone has a mobile phone in their pocket, but it is not easy to access its functionality. First, we would need to know how to root the mobile phone and then gain access to the modem. This process is different for every mobile phone, is not a common practice, has zero virtually zero support from the community, and has no software ready to work with this. It also voids the warranty on the phone.

Outside of mobile phones, hardware is mostly not made for DIY tinkerers. Instead, the hardware is mainly created for mobile network operators. This fact implies that the hardware is mainly made to create large-scale enterprise mobile networks. Because of it, the unit costs are enormous, and the support is not available outside the enterprise environment. As a result, such a system is hardly usable for research purposes.

The way around missing hardware options is to use Software Defined Radio. SDR is a general-purpose radio peripheral where many components traditionally implemented by hardware are replaced by software solutions. It is a relatively novel approach, first defined in the early 1990s by Joseph Mitola. The ideal SDR should only consist of an antenna, Analog-to-Digital Converter and Digital-to-Analog Converter.[24] It is still not achievable, but modern SDRs indeed replace many components with software.

One of the first projects developing the technology was run by the US Navy. Projects Speakeasy and speakeasy II tried to develop a universal radio for its wide range of radio frequencies used across different branches and departments of the armed forces. In 1996, Speakeasy phase II produced a prototype that allowed interoperability between HF, SINCGARS, Civil Aviation, Have Quick II and UHF SATCOM radios. [22] [42]

The most popular SDRs for the public are re-purposed DVB-T receivers. With some software modifications, very cheap USB dongles receive signals other than DVB-T, for example, GSM. However, they are not universal. Their hardware design limits them to lower frequencies, and they cannot receive all the mobile frequencies. They are also limited in the bandwidth they can receive at a given frequency tuning.

If our goal is to use SDR in mobile network applications, it has to meet some criteria:

1. Depending on exact needs, the device might be required to be able of full-duplex [1],

[1] Full-duplex describes a device that can receive and transmit at the same time. It is in contrast to half-duplex, which can only transmit or receive at a time; and with receive-only or transmit-only devices.

GSM - 2G	LTE - 4G
800-1000 MHz	700-900 MHz
1700 - 1900 MHz	1700 - 1900 MHz
1900 - 2170 MHz [2]	
2500 - 2700 MHz	
3400 - 3600 MHz[3]	
3600 - 3800 MHz[4]	

Table 3.1: Table of frequencies used for mobile services in Czechia, divided by protocols.

2. Device has to be able to operate on the frequencies of the given standard. The broadcast licensing in Czechia is very roughly outlined in Table 3.1 [8]

3. The SDR should have a wide support base; An active community, receiving support from manufacturers, drivers, and Testing for given applications.

If we consider these requirements, the hardware options are suddenly few and far between. In the next few chapters, we will describe these options a pick an optimal platform for further use.

3.2 SDR Devices

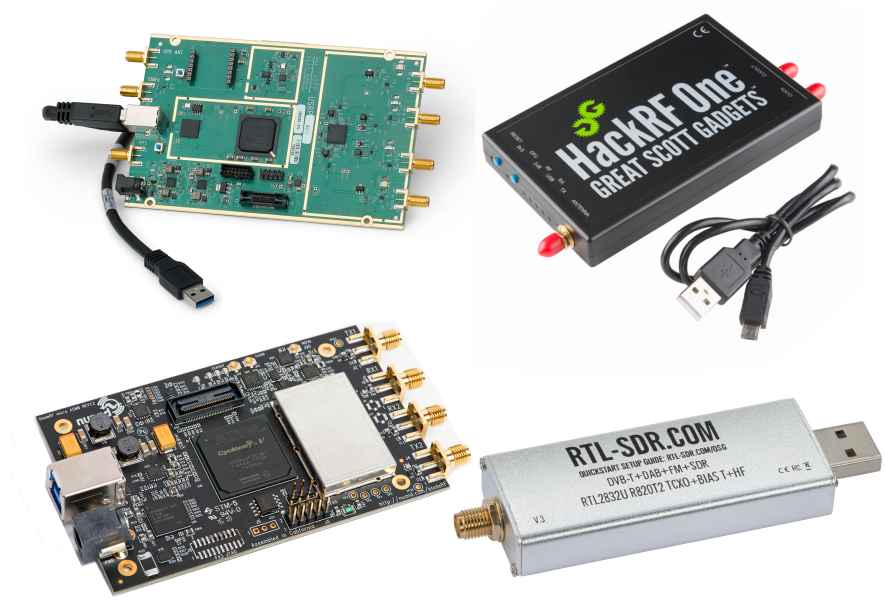

Figure 3.1: Hardware options for an SDR. Clockwise from top left: USRP B210, HackRF One, BladeRF mini, RTL-SDR

3.2.1 Required Hardware for Mobile Network Attacks

When it comes to mobile networks, it is way harder to obtain both software and hardware for such an attack. First of all, there are no off-the-shelf solutions available to the public known

	BladeRF 2.0 Micro XA9	HackRF	USRP B210	RTL-SDR
Manufacturer	Nuand	Great Scott Gadgets	National Instruments	RTL-SDR
Radio Spectrum	47 - 6000 MHz	1 MHz - 6000 MHz	70 Hz - 6000 MHz	0,5 - 2400 MHz
Max bandwidth	56 MHz	20 MHz	56 MHz in 1x1, 30.7 MHz in 2x2	2.4 MHz
Sample Size	12 bit	8 bit	12 bits	8 bit
Sample Rate	61.44 MHz	20 MHz	61.44 MHz	28.8 MHz
Interface	USB 3.0	USB 3.0	USB 3.0	USB 2.0
FPGA Logic elements	300K	–	147K	–
Radio interface	2x2 MIMO	1x Half Duplex	2x2 MIMO	1x receive
Price	800 EUR	390 EUR	1600 EUR	30 EUR

Table 3.2: Specification comparison of popular SDR hardware solutions.

to us; instead, these are reserved for use by the state only. This means that replicating the already known attacks is a lot harder.

In mobile networks, one must strongly differentiate between passive and active attacks. If the attack is passive, we can use a receiver only. Those are generally way cheaper, and some are widely available as re-purposed DVB-T dongles. However, if the attack is active, one must use an expensive SDR transceiver. The cost can quickly rise to over 1000€, based on features.

Nevertheless, the tradeoff is substantial. Active attacks can achieve vastly better results than passive ones.

3.2.2 BladeRF

BladeRF is an SDR manufactured by company Nuand. The company strongly supports starting hobbyists, emphasising documentation and tutorials.

Device supports frequencies between 300 and 3800MHz and is able of full-duplex 28MHz channels, which is all we need. The device uses a USB3.0 interface. The device is also relatively compact (13 × 9 × 2 cm), uses 5V power, and can run headless (without being connected to a PC), making it reasonably mobile. The company also offers versions, that are made to withstand temperaures below 0 °C.

BladeRF is also supported in OpenLTE for scanner Application and File recording[28]. It can even run srsLTE eNodeB [23].

Different versions of BladeRF (differences are in used FPGA, thermal modifications, size etc.) go generally for 400-1,000 EUR [26].

3.2.3 USRP B2x0

USRP B200 and B210 are made by Ettus Research, as part of their wider networking portfolio. Therefore, they are tooled more to be compatible with the existing Ettus research solutions and less as a low-cost prototyping device.

Device supports frequencies from 70 - 6,000MHz with a bandwidth of 56 MHz. Again the main peripheral is USB 3.0. It uses Spartan6 XC6SLX150 FPGA. To this day, it is the

only device supported by OpenLTE as eNodeB. [28] It is also be able to run eNodeB from srsRAN. [23]

The device is sold for around 1,600 EUR. [13]

3.2.4 Nooelec Nesdr

NESDR RTL-SDR is a series of SDR receivers manufactured by the company Nooelec. They are in a form factor of a USB dongle, offer connection to an external antenna, and draw some resemblance to DVB-T tuners.

Most of these devices offer a maximum operable frequency of 1700MHz. However, version „XTR“ offers a maximum operable frequency of 2300MHz. It still cannot receive all mobile network frequencies, but it can receive the most widely used ones. It is also limited in received bandwidth compared to other SDRs.

Now, the elephant in the room: it is only a receiver, severely limiting its usability. Nevertheless, it is very affordable. It also has OpenLTE support for scanning and recording. Despite its shortcomings, it is an appealing choice for first-contact experiments.

3.3 Existing Software

3.3.1 SoapySDR

SoapySDR is a generalised open-source API and run-time library for interfacing with SDR devices. It is supported by most SDR hardware platforms and offers bindings for frameworks such as GNU radio (such as source/sink blocks).

Its philosophy is not to be a full hardware abstraction library, where the implementation can be platform agnostic. Nevertheless, it is possible to use it as such. [33]

3.3.2 GNU Radio

GNU radio is a development toolkit that provides signal processing to implement software radios. It is made to be used using hardware (such as SDR) or with mocked input in a simulation-type environment.

The foundation of the GNU radio is a block. Block implements a single functionality (low pass filter, Fast Fourier Transform, sink/source block from a file or a peripheral and others). These blocks are then used together to create the desired output. This concept reminds us of the ideal SDR described in Chapter 3. The ideal radio is only an ADC, and all of the other hardware functions are replaced by software implementation. These blocks can, in most cases, replace a given circuit one-to-one.

The GNU radio has broad support, being used in research and development, academia, government and hobbyists alike. [14]

3.3.3 ASN.1

ASN.1 is a formal notation for data serialisation and deserialisation of transmitted data, especially in telecommunications. It defines a formalism for the specification of abstract data types. It sends any type of data, such as audio, video or data. It is very conservative in bandwidth, which proves to be useful in the modern world.

The S1AP protocol, which mediates communication between eNodeB and MME, is defined by ASN.1. The definition of the language is part of the technical specification of S1AP. [1]

3.3.4 OpenBTS

OpenBTS is an open-source Unix application, first created in August 2008. It aimed to implement the lower three layers of the GSM standard; use the then-modern USRP [5] as radio equipment; and VoIP switching such as Asterisk. This combination meant that only a single PC with a USRP was required for running a fully-fledged network. [7]

It was the first open-source implementation of GSM. It is claimed to have opened the floodgates to much research in telecommunications, lowering the cost of entry from tens of thousands of dollars and legal barriers to just a PC and SDR.

The original developers have since written a book, „Getting Started with OpenBTS“ [19]. GSM is, therefore, a technology fully embraceable by an audience wider than just a selected few.

However, the project is considered finished and is abandoned, with the last functional updates to it done in 2016 [34].

3.3.5 OpenAirInterface

The industry has been fully aware of all the positive effects OpenBTS has brought. Following its footsteps, a french non-profit, OpenAirInterface, has decided to create an open-source implementation of 3GPP (LTE) as a successor to OpenBTS. The goal was to „enable research outside of vendor and operator R&D groups“ and allow for a low-cost open LTE network, thus democratising LTE. [25]

After finishing and demonstrating the LTE platform, its focus has shifted towards 5G networks. LTE systems are grouped in repositories with their 5G counterparts [27].

3.3.6 OpenLTE

OpenLTE is an open-source implementation of 3GPP (LTE) specifications. It has been developed by a small group of programmers since 2012 and is yet to reach version 1.0 [43].

It also does not split the different parts of SDR, such as eNodeB and MME, which complicates its use and goes against the principle of modularity.

Even though it seems to be working, we could not find a single application of this tool.

3.3.7 srsRAN

srsRAN (originally srsLTE) is an open-source platform implementing eNodeB (base station). It is fully compliant with LTE protocol release 8. One of its main goals was to make it a test-bed. Therefore, it allows for modularity and ease of changes inside the platform. It greatly simplifies experimentation with the technology. [23]

Unlike OpenLTE, it splits the different implementations of the protocol (such as MME, HSS, eNodeB ...) into separate pieces. The split of different parts is visualised in Figure 3.2. This great modularity allows us to swap all parts for our implementation. Its conformity to the standards also allows us to use the documentation as it was meant to be used.

[5] USRP = Universal Software Radio Peripheral (essentially the same thing as Software-defined radio)

On top of that, the platform offers limited support in the form of additional Testing to a few selected SDR devices. These are USRP B2x0, BladeRF and HackRF. There is also the possibility that it can work with other hardware using the SoapySDR library, but there is no certainty. [23]

SrsRAN is supported by the company Software Radio Systems (SRS). The company is situated in France and continues to implement open-source implementations of the 5G protocol. When the company continued developing the product from 4G to 5G, the company was renamed srsRAN (srs Radio Access Network). The shift toward the 5G got the 4G protocol in the shadow, but it is kept updated, supported and up to spec.

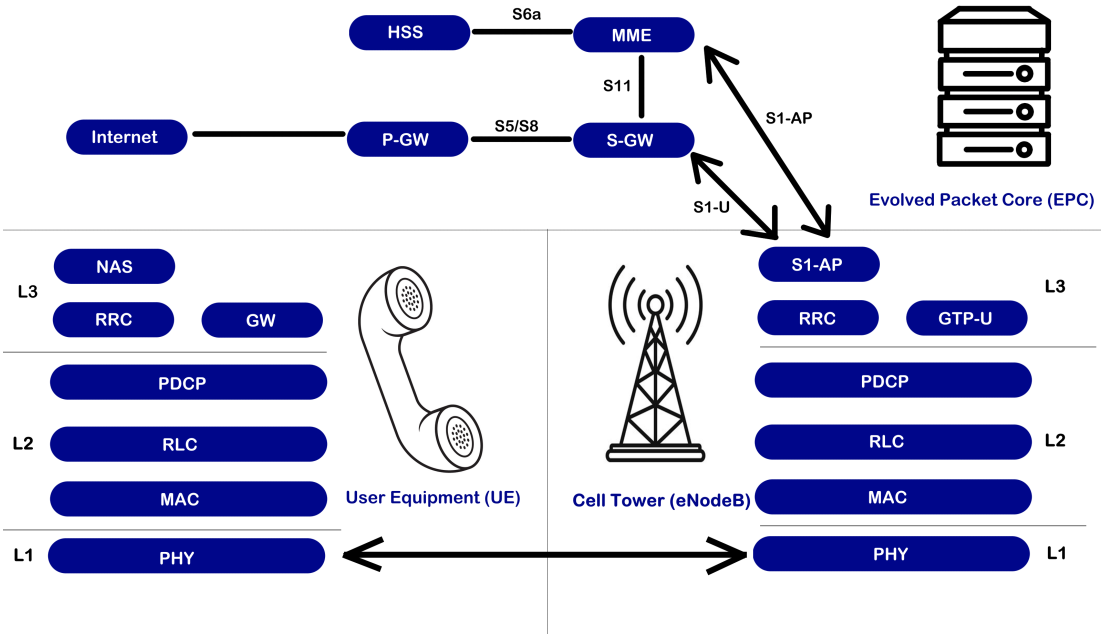

Figure 3.2: A layout of the full LTE stack as implemented by srsRAN: UE connects to the eNodeB through radio interface. eNodeB connects to EPC using S1 protocols. [23]

3.4 Already Attempted Attacks

In this section, we will summarise the already attempted attacks on mobile networks using SDR. We will go through the used platforms, applied methods and taken precautions. If possible, we will also pinpoint the most important or surprising information we got from that paper.

3.4.1 Rogue Base Station

Rogue base station is a stand-alone device that mimics legitimate base stations in a mobile network [41]. It is possibly the most often used baseline tool when attacking mobile networks. If implemented with proper care, phones have no way of distinguishing rogue and actual base stations.

The attack uses open-source software implementing base stations. This software follows the protocols for the type of network for which we masquerade. Some of these open source

tools are explained in Section 3.3. Tools in their original shape are legitimate software. They might allow, for example, the creation of a DIY mobile network. However, they offer a great head-start and can become an attack tool with just a few modifications. On top of it, the base station is set up in such a way to get as many connections by UE [6] as possible.

Due to the architecture of the LTE protocol, it is not possible to establish a full MitM attack. The attacker would need to be able to do a security key exchange. For that, he needs to know the pair of IMSI and Ki: security key known to the operator and written on the SIM card. This security key does never change. The complete connection in LTE happens only after the key exchange, where IMSI and Ki are used to establish the trust between the UE and eNodeB. That is not the case for the attacker; therefore, the MitM is impossible. Furthermore, the encryption algorithm is a lot stronger than on GSM networks. There are only a few protocol procedures accepted prior to the key exchange.

It is, however, possible to establish a full MitM attack with GSM protocol. Whether we do a full MitM or just want to use the unprotected procedures in LTE, we want the mobile phones to connect to our network and not to other networks. One might say that **we want to increase the attractiveness of the tower.** This can be achieved in multiple ways.

The first and most apparent is setting the country code, operator code and cell identifier. UE will try to avoid connecting to foreign operators or roaming.

The second is setting region code. If UE is detecting base stations all with the same region code and one with a different region code, it will assume that it is going from region to region; and connect to the tower with that different region code. Therefore, setting a different region code in use around the place of attack increases the chances of receiving connections.

The third way uses a protocol feature used to balance the load on different base stations dynamically. The process is called Radio Resource Management (RRM). We can set RRM parameters to look underused. With that, the UE will try to connect to this base station.

The fourth option is to use preferential frequencies. *Absolute priority-based cell reselection* was introduced in LTE release eight specifications. Even if the phone has a good enough connection with its current eNodeB, it will switch to another eNodeB if it uses preferential frequency. This information is defined in SIB Type numbers four, five, six and seven in the entire network. We can sniff this information and set up our rogue eNodeB accordingly. [36]

It should be noted that broadcasting on an unlicensed spectrum is illegal per US and EU law. Therefore, all experiments should be done in an isolated environment - a Faraday cage.

3.4.2 Downgrading Attacks

Introduction

The basic principle of a downgrading attack is to force the target device to use older standards than it may otherwise use (practically to downgrade from LTE to GSM). Since the security of every subsequent generation is continually rising, forcing the use of an older standard can be very effective. The methods used to achieve this are based on the standard and executed using fabricated control messages. The user has no real defence against them.

[6] UE = User equipment. Usually, it means a mobile phone, but broadly speaking, it includes any user device with a modem.

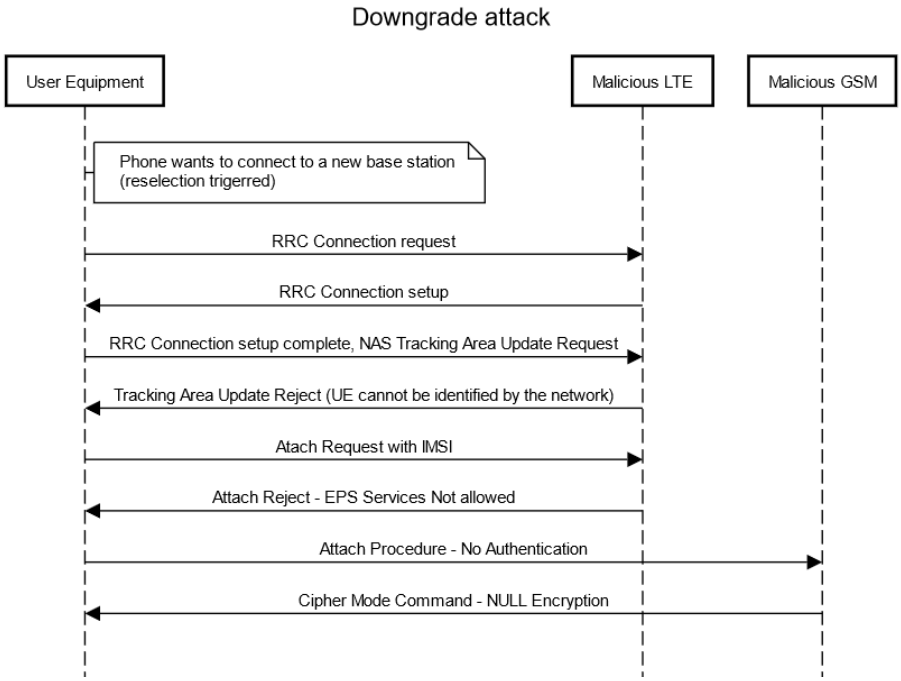

Figure 3.3: Downgrade attack sequence diagram, as executed by [45]

Furthermore, it may not even be possible without a change in protocol. The most frequent and easiest to implement and misuse is downgrade the mobile phone to GSM service.

Broken Encryption

The security of the GSM standard is widely considered broken. The encryption in GSM used is A5/1, A5/2 and A5/3, respectively. A5/1 and A5/2 are thoroughly broken, can be decrypted almost real-time and are slowly being phased out in favour of A5/3. However, mobile phone operators have been slow in adopting A5/3 encryption. Not only that, the new encryption is not a silver bullet.

While implementing A5/3, developers moved from MISTY encryption to its sub-variant KASUMI. At first, it was argued that it would be only a slight downgrade with a significant benefit in reduced hardware requirements. However, this has resulted in much weaker encryption than initially planned. Unlike A5/1 and A5/2, A5/3 is still not real-time decrypted. Nevertheless, it is not unbreakable. A single PC was able to decrypt it in two hours [11].

It is still not a very practical attack, but moving forward, we can expect that A5/3 will rapidly become obsolete as well.

The weakness of GSM is the reason for the efficiency of this attack: if we can force a user to use old and broken protocol, it is a lot easier for us to create MitM or to eavesdrop on the network. Nevertheless, even with the upgraded encryption, attacks downgrading to GSM will still carry much potential and target even modern devices.

Functionality Description

The simplest method of achieving a service downgrade is to exploit TAU Reject and Attach reject messages.

Using this, the rogue station can indicate to a target device that it is not allowed to access 3G/LTE networks in the area. Therefore, the target device will not even attempt to connect to these networks and only connect to GSM base stations. [32]

Per protocol, the behaviour is described as follows [3]:

> The UE shall take the following actions depending on the EMM cause value received in the ATTACH REJECT message: #7 (EPS services not allowed):
>
> The UE shall set the EPS update status to EU3 ROAMING NOT ALLOWED (and shall store it according to subclause 5.1.3.3) and shall delete any GUTI, last visited registered TAI, TAI list and eKSI. The UE shall consider the USIM as invalid for EPS services until switching off or the UICC containing the USIM is removed or the timer T3245 expires as described in subclause 5.3.7a. Additionally, the UE shall delete the list of equivalent PLMNs and enter state EMM-DEREGISTERED. If the message has been successfully integrity checked by the NAS and the UE maintains a counter for „SIM/USIM considered invalid for GPRS services“, then the UE shall set this counter to UE implementation-specific maximum value.
>
> If A/Gb mode or Iu mode is supported by the UE, the UE shall in addition handle the GMM parameters GMM state, GPRS update status, P-TMSI, P-TMSI signature, RAI and GPRS ciphering key sequence number as specified in 3GPP TS 24.008 [13] for the case when the normal attach procedure is rejected with the GMM cause with the same value.
>
> If the UE is operating in single-registration mode, the UE shall in addition handle the 5GMM parameters as specified in 3GPP TS 24.501 [54] for the case when the initial registration procedure is rejected with the 5GMM cause with the same value.

Note: The attack lasts either until USIM is removed (i.e. restart of phone or SIM reinsertion) or until T3245 expiration (that is random time in the range of 24-48 hours [2]).

After that, the attacker faces an option. Either they will just passively eavesdrop on the GSM communication going to a legitimate GSM station (and breaks the antiquated encryption), or they will establish a rogue GSM station of their own, as described in Subsection 3.4.1 (if A5/3 is used).

Past Demonstrations

The concept is not new, and it is hard to track down the original demonstration. But to give an example of combining downgrade by a rogue LTE station with forcing a connection to a rogue GSM station, it was demonstrated by Shuhui Chen and collective in 2019, using off-the-shelf hardware. [45] First, a rogue LTE base station was created using USRP B210. This was used to trick a phone into connecting. If the phone decides to initialise the connection, it sends a Tracking Area Update request.

After that, the attackers were able to transfer the device into connecting to a rogue GSM station using the Tracking Area Update. But before forwarding to a GSM station,

the attackers send an Identity Request to gain more data. To achieve a full MitM attack, the attackers use this data to create a connection to a real GSM station using an old phone and spoofing the IMSI that was gathered before. [45]

The project has used about four PCs to run the networks and forwarding. In our humble opinion, this prototype setup would beneficiate from at least some downsizing.

3.4.3 Single Device Attach Procedure DoS Attack

Denial of Service (DoS) and Distributed Denial of Service (DDoS) are more commonly associated with wired rather than wireless connections. However, their potential should not be underestimated. Denial of mobile network services at the premises of a market competitor or a stock exchange can have serious repercussions and should be taken into account. [31]

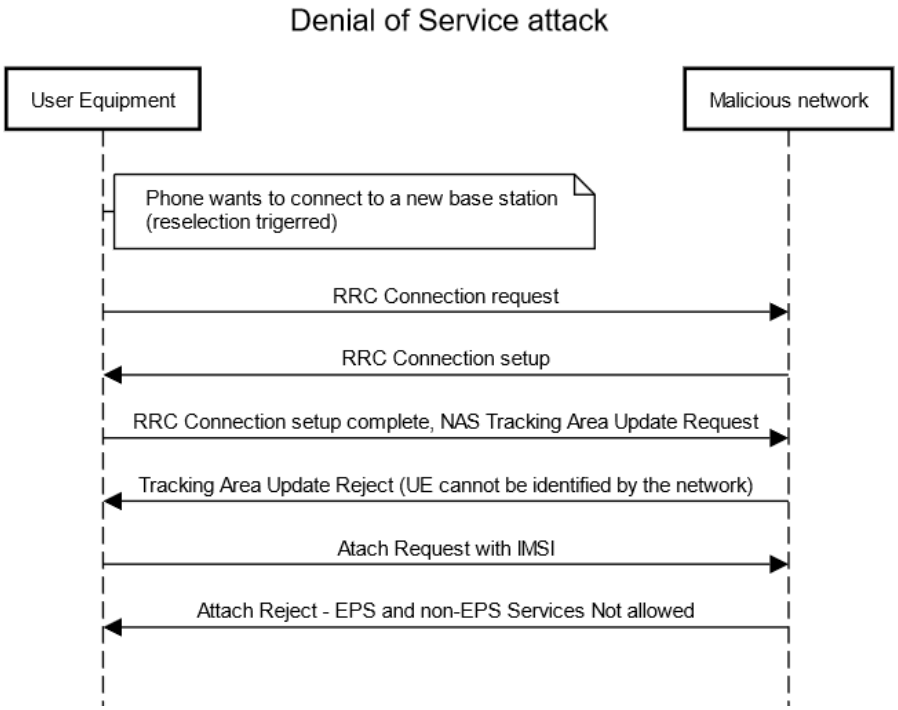

Figure 3.4: Simple targeted Denial of Service attack.

This attack is again very similar to a Downgrade attack or an active IMSI catcher. The difference is that the EMM Attach reject cause is *Cause #8: EPS and non-EPS Services Not Allowed*. This message is sent after acquiring the IMSI of a given phone. Therefore, this DoS message can target a specific user.

Per protocol, the reaction to this Attach reject is described as [3]:

> The UE shall take the following actions depending on the EMM cause value received in the ATTACH REJECT message: #3 (Illegal UE); #6 (Illegal ME); or #8 (EPS services and non-EPS services not allowed);
>
> The UE shall set the EPS update status to EU3 ROAMING NOT ALLOWED (and shall store it according to subclause 5.1.3.3) and shall delete any GUTI, last visited registered TAI, TAI list and eKSI. The UE shall ETSI consider the

USIM as invalid for EPS services and non-EPS services until switching off or the UICC containing the USIM is removed or the timer T3245 expires as described in subclause 5.3.7a. Additionally, the UE shall delete the list of equivalent PLMNs and enter state EMM-DEREGISTERED. If the message has been successfully integrity checked by the NAS and the UE maintains a counter for „SIM/USIM considered invalid for GPRS services“, then the UE shall set this counter to UE implementation-specific maximum value. If the message has been successfully integrity checked by the NAS and the UE maintains a counter for „SIM/USIM considered invalid for non-GPRS services“, then the UE shall set this counter to UE implementation-specific maximum value.

If A/Gb mode or Iu mode is supported by the UE, the UE shall in addition handle the MM parameters update status, TMSI, LAI and ciphering key sequence number, and the GMM parameters GMM state, GPRS update status, P-TMSI, P-TMSI signature, RAI and GPRS ciphering key sequence number as specified in 3GPP TS 24.008 [13] for the case when the normal attach procedure is rejected with the GMM cause with the same value.

If the UE is operating in single-registration mode, the UE shall in addition handle the 5GMM parameters as specified in 3GPP TS 24.501 [54] for the case when the initial registration procedure is rejected with the 5GMM cause with the same value.

NOTE 2: The possibility to configure a UE so that the radio transceiver for a specific RAT is not active, although it is implemented in the UE, is out of the scope of the present specification.

Note: The time the attack is active is either until USIM is removed (i.e. restart of phone or SIM reinsertion) or until T3245 expiration (that is random time in the range of 24-48 hours [2]).

Reaction of UE to this message seems not to be universal. Some sources claim that this UE will not attempt to connect to a mobile network until restarted, SIM reinserted, or timer T3245 expires (24-48 hours) [44, 2]. Others suggested that a UE will not connect for a relatively small limited amount of time (like 10 minutes)[17]. Therefore, the attack's efficiency depends on the UE manufacturer's implementation.

Every successive generation of mobile networks, from GSM to 5G, aimed to accommodate more data from more users than their predecessor. To allow this, operators began using so-called picocells and femtocells - base stations with a very low effective range (femtocells are up to 100 meters). Also, many more devices access this network, thanks to the trend of IoT.

3.4.4 Overload Style DoS Attack

This DoS attack tries to overload the eNodeB resources and Evolved Packet Core resources. After analysing the protocol, researchers selected a few methods that generated the most work for the network. For example, a UMTS session setup and resource release require 27 signalling messages to occur. The LTE attack works similarly. LTE has a concept of Bearers. These are responsible for carrying information in the network for a given connected UE. We can differentiate payloads based on the type of traffic (based on the Traffic Flow Template field), for example, FTP or HTTP. A single UE can establish up to 8 bearers. Bearer activation procedure and deactivation procedure for a single bearer require 24 messages, 12

of which are processed by the eNodeB. Using this mechanic, we force the network entities to attach and detach bearers, which can quickly eat all available resources on eNodeB. We can amplify this by 1. timing the bearer activation, 2. inserting a small data transfer between activation and deactivation, 3. using all available bearers, and 4. using more devices. [6]

In our opinion, this attack seems to be heavily temperate on used eNodeB hardware and having multiple legit SIM cards attached. Such behaviour might be interpreted as against the Terms of Service by the operator. Additionally, this attack will be less powerful as we move towards the IoT trend of many devices. The network will have to be ready for this influx, and some additional (although malicious) users do not add up to the weight required. There is also the problem of load balancing in the network. It would be hard to concentrate the attack on a single tower because all the towers will try to balance loads on them. Moreover, the solution to the attack is as simple as dropping the demanding users' connections or blocking the SIM cards entirely for breaching the TOS. Again, in our opinion, the feasibility is not there.

3.4.5 Brute Force Jamming

Radio jamming describes an act of transmitting radio signals, intending to disrupt legitimate communication. Classical radio jamming transmits on a broad band of radio signals with the goal of noise out the communication. From the principle, it is hard to direct.

The jammer was attempted by Van Rijsbergen at University of Amsterdam. They created an experimental base station using Blade-RF (with some modifications) and initialised a test phone call between the Blade-RF and a phone. Then, they tried to jam the phone call using HackRF. This was done in a lab, using a Faraday cage to not interfere with other people. Results were largely unsuccessful: it dropped calls consistently only in the GSM900 range, and even then, the phone was able to begin a new test call and go around the jamming. Frequencies of 1800 and 2100MHz were not affected at all. One of the possible explanations offered was the low signal strength of the jamming device, combined with the ability of mobile networks to bypass interference. [40]

Some jammers specify the jammed band to just control channels. Since the attack is much more concentrated, it requires smaller devices that consume less power. Also, it is much more powerful since it does not block devices within a range of the radio jammer but in range of the jammed base station - which is generally a lot wider. [31]

3.4.6 Spoof Jamming

To „brute-force jam“ a normal base station, one needs a lot of transmitter power. The station has a powerful signal, and the standard was built to work even during strong interference. However, if we use our knowledge of the standard and abuse its properties, we can gain a measurable advantage in jamming capability.

Instead of trying to noise out transmitted data, we add data of our own, which is harder to filter out. Different channels are used in the LTE standard. Researchers have tried spoof jamming different channels and signals and measured its effect. The most successful was the jamming of the Control Format Indicator Channel (PCFICH). If they were to barrage jam with the same efficiency, they would need a 27dB stronger broadcast. [21]

However, the best implementation time to efficiency ratio was gained by jamming the synchronisation signals. The LTE protocol has two such channels - primary synchronisation signal (PSS) and secondary synchronisation signal (SSS). It works by spoofing the synchronisation frame and sending it sooner than expected. The devices then expect a Master

information block right after the signal. It will not be there since it got the synchronisation signal ahead of time.

This information is interesting because it allows for very powerful jamming without potent equipment.

3.4.7 Insider Attacks

The ever-increasing complexity of the network keeps opening new possible vectors for attack. For example, it has been demonstrated that it is possible to gain root access to a femtocell access point [15]. This fact breaks our assumption that access to the internal network is only possible for actual base stations [31]. Gaining access to a base station could theoretically open their use as a much stronger radio for illegal activity. These attacks are primarily conceptual and out of this project's scope.

3.4.8 IMSI Catcher

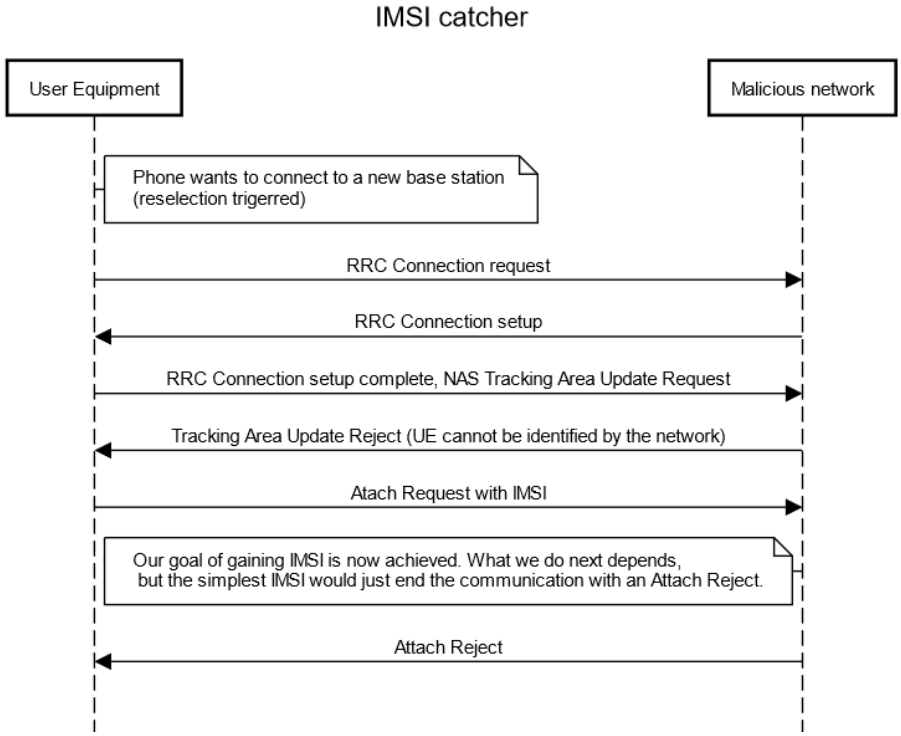

Figure 3.5: Active IMSI catcher sequence diagram.

This type of attack is one of the most primitive attacks on the mobile network. Even its highest form does not require full implementation of a base station. Its goal is to capture the IMSI of a given UE.

IMSI is a code used to identify UE on the mobile network.

There are two ways to execute an attack: passive and active.

In the passive attack, we try to sniff mobile communication and extrapolate IMSI. It is generally more effective with 2G networks. Not only is it easier to sniff 2G networks. 2G networks will use the IMSI (and not TMSI) more often than 4G networks.

In the active attack, we make something resembling a base station. When the UE starts to connect to our base station, we usually cancel the connection. We receive an IMSI from the UE during this process, per protocol. [3]

There are two mechanisms through which the UE will try to connect to our tower.

The first is Tracking Area Update Request (TAU Request). This is an effort by the UE to switch between towers if it is already connected to one. Because of this, it will mainly contain TMSI (GUTI) as an identifier (although IMSI is possible as well. We reject this request using TAU request, where we choose a *Cause #9 - UE Identity Cannot be derived by the network*. Our network has no record of the already assigned TMSI (GUTI).

The second mechanism is `Attach request`. The examples of causes are that the TAU Request fails or TMSI is not assigned. Usually, the Attach request comes with the IMSI identifier. If, however, it will come with TMSI or other identifiers, we can respond with `Identity request` and demand IMSI. `Identity response` then shows us this IMSI, which was our goal. After that, we can cancel the connection with any cause we want that is accepted without security mode.

Since the attack is based on a protocol and UE just blindly follows the protocol, there is no easy defence against this attack by the individual. However, there are some rather unusual solutions. For example, one can gain root access to a mobile phone and modify its software to ignore the messages.

There were also studies trying to detect IMSI catchers. [4] These methods include methods such as fingerprinting the cells and focusing on Location Area codes.

Chapter 4

Design

In this chapter, we go over the final product's design.

First, the Faraday cage has to isolate all the experiments. The existing solutions are lacking for our application, so we will attempt to design own makeshift Faraday cage.

A significant focus has to be put on the hardware requirements. The used combination of radio equipment and control computer needs to create an eNodeB and execute attacks on it.

The used computers have to be able to take on the load of the radio equipment and software. Our main focus is the required bandwidth and required computational power. However, with the focus on miniaturisation, we want the smallest, the least power-hungry computer to do the job.

4.1 Faraday Cage

In most countries around the globe, broadcasting on licensed frequencies without a licence is illegal. Going the other way around, the licensed broadcast can interfere with our experiments. Therefore, all research experiments must be executed in an isolated laboratory environment. The easiest way of doing this is to buy or create a Faraday cage.

Our Faraday cage must be able of two things: keeping the radio broadcast of a public cellular network out, and keeping our experimental radio broadcast in. The required signal reduction is quite high: we might need to lower our signal by up to 100 dB [20]. It should be big enough to hold all of the required equipment. This, at a minimum, includes a single mobile phone and an SDR (therefore, the Faraday cage has to have a size of at least a shoe-box). Based on the efficiency of the design, we might be able to run cables into the Faraday cage. If not, the Faraday cage has to enclose not just the minimal equipment but the controlling PC (most likely a Raspberry Pi or other USB 3.0 equivalent[1]), and a battery for the computer as well as the SDR.

To verify our experiments, we simply put a mobile phone inside the proposed Faraday cage and tried to initiate a phone call. If the phone starts to ring, the insulation is insufficient.

When buying a Faraday cage, one can easily see two categories of products.

The first is a consumer-targeted cage. They are usually unable to block relatively strong transmission of an SDR. Additionally, most of them do not depict any measured reduction in

[1] Most of the SDRs on the market require an USB 3.0 connection for full bandwidth

signal on the package. Their effectiveness is therefore unknown until bought and measured by ourselves.

The other category is licensed Faraday cages for laboratory and industrial uses. This option is, however, too expensive. For this reason and the sake of the experiment, we started with creating a Faraday cage of our own.

In the first iteration, we tried to use aluminium kitchen foil, as it is the most readily available material. However, this has been proven ineffective for mobile network broadcasts. The only way we could block the phone signal completely was when the phone was folded in the tinfoil without an SDR and any wires attached. Even then, results depended on the quality of the folding performed.

The re-usability is also minimal. When measuring the impact of aluminium, the usual signal reduction was only around 40dB[2].

In the second iteration, we tried a small metal box. While it brought a measurable signal reduction, the signal was never blocked.

In the third iteration, we tried to use probably the most famous type of a Faraday cage - a microwave. To our surprise, this has not brought the wanted results as well. While the LTE signal was shielded a bit, the phone could still get the signal of a public GSM network and receive a phone call.

The final solution combined second and third iterations - the mobile phone was put into a metal box, which was put inside a microwave. This delivered the signal blocking quality and the required consistency of results. This setup is also easily replicable and serves as a possible guide to creating a dedicated Faraday cage for mobile network experimentation.

However, this setup does not allow for cables running into it. Therefore, the setup necessitates that the PC and batteries be included in the cage.

4.2 Radio Equipment

When it comes to the radio equipment, the options are laid out in Section 3.2, with the images of the solutions on Figure 3.1. In this section, we ignore the UE and focus on SDR.

We need a single device serving as a BTS/eNodeB. These are clearly outlined for each software platform. This should be enough for some attacks, such as the IMSI catcher, DoS and Downgrade.

For other attacks, such as jamming, We need at least two radio devices - one for the BTS/eNodeB, the other for attacking it. It should be enough even for the downgrade attack. The real-life scenario also expects a real tower, from which we have to capture the target UE. However, the preceding work used just two radios (one for the GSM BTS, the other for the LTE eNodeB [45]) and ignored the UE capture problem.

It should be possible to combine two different SDR devices. However, it has been rarely done and brings extra work to make sure it works on two different devices. Therefore, the SDR devices should be the same.

Since the cost of the SDR devices is not negligible, and we already have access to one BladeRF, the setup will be most likely made of two BladeRFs. The implemented attacks require only one SDR. The chosen BladeRF is displayed on Figure 4.1 and Figure 4.2, respectively.

[2]Measured with the help of an Android app Gmon pro.

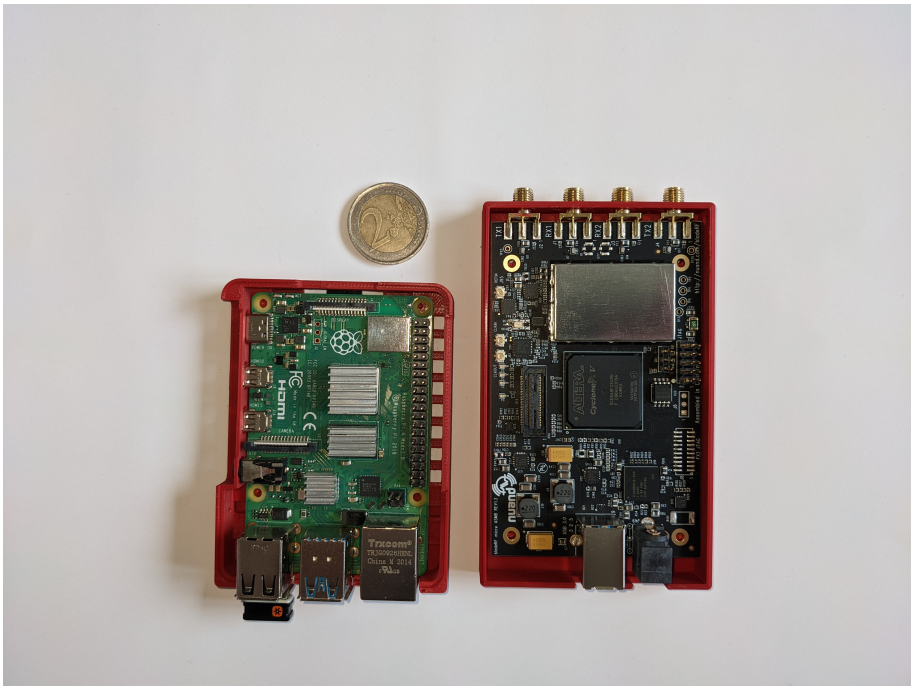

Figure 4.1: Our chosen hardware - Raspberry Pi 4B+ and BladeRF Micro 2.0 AX9. Both in opened matching 3D printed case. 2 EUR coin included for size reference.

4.3 Computer Usage

Our first focus comes to the bandwidth required. All of the SDR devices require a USB 3.0 bandwidth. That limits our choices considerably. This first requirement blocks us from using the lowest level microprocessors.

It also needs to be able to pull the software of eNodeB. Our MME is mostly negligible. The first benchmark we have is a test done by the srsRAN team. They have run srsENB on Raspberry Pi 4. The RAM utilisation was around 2 GB, and CPU utilisation sat at around 25%. There is also recommendation to not use more than 6 Physical Resource Blocks [3] with the newest version of srsENB, as it may cause problems. That is not a problem for us, as we do not attempt to create a real-use tower to download high amounts of data. [37]

The computer should be able to be powered by battery. This limits us even further. The best area of search for computer should be cheap SoC computers.

Last, if we do not want to recreate all the libraries and available tools foreshadowed in Section 3.3, it would be optimal to have a computer running Linux. After some experiments, the best option is Ubuntu since it offers the most comprehensive support for non-standard repositories.

Therefore, our first choice is a Raspberry Pi 3 B or above. It offers USB 3.0, but it also offers the standard Linux support and some considerable computational power. However, there remains a question about its memory speed. Its power requirements are for a 5V/3A

[3] It is the smallest unit of resources, that can be allocated in LTE.

Figure 4.2: Our chosen hardware - Raspberry Pi 4B+ and BladeRF Micro 2.0 AX9. Both enclosed in 3D printed Case.

charger, but the average power draw is said to be under 7W. That is a bit too high, but for a prototype, it is good enough. It should be possible to power it using just any power bank with fast charge. For Raspberry Pi also speaks that the SRS has tested running srsENB on Raspberry Pi with a high degree of success. While the USB bandwidth does indeed support the SDR, there is a small possibility that memory speed will be an insurmountable bottleneck.

There are quite a few more powerful step-ups from the Raspberry Pi 4. All of them have one in common: lower support, smaller community, higher performance and higher power consumption.

The solution for higher performance is for example ROCKPro64. Its upside is that it has an onboard MMC storage. It is also slightly more powerful than Raspberry Pi. Its first downside is higher power requirements (requires as much as 12V/5A), which would require a lot bigger battery than a standard power bank. The second is lower support from the hobbyist community.

When choosing hardware platform, it should be noted that srsENB uses priority threads and performance scaling CPU governor to keep latency low. It is therefore not advisable to limit power consumption by software limitations.

But, to reiterate: in our expectation, Raspberry Pi should be good enough. The Raspberry Pi 4 is displayed as a part of setup on Figure 4.1 and Figure 4.2.

Chapter 5

Implementation

This chapter goes into the details of our implementation. We first discuss real software requirements on our code. We try to find the best libraries and services for these requirements. After that, we present the implementation proper: a class diagram and a state machine.

5.1 Software Implementation Outline

We propose to implement these attacks: Attach-targeting denial of service, IMSI catcher and Downgrade attack. All of these attacks work on a simple principle. The UE connects to our fake eNodeB and sends an Attach request (usually also with a PDN connectivity request). The eNodeB sends this Attach request using the S1AP protocol to the MME. This is done using NAS transport message in the S1AP. After that, the MME similarly responds to the eNodeB with what to do with the request. In our case, we might want to send an Identity request first, and wait for the response with IMSI. But in the end, we always send an Attach reject (and if we got a PDN connectivity request, we send PDN connectivity reject as well). Based on attached reject *Cause*, the UE will behave as said by the protocol.

When we analyse the way these attacks are implemented, one thing becomes apparent. We do not need to touch the implementation of eNodeB at all. Instead, we just have to replace the MME with our own mock. The MME has to be able to:

1. Communicate with the eNodeB using S1AP protocol;
2. Create connection with eNodeB;
3. Encode and decode `Attach request`, `Attach reject`, `Identity request`; and
4. Work with `Attach request`, `Attach reject`, `Identity request`;

Looking back on these requirements, we need to ensure that the chosen development platform has dedicated eNodeB and MME. If that is not the case, it is not easy to replace the MME.

Another feature is that the S1AP protocol is defined using ASN.1. There are ASN.1 parsers [30] that can take some burden off of our shoulders in this work. However, it is noteworthy that the work with ASN.1 will probably result in many dynamic structures. That would favour using a more freely typed language.

When picking the implementation language, we need to consider available tools, development speed, and support on the target PC. As discussed in the previous paragraph, using a more freely typed language might be advisable.

At first glance, there are two outlying programming languages for implementing the software. The first is C/C++. The language has ever-present support and is fast during runtime. The second is Python. The language has also strong support, has fast implementation time and has more easily supports the dynamic structures that the ASN.1 parser will present us with. Additionally, Python has to be installed on the PC anyway, so it does not add as much burden to it as it might seem.

At last, we decided to use Python.

5.2 Used Libraries

As outlined in the Section 5.1, numerous tasks are probably already implemented using libraries. We will divide the libraries by their function and describe them.

5.2.1 SCTP Connection

The S1AP protocol (used for communication between eNodeB and MME) is run over the SCTP protocol. *Stream Control Transmission Protocol* is a transport layer protocol especially used in the telecommunications industry. In our case, we will create only a single connection from the MME to the eNodeB.

As we decided to use the Python language, the best option is to use PySCTP by P1Sec. It is an SCTP stack for Python. In concurrence to the usual socket functions like `send()`, it also adds functions as `sctp_send()`. These functions provide the programmer with additional content and options specific to the SCTP protocol. It also takes care of establishing and confirming connections, and confirming the arrived packets. [29]

With these functions, we can also set the port number. This is important as the S1AP protocol runs over a specific port number.

SCTP is not a widely used protocol. It is *usually* included in Linux kernel. We can install SCTP libraries onto the PC, but these only interact with kernel implementation. We, therefore, have to pick a kernel that implements SCTP. This can be checked either by the presence of the SCTP module in kernel files or by `checksctp()` utility included in SCTP libraries.

In our case, Ubuntu 22.04-Raspi inexplicably lacks kernel support for the SCTP. We instead used Raspberry OS to solve this issue, which works just fine.

5.2.2 S1AP

As previously discussed, the S1AP is instrumental to the connection of eNodeB and MME. This protocol is purpose-made for something that people often do not work with. That means that the options are a bit limited. It is defined by ASN.1 language.

The best tool we could find is Pycrate by the company P1Sec [30]. It is a Python library that acts as an encoder and decoder for the various ASN.1-written protocols. It was developed with the focus of being used in the telecommunications industry. It has broad compatibility (working on Python2 and 3 as well). It also supports the Cython engine, which could increase the program's speed. It has no run-time dependencies, but Python module `setup-tools` is required for setup.

5.2.3 NAS-PDU Messages

Some messages, such as Attach requests or PDN connectivity requests, are not an elementary procedure within the S1AP protocol. Rather they are NAS messages encapsulated into S1AP message of type `Downlink NAS Transport` and `Uplink NAS Transport`. We need to work with these messages a bit separately.

This is also solved by the Pycrate by P1Sec [30]. It contains a pycrate_mobile subdirectory, which implements most of the 3GPP NAS protocol. This includes EPS mobility management messages.

5.3 Code Implementation

In this section, we will describe our implementation of the foreshadowed problems. We will describe class structures, states of the program, options it provides and functions it contains.

5.3.1 State Machine

Our MME implements only a few S1AP Elementary procedures, such as `S1 Setup`, `Initial UE message`, `Downlink NAS Transport` and `Uplink NAS Transport`. The last two elementary procedures serve as a tunnel for NAS communication between UE and MME. This makes it really easy to just create a simple state machine, where we expect a connection from eNodeB. Then, we can react to the messages that come from it and send response back.

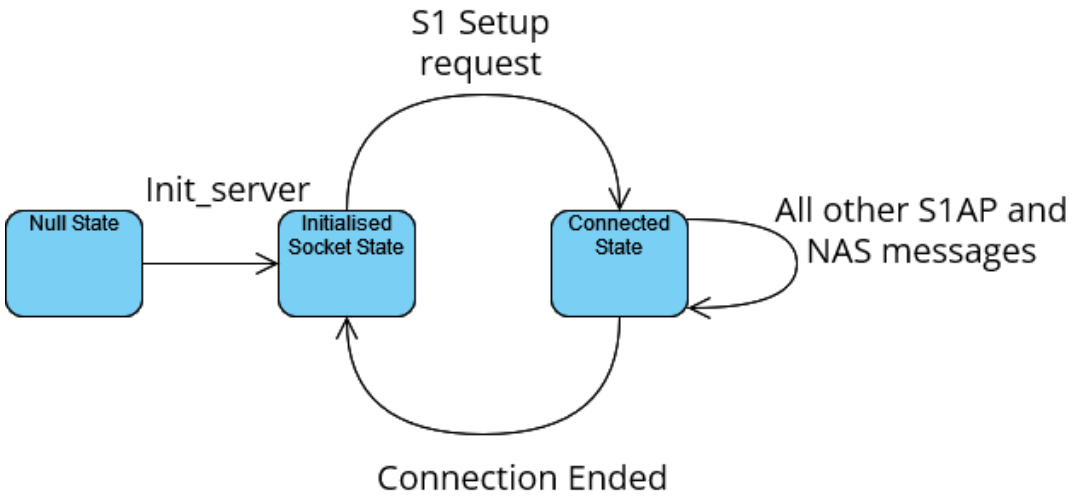

Figure 5.1: Finite state machine of our implementation.

5.4 Code Structure

5.4.1 Division Into Files

The implementation is logically divided into multiple distinct files, visualised on the Figure 5.2.

Intrusive-LTE-MME/
|– MME.py
|– state_machine.py
|– EPC.py
|– parsing.py
|– install_script.sh
|– parsing_test.py
|– EPC_test.py

Figure 5.2: Visualised file structure of our implementation.

The first is the main file, called *MME.py*. It contains the main loop and calls for socket initialisation, object creation, and argument parsing.

The next file is called *state_machine.py* contains only class `EPCStateMachine`, since it is a standalone class independent on other code. This is included inside `EPCServer`.

Another file is *EPC.py*, that contains the class `EPCServer`.

The largest file is *parsing.py*. The file contains class `parsing`, used to dissect, encode and decode messages. There are also some files containing tests and scenarios for testing.

Last but not least, there is a file called *install_script.sh*. It is a simple script trying to simplify and streamline all of the dependencies and required software. The installed dependencies are *pysctp*,*pycrate*, all of the bladeRF utilities (*libbladeRF*,*bladeRF-cli*), *SoapySDR*, and *srsRAN*.

The script contains some duplicity since we install SoapySDR even though it is not required. It is, however, very nice to have. It should be noted that it just stops when something fails - it contains no rollback. It also cannot monitor the results of srsRAN Testing or kernel support for SCTP. Nevertheless, all of the found dependencies were accounted for, and the script is better than searching for guides and libraries for each of the tools online.

5.4.2 Class structure

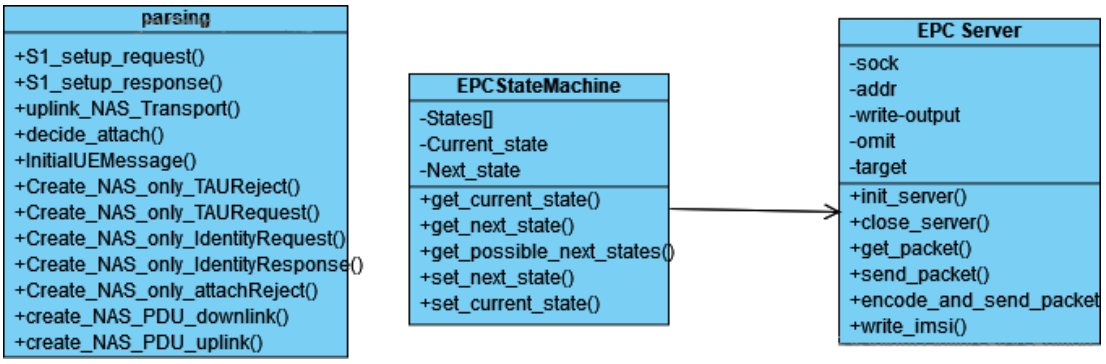

Figure 5.3: Class diagram of our implementation.

In this part, we will describe the code structure of our implementation.

The main component is class `EPCServer`. This component is meant to represent our entire server and encapsulate its behaviour. Inside it are placed other structures, such as `EPCStateMachine`, saved connections and arguments.

5.4.3 Arguments

The program accepts arguments on startup from the console. All of the arguments are optional and have their default value.

The first argument is a response to Attach request. The program accepts both strings and Cause Numbers from the protocol. The default is *Cause #8 - EPS services and non - EPC services not allowed.*

The second argument is a pair of target-omit IMSIs. The user can choose only to target specific IMSIs with the chosen action. Alternatively, the user can choose to exclude specific IMSIs from this action. If the device is targeted or not omitted, it receives the cause specified by the first parameter. Otherwise, it receives *Cause #111 - Protocol Error - unspecified.*

The last argument is the IP address to use for the srsENB connection. The protocol specifies port; therefore, it is impossible to modify it.

Chapter 6

Testing

In this chapter, we discuss the validation and verification of our implementation. There are different paths to test final product. First of all, we go over unit tests, mock eNodeB and test for the correct answer. After this initial check, we test the implementation by experiment. We also discuss practical problems, such as abilities while powered by a battery.

6.1 Isolated Tests

6.1.1 Unit Tests

We used the `unittest` library to test some of our functions in the implementation. These tests are generally used to verify single functions or classes. The functions we focused on were those that are not too self explicable, involved a lot of our coding instead of library calls, and are used by other functions. Some problems were found and fixed.

We encountered a problem with testing the network functions. The unit tests kept blocking each other from using the IP-port pair needed for the functionality. For example, it is not very useful to test functions that serve only as an encapsulation for another library call. Similarly, it does not make much sense to test library implementations of our code, and many functions do just those.

For each of the functions selected for Unit Testing, we created a set of possible parameters and expected outcomes. Using the library `unittest`, we asserted that when running these functions, the outcomes are the as expected. Tests are organised in the files called `*_test.py`, where the first part of the filename is the same as the tested filename. The files are therefore *parsing_test.py*, `EPC_test.py`. The main file has no functions or classes and therefore is not tested.

6.1.2 Black-Box Tests

During the development, we also created a set of eNodeB mockups. With pre-crafted messages, we can compare the answers from the MME to the expected result. These tests present a few prepared scenarios, such as invalid data input or unexpected messages. These tests are black-box in the sense that we do not watch the functionality inside, only the outside appearance of it.

A lot of the mock messages come from the real communication recorded using Wireshark. The expected behaviour was gained using protocol analysis or the communication as presented. Results of these scenarios were checked manually, again using wireshark.

To give an example to these scenarios, i would like to present a few of them.

The first scenario tries to verify that it is possible to create SCTP connection, and establish an S1AP connection. Therefore, it initialises socket and sends S1 Setup request. The expected behaviour is that the connection is established and MME answers with S1 Setup Response.

The second scenario establishes connection, and after that sends Initial UE Message with Attach reject and Identity other than IMSI. It is expected to see Identity request, and we send Identity response. The MME should answer with Attach reject.

6.2 Testing by Experiment

6.2.1 Testing Overview

Sadly, we could not test the application in public. The testing had to be confined into a Faraday cage. This means that our experimentation is minimal - we do not have the option to test things such as the attractiveness of our device to attach. It is also impossible to research how our IMSI catcher behaves when there is legitimate eNodeB transmitting.

6.2.2 Testing Different Phones

First, we inserted different phones into the Faraday cage for the experiment. After that, we started the programs and awaited the results. We also recorded the sessions between MME and eNodeB, as well as communication on MAC layer in the eNodeB. After that, we watched how the different phones reacted to this.

It should be noted that the MAC recordings were done by srsENB. The protocols do not work by default. To make Wireshark able to read MAC recordings, we have to go to *Protocol Preferences -> DLT_USER* and set DLT to *149* and payload to *udp*. [23]

The first tested phone was Google Pixel 4A. It was very reluctant to send TAU or Attach request to our tower. But still, it did. If, during the development, it did not get answer to the Attach request, it would not send more than two or three of them. The phone did try to connect again.

The second was Honor 9X. It was a lot swifter in sending the Attach requests. It also sent up to 5 Attach requests. This phone also did try to connect again after getting Attach Reject #8.

The last tested phone was Xiaomi Redmi 4 Pro. The oldest phone was the fastest in trying to connect. But it also tried to connect again after receiving the Attach Reject #8.

After testing these three phones, it was apparent that the attacks did not carry as great of the weight as first expected when studying this material. Through the recordings of the eNodeB - MME, we are able to see that the communication goes as expected. In the MAC recording, we can clearly see that the Attach Reject is propagated to the mobile phones. It is therefore not clear why these Denial of Service Attacks do not work.

The mentioned recordings are added to the work.

6.2.3 Hardware Performance

We ran the srsENB with 6 Primary Resource Blocks, the lowest setting we can set. With this setting, we got the average CPU load at around 60%, and memory utilisation (including Raspberry Pi OS) at around 1GB. Therefore, we have found that we can use Raspberry Pi 4 with the smallest, 2GB RAM.

6.2.4 Power Requirements

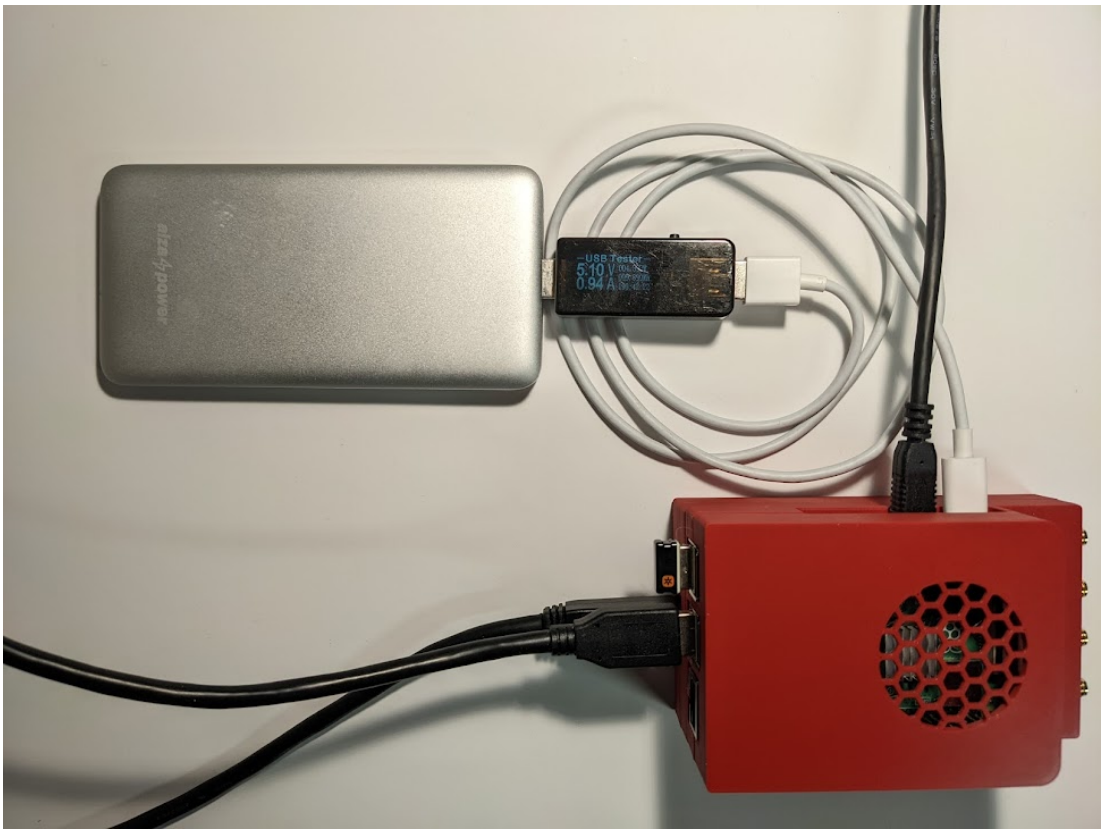

Figure 6.1: Running the setup on the battery. We were able to run BladeRF without second power bank.

The power requirements are key to making this intrusive device work. It must be therefore tested if and how it can be done.

First things first, the Raspberry Pi 4 has an official wall plug charger with nominal characteristics of 5V/3A. The power supply is done over a USB-C port. However, the Raspberry Pi does not exceed 7W (5V/1.4A) while under load. Therefore, it is an exerted rule of thumb that the power supply must be able to provide twice the maximum average power draw so that it can cover the peaks.

Our experiments aligned with these expectations. First, we experimented with running the software on Raspberry Pi OS.

We also tried scaling down the power bank to just a 5V 2A source. However, the Raspberry did not even boot correctly with this power source.

When connected to a 5V/3A power bank, the OS kept popping us with a low voltage warning. Nevertheless, this has stayed the same. On desktop, the draw was at around 1A. However, when running our software, the draw was right under 1.4A. So raising the transmit gain from 30 to 80 increased the draw to 1.6A. This setup is on Figure 6.1.

Considering the official battery capacity on the figure to be 20Ah, we expect our setup to run for around 12 hours.

Chapter 7

Summary

This work has been focused on finding ways to break Wi-Fi and mobile networks using radio, especially Software-Defined Radio. We compared the existing tools in the field, such as tools to create a base station using an SDR. We summarised available hardware in the sphere of SDR. We have also identified past research in the field. We proposed a hardware setup to execute the selected attacks with as small equipment as possible.

We also proposed and developed a software solution that creates a mock of MME. This MME is then used to launch attacks in cooperation with an open-source eNodeB from srsLTE. This mock was tested using unit tests and successfully experimented within the laboratory environment. We tested our solutions using different methods, such as experiment, unit tests and prepared scenarios. We tested the solution while powered on battery, measured its power draw. Thus, we have demonstrated, that making these attacks is possible with the miniaturised hardware, and that it is possible to run it on a battery.

We also tested its effectiveness on different mobile devices. While we were able to execute the attacks as described in earlier work, the attacks were not effective on any of the tested devices. From what we were able to observe, are unable to say, why the attacks were not successful. What also characterises the past attacks and was omitted from the work is trying to trigger TAU. Since we decided to mock MME, it is out of scope of this work.

Bibliography

[1] 3GPP. *LTE; Evolved Universal Terrestrial Radio Access Network (E-UTRAN); S1 Application Protocol (S1AP)*. Technical Specification (TS). 3rd Generation Partnership Project (3GPP), September 2018 [cit. 2022-05-08]. Version 15.3.0. Available at: https://www.etsi.org/deliver/etsi_ts/136400_136499/136413/15.03.00_60/ts_136413v150300p.pdf.

[2] 3GPP. *Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification*. Technical Specification (TS). 3rd Generation Partnership Project (3GPP), October 2020 [cit. 2022-05-08]. Version 15.9.0. Available at: https://www.etsi.org/deliver/etsi_ts/124000_124099/124008/15.09.00_60/ts_124008v150900p.pdf.

[3] 3GPP. *Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3*. Technical Specification (TS). 3rd Generation Partnership Project (3GPP), January 2020 [cit. 2022-05-08]. Version 15.8.0. Available at: https://www.etsi.org/deliver/etsi_ts/124300_124399/124301/15.08.00_60/ts_124301v150800p.pdf.

[4] Alrashede, H. and Shaikh, R. A. IMSI Catcher Detection Method for Cellular Networks. In: *2019 2nd International Conference on Computer Applications Information Security (ICCAIS)*. 2019, p. 1–6. DOI: 10.1109/CAIS.2019.8769507.

[5] Alza.sk. TP-Link Archer T3U Plus - WiFi USB adaptér. *Alza.sk*. 2020 [cit. 2021-09-30]. Available at: https://www.alza.sk/tp-link-archer-t3u-plus-d5816401.htm.

[6] Bassil, R., Elhajj, I. H., Chehab, A. and Kayssi, A. Effects of Signaling Attacks on LTE Networks. In: *2013 27th International Conference on Advanced Information Networking and Applications Workshops*. 2013, p. 499–504 [cit. 2022-05-08]. DOI: 10.1109/WAINA.2013.136.

[7] Burgess, D. A., Samra, H. S. et al. The openbts project. *Report available at http://openbts. sourceforge. net, http://openBTS. org*. August 2008, [cit. 2021-12-02].

[8] Bílý, V. Frekvenční příděl GSM,DCS,UMTS a LTE v České republice. *Gsmweb.cz* [online]. 2021 [cit. 2021-10-24]. Available at: https://www.gsmweb.cz/clanky/freq2.htm.

[9] Chandel, R. Wireless Penetration Testing: PMKID Attack. *Hackingarticles.in*, 24. June 2021 [cit. 2021-12-28]. Available at: https://www.hackingarticles.in/wireless-penetration-testing-pmkid-attack/.

[10] Cilleruelo, C. Why you should not buy the new WiFi Pineapple Mark VII. *Infosecwriteups.com*. September 2020 [cit. 2021-10-23]. Available at:

https://infosecwriteups.com/why-you-should-not-buy-the-new-wifi-pineapple-mark-vii-f388544528c9.

[11] Dunkelman, O., Keller, N. and Shamir, A. A Practical-Time Attack on the A5/3 Cryptosystem Used in Third Generation GSM Telephony. *IACR Cryptol. ePrint Arch.* N/Ath ed. 2010, N/A, p. 13, [cit. 2021-10-25]. Available at: http://eprint.iacr.org/2010/013.

[12] Espressif Systems. ESP32 Wi-Fi and Bluetooth MCU. *Espressif.com.* 2021 [cit. 2021-10-23]. Available at: https://www.espressif.com/en/products/socs/esp32.

[13] Ettus Research. USRP B210 USB Software Defined Radio (SDR). *Ettus.com.* 2021 [cit. 2021-10-24]. Available at: https://www.ettus.com/all-products/ub210-kit/.

[14] GNU Radio. About GNU Radio. *GNU Radio* [online]. 2022 [cit. 2022-05-05]. Available at: https://www.gnuradio.org/.

[15] Golde, N., Redon, K. and Borgaonkar, R. *Weaponizing Femtocells: The Effect of Rogue Devices on Mobile Telecommunications.* 2012 [cit. 2021-10-25].

[16] Hak5. Wifi Pineapple. *Shop.hak5.org.* 2021 [cit. 2021-10-23]. Available at: https://shop.hak5.org/products/wifi-pineapple.

[17] Huber, M. *Breaking LTO on Layer 1, 2, and 3* [Youtube]. ShellCon, November 2018 [cit. 2022-01-16]. Available at: https://www.youtube.com/watch?v=PiO__nr63Lo.

[18] Iconic Devices. Applications of NORVI industrial IoT Devices - NORVI Industrial Arduino. *Norvi.lk.* May 2021 [cit. 2021-10-23]. Available at: https://norvi.lk/applications-of-norvi-industrial-iot-devices/.

[19] Iedema, M. *Getting started with OpenBTS: build open source mobile networks.* N/Ath ed. „ O'Reilly Media, Inc.“, 2014 [cit. 2021-12-02]. ISBN 9781491910658.

[20] Joki, A. and Lux, J. Do faraday cages block cell signal? *Quora.com.* 2019 [cit. 2022-01-16]. Available at: https://www.quora.com/Do-Faraday-cages-block-cell-signal.

[21] Lichtman, M., Jover, R. P., Labib, M., Rao, R., Marojevic, V. et al. LTE/LTE-A jamming, spoofing, and sniffing: threat assessment and mitigation. *IEEE Communications Magazine.* N/Ath ed. 2016, vol. 54, no. 4, p. 54–61, [cit. 2021-10-25]. DOI: 10.1109/MCOM.2016.7452266.

[22] Machado Fernandez, J. Software Defined Radio: Basic Principles and Applications. *Revista Facultad de IngenierÃa.* N/Ath ed. scieloco. January 2015, vol. 24, N/A, p. 79 – 96, [cit. 2021-10-23]. ISSN 0121-1129. Available at: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0121-11292015000100007&nrm=iso.

[23] Miguelez et al. SrsLTE: An Open-Source Platform for LTE Evolution and Experimentation. February 2016, [cit. 2022-01-01].

[24] Mitola, J. Software radios: Survey, critical evaluation and future directions. *IEEE Aerospace and Electronic Systems Magazine.* N/Ath ed. 1993, vol. 8, no. 4, p. 25–36, [cit. 2021-10-23]. DOI: 10.1109/62.210638.

[25] Nikaein, N., Marina, M. K., Manickam, S., Dawson, A., Knopp, R. et al. OpenAirInterface: A Flexible Platform for 5G Research. *SIGCOMM Comput. Commun. Rev.* N/Ath ed. New York, NY, USA: Association for Computing Machinery. October 2014, vol. 44, no. 5, p. 33–38, [cit. 2021-12-02]. DOI: 10.1145/2677046.2677053. ISSN 0146-4833. Available at: https://doi.org/10.1145/2677046.2677053.

[26] Nuand. Products Archive. *Eshop Nuand*. 2021 [cit. 2021-10-24]. Available at: https://www.nuand.com/shop/.

[27] OpenAirInterface. Open Air Interface source code. *OAI Code*. [2014] 2021. Available at: https://openairinterface.org/oai-code/.

[28] OpenLTE. OpenLTE Readme File. *Github Code Repository*. 2021 [cit. 2021-10-24]. Available at: https://github.com/warmchang/openLTE.

[29] P1Sec. P1Sec/pysctp. *Github Code Repository* [online]. November 2021 [cit. 2022-05-05]. Available at: https://github.com/P1sec/pysctp.

[30] P1Sec. P1Sec/pycrate. *Github Code Repository* [online]. May 2022 [cit. 2022-05-05]. Available at: https://github.com/P1sec/pycrate/wiki.

[31] Piqueras Jover, R. Security attacks against the availability of LTE mobility networks: Overview and research directions. In: IEEE, ed. *2013 16th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. 2013, p. 1–9 [cit. 2021-10-25]. ISBN 9781479904631.

[32] Piqueras Jover, R. *LTE security, protocol exploits and location tracking experimentation with low-cost software radio*. July 2016 [cit. 2021-10-25].

[33] Pothosware. SoapySDR wiki. *Github Code Repository* [online]. 2022 [cit. 2022-05-05]. Available at: https://github.com/pothosware/SoapySDR/wiki.

[34] Range networks. *Github Code Repository*, 21. March 2014 [cit. 2021-12-02]. Available at: https://github.com/RangeNetworks/dev/commits/master.

[35] Ryan, G. s0lst1c3/eaphammer. *Github Code Repository*. July 2020 [cit. 2021-10-25]. Available at: https://github.com/s0lst1c3/eaphammer.

[36] Shaik, A., Borgaonkar, R., Asokan, N., Niemi, V. and Seifert, J. Practical attacks against privacy and availability in 4G/LTE mobile communication systems. *CoRR*. 2015, abs/1510.07563.

[37] Software Radio Systems. srsRAN documentation. *Srsran.com* [online]. 2019. 2022 [cit. 2022-05-08]. Available at: https://docs.srsran.com.

[38] Stehlík, R. ESP32 Wi-Fi Penetration Tool. *Github Code Repository*. 2021 [cit. 2021-12-02]. Available at: https://github.com/risinek/esp32-wifi-penetration-tool/.

[39] Stehlík, R. and Pluskal, J. *Útok na Wi-Fi Sítě s Využitím ESP32/8266*. Bozetechova 2, Brno - Kralovo Pole, Czech Republic, 2021. [cit. 2021-12-02]. Bachelor's Thesis. Brno University of Technology.

[40] Van Rijsbergen, K. The effectiveness of a homemade IMSI catcher build with YateBTS and a BladeRF. *University of Amsterdam*. 1st ed. rp.os3.nl. 2016, vol. 28, no. 1, [cit. 2021-10-25].

[41] Venkata, M. *Rogue Base Station Detection Techniques*. Technical Disclosure Commons, January 2021 [cit. 2021-12-31]. Available at: https://www.tdcommons.org/dpubs_series/4001.

[42] Vidano, R. SPEAKeasy II-an IPT approach to software programmable radio development. In: Vidano, R., ed. *MILCOM 97 MILCOM 97 Proceedings*. IEEE, 1997, vol. 3, p. 1212–1215 vol.3 [cit. 2021-10-23]. DOI: 10.1109/MILCOM.1997.644961. ISBN 0-7803-4249-6.

[43] Wojtowicz, B., Murphy, A., He, Z., Bereski, P. and Senyonjo, D. M. OpenLTE download. *Sourceforge.net Software Platform*. [2012] 2021 [cit. 2021-12-02]. Available at: https://sourceforge.net/projects/openlte/.

[44] Yu, C. and Chen, S. On Effects of Mobility Management Signalling Based DoS Attacks Against LTE Terminals. In: *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*. 2019, p. 1–8 [cit. 2022-05-08]. DOI: 10.1109/IPCCC47392.2019.8958725.

[45] Yu, C., Chen, S. and Cai, Z. Lte phone number catcher: A practical attack against mobile privacy. *Security and Communication Networks*. 1st ed. Hindawi. 2019, vol. 2019, no. 1, [cit. 2021-10-25].

Appendix A

Contents of the CD

The following directories and files are present on the CD:
bachelor_thesis.pdf
enb.conf
EPC.py
EPC_tests.py
install_script.sh
latex.zip
MME.py
parsing.py
parsing_test.py
scenarios.py
state_machine.py
pcaps
|—honor_enb.pcap
|—honor_enb_s1ap.pcap
|—honor.pcap
|—pixel_enb.pcap
|—pixel_enb_s1ap.pcap
|—pixel.pcap
|—xiaomi_enb.pcap
|—xiaomi_enb_s1ap.pcap
|—xiaomi.pcap

Appendix B

Installation of Dependencies

The dependencies for this program are pysctp, pycrate bladeRF libraries, and srsRAN stack. We created a simple script to install all of the dependencies required. This file is called *install_script.sh*. To use it, launch bash install_script.sh, in the directory where will all of the dependencies will be cloned.

Appendix C

Quick Start-Up

For launching the program, you need launch two things. The first is MME.py. To launch this, use command `python3 MME.py <opts>` The second is eNodeB. To launch this, use command `srsenb /path/to/enb.conf`. Usually, *enb.conf* is installed with the stack in */.config/srsran* directory. But since it is the only config file acceseed in eNodeB, for good measure, we include one copy in the project folder as well.