Automated 2G traffic interception and penetration testing

Veens, T.

Award date:
2018

Eindhoven University of Technology

Department of Mathematics and Computer Science
Security Research Group

Automated 2G Traffic Interception and Penetration Testing

Master's Thesis

Thomas Veens

Supervisors:
prof.dr.ir. W.P.A.J. Michiels, TU/e
R. Moonen, Secura

Final Version

Eindhoven, 19th October, 2018

Abstract

While mobile phone users have moved on to 4G, there are still many embedded devices around that make use of the 2G cellular network. In the field of penetration testing, security researches are often confronted with such devices that still use 2G data services. Since 2010, it is well known that 2G networks can be spoofed with consumer-grade hardware to trick devices into connecting. The required setup used to be expensive, but has become more affordable due to the development of increasingly powerful hardware. In this document, we use the YateBTS software package and the bladeRF to operate a 2G network for penetration testing.

The first phase of penetration testing requires the researcher to acquire information about the target, which is called reconnaissance. Several steps in this phase allow for automation. Therefore, we introduce a setup and develop software that allows for information to be automatically gathered. However, this requires the researcher to know which 2G network is used by the device. We intend to acquire this information by means of brute-forcing.

Before developing software, we investigate the circumstances required to connect devices to an impersonated network. Furthermore, we investigate the information and communication that can be observed. Finally, we investigate active attacks that can be performed on connected devices.

The developed software will be tested on three embedded devices, after which the findings are reported.

Preface

This master's thesis is the result of my graduation project for the master's program Information Security Technology at Eindhoven University of Technology (TU/e). The project was carried out at Secura in Eindhoven, where the necessary resources to perform the research were provided.

First of all, I would like to thank Wil Michiels for supervising me throughout the research. Secondly, I would like to thank my supervisor at Secura, Ralph Moonen, for providing me this opportunity, support and helpful insight. Additionally, I would like to thank all the amazing colleagues at Secura, including the interns, whom where eager to help whenever I had questions. Thank you for the interesting (lunch) discussions and challenging foosball sessions. Special mention to Bart, Berry, Joe, Mathijs and Nick who never failed to make me laugh.

Finally, I would like to thank my friends, parents and brother for their continuing support. Special thanks to Gijs Rijnders for proofreading my thesis and providing helpful feedback.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

This thesis is the result of the graduation project for the master's program Information Security Technology at Eindhoven University of Technology (TU/e). The program is offered in collaboration with the Radboud University and known as the TRU/e master in cyber security[1]. The graduation project was carried out at Secura[2] (formerly known as Madison Gurkha) in Eindhoven and within the Security research group of the Mathematics and Computer Science department of the TU/e. It was supervised by Wil Michiels (TU/e) and Ralph Moonen (Secura).

1.1 Motivation for the Research

There are more and more Internet of Things (IoT) devices that are used throughout the world every day [52]. IoT can be defined as: the embedding of computing devices in everyday objects to provide an interconnection in order to transfer data. They can be divided into two categories, short-range technologies (e.g. Wi-Fi, Bluetooth, ZigBee) and wide-area technologies (e.g. cellular connections, Sigfox, LoRa and Ingenu). It is anticipated that 70% of the wide-area IoT devices in 2022 will use the cellular network as a means to send and/or receive data [11]. Regarding the use of cellular technologies, GSM still had the largest global market share in 2016. However, this has been changing in the recent and upcoming years [53]. Nevertheless, there are still many devices around that use second generation (2G) data services to send and receive data [5]. We should also consider that devices might fall back to 2G when higher generations are not available. Examples of 2G devices are smart meters [16], car charging stations, connected cars, smart switches, smart home automation, payment terminals and GPS trackers.

The 2G cellular technology is often used in embedded devices for the following reasons. It is cheaper to implement than 3G or 4G, which is particularly favorable when devices have to be mass-produced. Additionally, the power consumption is lower than for these higher generation technologies. These two properties (low

[1] https://true-security.nl/
[2] https://www.secura.com/

cost and power consumption) are often the primary concerns of IoT devices[3]. Furthermore, the geographical coverage of the technology is also an important factor to consider when a vendor has to make a decision on which technology to use. 3G and 4G might not be a available on all locations that a device is expected to function. The downside of 2G is that the maximum throughput is low compared to newer generations. However, this is not a problem when only small amounts of data have to be sent infrequently.

While the technology is still widely in use, 2G has security problems that stem from design limitations [14, 44]. The design does not address an active attack whereby parts of the network are impersonated. It is known that a 2G cell tower can be spoofed with consumer-grade hardware since 2010 [37, 40]. This has been used to make mobile phones connect in order to analyze their traffic and SMS messages. Now that IoT devices have become more common, they are often encountered in the penetration testing domain. Penetration testing is the act of testing a computer system in order to evaluate the security of the system. In this domain, we want to cover all aspects of device connectivity in order to gather information about the device. This is why we would like to intercept the traffic by means of a spoofed cell tower.

Secura has recently shown an interest in such a setup for devices that make use of 2G to transmit data [20]. The goal of the experiment was to obtain a Man-in-the-Middle (MITM) position for the data services by setting up a fake 2G cell tower, see Figure 1.1. In such a MITM position, the communication between two parties is secretly relayed by an attacker, while the two parties assume they are communicating with each other directly. This may allow the attacker to eavesdrop on the communication, intercept or alter it.

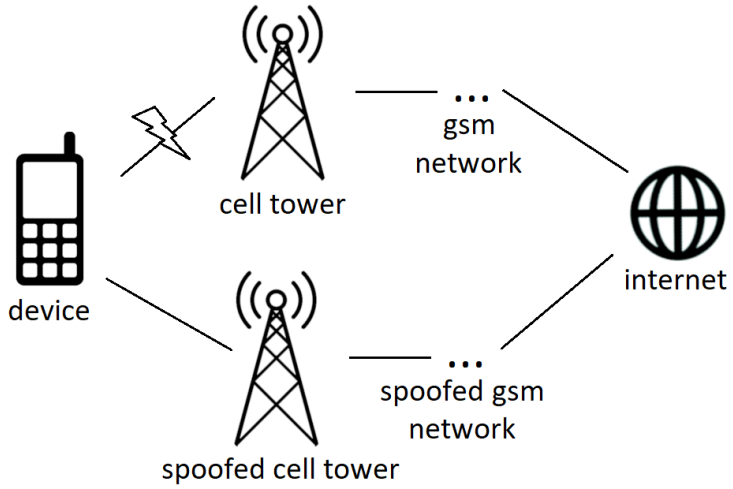


Figure 1.1: A simplified model of a MITM attack for 2G data services.

What is important to realize, is that the 'sunset' of 2G and 3G networks is approaching [19]. However, it approaches at a different rate. In the United

[3]Note that there are IoT focused technologies coming up, such as LoRaWAN, NB-IoT and LTE-M, which intend to provide low cost, low power consumption and low service costs.

States, the 2G sunset has already begun, but in Europe some providers have postponed it in order to "*support legacy machine-to-machine (M2M) and internet of things (IoT) connections that run on 2G*" [39]. In the Netherlands, the sunset of 3G comes first for KPN[4] (2022) and Vodafone (2020), while their 2G network will stay up until at least 2025.

1.2 Problem Statement

To get a better understanding of the problem, first some context is provided, then the attacker model is described and finally the problem description is given.

1.2.1 Problem Context

We define 'device' as equipment that is able to communicate by making use of data services on a 2G network. Given such a device, a security researcher wants to perform a penetration test. Part of such a test consists of connecting the device to a spoofed cell tower in order to:

1. Gather information about the device that might aid the researcher in their further investigation (passive attack). This consists of both passive reconnaissance (e.g. eavesdrop on communication) and active reconnaissance (e.g. a port scan to detect available services).
2. Perform active attacks on the device (e.g. manipulate communication).

1.2.2 Attacker Model

Here the position of the adversary is described to provide a clear perspective of the possibilities. The device (including hardware and software) as well as the cellular network owned by the mobile operator are considered trusted. We assume that the adversary has the necessary equipment to operate a 2G cell tower, has physical access to the device and an Internet connection. In a real-life scenario, whether physical access is realistic depends on the circumstances a device operates in. For example, access restriction in a critical industrial environment might be tightly regulated, but devices in remote locations might not be. Lastly, we also assume the adversary is limited by the constraints of the cryptographic methods that are used by the device and the protocols.

1.2.3 Problem Description

There are multiple 'official' 2G networks available, which are controlled by different mobile operators. A device can be configured to only connect to a particular set of networks. This requires the attacker to determine a network that is used by the device, before it can be spoofed. Brute-forcing is one method to determine this: we spoof different networks until a network is found to which the device connects. However, manual reconfiguration of a spoofed network can be a tedious and time-consuming task.

[4] `https://www.kpn.com/netwerk/3g.htm`

Once a device is connected to a spoofed network, we would like to perform passive attacks (reconnaissance). Based on these results, active attacks might be constructed. While active attacks often have to be constructed manually (e.g. requires reverse engineering of a protocol), the passive attack can often be automated.

1.3 Goal of the Project

Given the problem description, we want to achieve the following goal.

Goal of the project: *Create software that can brute-force 2G networks and automatically perform reconnaissance once a device has successfully connected.*

The following objectives are set answer to achieve the goal:

1. What circumstances are required to have a device connect to a spoofed cell tower?
2. How reliable is this connection?
3. What communication can be observed from a connected device?
4. What discoverable information about the device is relevant?
5. What active attacks can be performed on the device?

1.4 Boundaries

In this section a few boundaries are mentioned in order to further clarify the direction of the project. First of all, it is not part of the objective to discover new (unknown) vulnerabilities in the design of 2G. Secondly, it is not a goal to analyze the cryptography used in the cellular communication in order to find a new way to break it. Finally, the research will be limited to devices compatible with networks that are operated in the Netherlands.

1.5 Contributions

This document intends to make the following contributions.

- Aggregate passive and active attacks that can be performed on IoT devices connected to a spoofed 2G network.
- Software that can assist in penetration tests of IoT devices by brute-forcing 2G networks, automatically perform attacks and offer a platform for further investigation.

1.6 Outline

The remainder of the thesis is structured as follows:

- Chapter 2 provides preliminary and background information about concepts relevant to the problem.
- Chapter 3 covers related work and describes existing uses of 2G network spoofing.
- Chapter 4 introduces the research setup, configuration and covers circumstances required to spoof a cell tower.
- Chapter 5 tests the research setup by performing experiments on devices. The results of the tests are discussed.
- Chapter 6 will identify the relevant information that can be discovered, goes into detail on the communication that can be observed and introduces active attacks.
- Chapter 7 discusses brute-forcing of the networks, presents the tool and findings on test devices. Furthermore, the results are discussed and recommendations for future work are presented
- Chapter 8 provides the conclusions of the thesis.

Chapter 2

Background and Preliminaries

This chapter is used to introduce concepts that will be used throughout the thesis and provide background information. The chapter starts with an overview of the radio frequencies, telephone technologies and terminology, after which network infrastructure and regulations are discussed.

2.1 Electromagnetic Radiation

Before we delve into the further depths of the project, it is a good idea to get at least a basic understanding of radio frequencies and the related concepts. An overview of the electromagnetic (EM) spectrum is also provided and the relevant frequencies are mentioned.

Radio frequencies are electromagnetic radiation, which in turn consists of electromagnetic waves. These waves are formed when an electric and magnetic field come in contact with each other. This reveals the key components of the EM wave: one oscillating wave in the electric field and one in the magnetic field. These two waves are transverse waves, meaning that the oscillations occur perpendicular to the propagation of the wave. Additionally, the two transverse waves are perpendicular to each other, see Figure 2.1 for clarification. EM waves propagate at the speed of light through a vacuum ($\approx 3.00 \cdot 10^8$ m/s) and consist of elementary particles called photons. Photons have zero mass and carry a certain amount of energy. The amount of energy carried by a photon increases at higher frequencies.

The distance over which the shape of a wave repeats is called the wavelength. As frequency increases, the wave length decreases provided that the velocity of the wave stays the same. The following formula is used for the calculation:

$$\lambda = \frac{v}{f}$$

Here, λ represents the wavelength, v the velocity of the wave and f the frequency. Note that when EM waves travel through a medium, the frequency stays unaffected but the velocity can decrease depending on the properties of the medium. This in turn means that the wavelength increases.

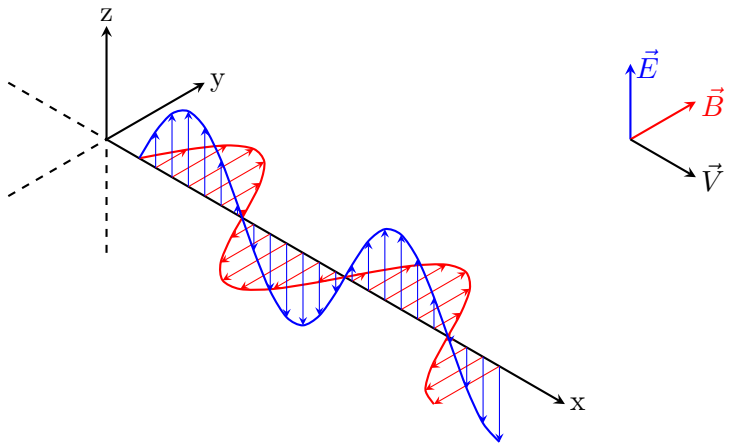


Figure 2.1: A representation of an electromagnetic (EM) wave. Here, $\vec{V}$ represents the direction of the wave, $\vec{E}$ the electric field and $\vec{B}$ the magnetic field.

EM waves may be modeled as a composition of sinusoids. The waves can then be decomposed into individual sinusoidal components. How this decomposition is done, is the field of frequency domain analysis and Fourier transforms. This decomposition, however, is the cornerstone of wireless communication. EM waves that contain multiple transmissions at different frequencies (e.g. radio stations broadcasting at the same time) can be reduced to individual signals. The actual signal that is received by a device depends on the shape, size, material and resonance properties of the antenna. After receiving the signal, the target frequency is filtered and amplified. The resulting signal can then be decoded and used accordingly.

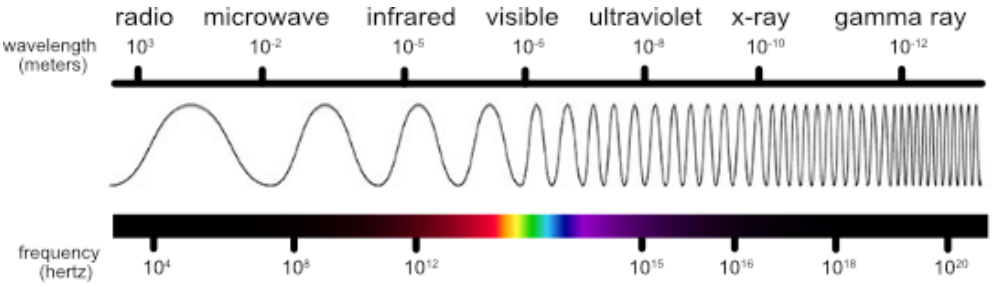


Figure 2.2: An overview of the electromagnetic spectrum.

EM radiation can be classified in radio waves, microwaves, infrared, visible, ultraviolet, x-rays and gamma rays based on wavelength, see Figure 2.2. The radio and microwave classifications are split further into subsets (also called bands) as defined by the International Telecommunications Union (ITU). These bands range from Extremely Low Frequency (ELF) 3-30 Hz to Extremely High Frequency (EHF) 30-300 GHz. The cellular communication that we are interested in happens in the Ultra High Frequency (UHF) 300 MHz - 3 GHz band. The reasoning for this is that lower frequencies penetrate and avoid obstacles easier and have a much greater range with the same transmission power level than

higher frequencies. On the other hand, higher frequencies have more bandwidth (can carry more data) and require a smaller antenna due to a smaller wavelength. For example, a dipole antenna should be about half wavelength for reasonable efficiency [6]. At 900 MHz, the half wavelength is:

$$\frac{1}{2} \cdot \lambda = \frac{1}{2} \cdot \frac{v}{f} = \frac{1}{2} \cdot \frac{3.00 \cdot 10^8}{900 \cdot 10^6} = \frac{1}{2} \cdot \frac{3}{9} \approx 0.17\text{m}$$

Seventeen centimeters is already larger than the length of a mobile phone! Fortunately, a lot of progress has been made in the field of mobile phone antennas in the last twenty years. Designs have been optimized to still fit in modern phones at these larger wavelengths. This is by no means trivial and requires a lot of modeling and prototyping.

In short, what is important to remember is the trade-off between range, penetration of matter, antenna size and bandwidth for a given frequency.

2.2 Generations

This section discusses the different wireless cellular technologies and related standards in order to provide background information. Section 2.2.7 aims to provide a quick overview with the different generations, technologies and data rates.

2.2.1 0G - Pre-cellular

The mobile radio telephone system was pre-cellular and is therefore (retroactively) dubbed the Zero Generation (0G). This system was the first to use a public telephone network as a commercial service, rather than a closed (private) network. These telephones were rather large and most commonly found mounted on vehicles, but mobile briefcase models existed. Analog signals were still used and half-duplex was the norm, although full-duplex was possible (and expensive).

2.2.2 1G - Cellular

The First Generation (1G) cellular technology was introduced in the 1980s. The main difference between 0G and 1G was the use of cells. Geographical areas were divided into sectors (cells) in order to reuse the same frequency without interference, which allowed for increased capacity of the network. This is also known as Frequency Division Multiple Access (FDMA). 1G also still used analog radio signals, meaning that there was no security for transmitted signals and any interference was a problem. In order to avoid interference between users of the network, large gaps in the mobile spectrum were necessary [42]. The mobile spectrum is finite and valuable, so these gaps are expensive. Lastly, it was not yet possible to send text messages or send data, but only make phone calls.

2.2.3 2G

The second generation (2G) was introduced in the 1990s. Digital signals were used, allowing for better quality at lower signal strength than analog signals, see Figure 2.3. It was also possible for communication to be digitally encrypted. Furthermore, 2G allowed for the first data services like Short Message Service (SMS) and Multimedia Messaging Service (MMS). The second generation also still used circuit switching (like the previous generations) rather than packet switching when it was initially released. With circuit switching, first a dedicated channel is established between two nodes of a network before communication takes place.

Multiple different 2G standards were developed: D-AMPS, GSM/GPRS, cdmaOne; but GSM became most popular. Interim Standard 54 (IS-54) and IS-136 are known as Digital-Advanced Mobile Phone Service (D-AMPS). These were once most prevalent in the United States and Canada. However, these technologies have since seen the sunset in both the US and Canada to release the radio spectrum for other cellular technologies.

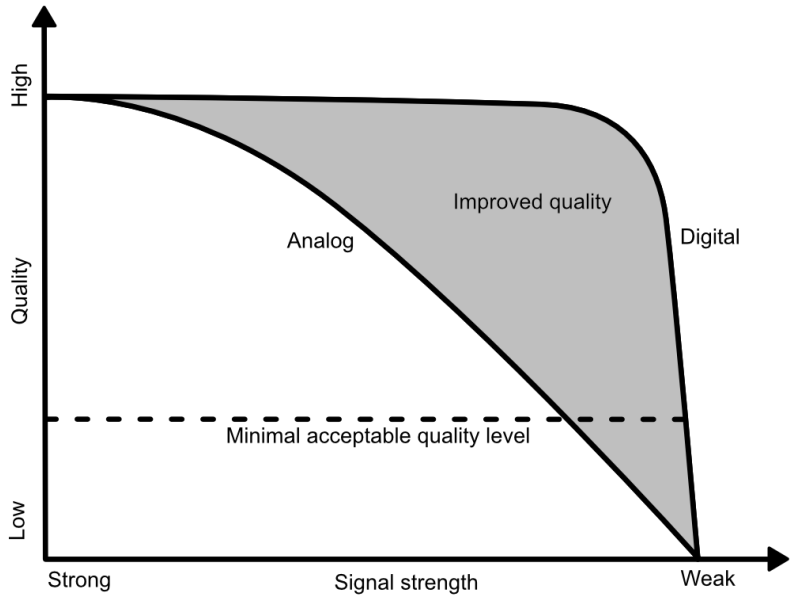


Figure 2.3: A representation of the difference in quality between analog and digital signals at decreasing signal strength.

GSM

Global System for Mobile communications (GSM) is a standard that was developed for 2G and has a 90% market share. GSM is the standard in Europe, but it was adopted by many other countries outside of Europe. Previously, it was called Groupe Spécial Mobile, but this changed as it gained more popularity. In addition to FDMA, it uses Time Division Multiple Access (TDMA). Time slots

were assigned on the same frequency, allowing several users to transmit in rapid succession on the same frequency channel [42].

2.5G - GPRS

Contrary to GSM, General Packet Radio Service (GPRS) is packet switched. With packet switching, messages are broken down into packets which are sent independently. Each packet is routed towards the destination specified in the header. A message can then be reassembled once all component packets have arrived. GPRS was introduced to allow packet-switched data protocols like IP to (reliably) run over 2G. It was possible to run IP over SMS previously, but the performance was unsatisfactory. GPRS attempted to improve this, allowing for packet-switched data with a theoretical peak speed of about 114 kbit/s. Theoretically, it is possible to achieve 160 kbit/s, but this requires 8 timeslots to be assigned to a single user. This increase in speed was achieved by making use of idle channels in TDMA and the maximum speed depends on the coding scheme that is used. In practice, however, the speed was much lower: about 40 kbit/s. Note that GPRS is a best-effort service, meaning that the performance depends on the number of users that are concurrently using the service.

GPRS made several additional services possible: higher SMS throughput, (always-on) internet access, MMS, push-to-talk over cellular, instant messaging and more. With SMS over GPRS, an SMS transmission speed of about 30 messages per minute can be achieved rather than 6-10 per minute with GSM. The service can typically be recognized on a mobile phone by the "G" icon in the notification bar, see Figure 2.4.

2.75G - EDGE

In order to improve GPRS, Enhanced Data Rates for GSM Evolution (EDGE) was developed and first deployed in 2003. EDGE is also known as Enhanced GPRS. It has a maximum data rate of about 384 kbit/s. Due to this higher bandwidth, EDGE qualifies as a 3G technology (the requirement is more than 200 kbit/s). When a mobile phone is making use of an EDGE connection, this can be recognized by the "E" icon in the notification bar as can be seen in Figure 2.4.

EDGE has since been improved into Evolved EDGE. Latencies have been cut in half and peak speeds of over 1 Mbit/s are achieved. Unfortunately, it has not seen widespread deployment on commercial networks.

(a) GPRS (b) EDGE

Figure 2.4: The GPRS and EDGE icons on Android.

cdmaOne

IS-95, also known as Code Division Multiple Access One (cdmaOne), was developed by Qualcomm. It was the first digital cellular technology to use Code Division Multiple Access (CDMA) [42]. With CDMA, simultaneous access of a wireless medium is supported by using different codes to transmit data. If these codes are orthogonal, it is possible for multiple users to use the same frequency band at the same time. The data that was sent by each user can then still be extracted from the combined signal.

cdmaOne was first deployed in 1995 and was used in the US and parts of Asia. Most importantly, cdmaOne and the use of CDMA in digital cellular technology was the stepping stone for 3G.

2.2.4 3G

The International Telecommunications Union (ITU) created requirements for a family of standards for the third generation. This set of requirements is called the International Mobile Telecommunications for the year 2000 (IMT-2000). The most important changes for the Third Generation (3G) are mutual authentication, improved encryption standards and faster network speed. The requirement for 3G was a transfer rate of 144 kbit/s for users with high mobility. Global Positioning System (GPS) was supported as well as mobile television and video conferencing. Also for the third generation multiple standards were developed. The most important standards will be discussed next.

UMTS

The previously mentioned GSM standard evolved into Universal Mobile Telecommunications System (UMTS) and was inspired by the use of CDMA. It is primarily used in Europe, Japan, China and other regions that were previously dominated by GSM. The first commercial deployment of UMTS was in 2001 and had a transfer speed of 384 kbit/s. It was developed by the 3rd Generation Partnership Project (3GPP), which is a collaboration between various telecommunication associations. The initial goal of the organization was to create a globally applicable 3G specification based on GSM. Later, the partnership became responsible for maintenance of GSM, GPRS, EDGE, UMTS, HSPA, LTE and upcoming 5G standards. Some of these technologies have not been discussed yet and will be explained later.

Since 2006, many UMTS networks have been upgraded to 3.5G, also known as High-Speed Packet Access (HSPA). HSPA is a combination of HSDPA (downlink) and HSUPA (uplink), providing even better connection speeds and reduced latency. Downlink refers to the communication from the cell tower to the mobile device. Uplink to the communication in the other direction. HSDPA aimed for a peak downlink data rate of 14.0 Mbit/s. HSUPA aimed for a peak uplink data rate of 5.76 Mbit/s.

In 2010, worldwide adoption of Evolved High Speed Packet Access (HSPA+) began. Downlink data rates of up to 42 Mbit/s can be achieved. New technolo-

gies like beamforming and Multiple-Input Multiple-Output (MIMO) communications are used. Beamforming is a technique for directional signal transmission and/or reception. It is used to focus the transmission power of an antenna in the direction of the user. MIMO exploits multi-path propagation by using multiple transmit and receiving antennas in order to multiply the capacity of a radio link. With Advanced HSPA+ (an improvement on top of HSPA+) even higher data rates can be achieved, see [41].

CDMA2000

The previously mentioned cdmaOne evolved into CDMA2000, which was developed by the 3rd Generation Partnership Project 2 (3GPP2). 3GPP2 is a collaboration between telecommunication associations established in order to upgrade cdmaOne networks to 3G under the standard CDMA2000 that competes with UMTS. Therefore, CDMA2000 is primarily used in regions where cdmaOne was previously dominating, like North America and South Korea. The first phase, CDMA2000 1X had a peak data rate of 144 kbit/s. The improved version increased the data rate to 307 kbit/s. With CDMA2000 1xEV-DO (Evolution, Data Optimized), the downlink first increased to 2.4 Mbit/s and later to 3.1 Mbit/s with revision A. Later, revision B raised the peak data rate to 14.7 Mbit/s. Unfortunately, this is still a long way to go from the speed that HSPA+ achieved.

2.2.5 4G

The Fourth Generation (4G) cellular technology has a requirement of 100 Mbit/s for high mobility and speeds of 1 Gbit/s on foot. Additionally, the technology has to use an all-IP packet switched network. These are the most noteworthy requirements for the International Mobile Telecommunications-Advanced (IMT-Advanced) standard. For 4G, Orthogonal Frequency Division Multiple Access (OFDMA) is used. OFDMA allows for an even more efficient use of the frequency spectrum by spacing channels even closer together using orthogonal subcarriers, removing the need for guard bands between channels.

There are two competing standards under development for 4G: Long Term Evolution (LTE) and IEEE 802.16m (also known as WiMAX). Ultra Mobile Broadband (UMB) was the intended successor to the CDMA2000 technology. However, Qualcomm announced it would end development in favor of LTE.

The first LTE release did not meet the required data rates, but had a downlink speed of 100 Mbit/s and an uplink speed of 50 Mbit/s. LTE was already marketed as 4G, so ITU decided to keep the name. LTE was proposed in 2004, but publicly available commercial networks were launched at the end of 2009. Networks were launched in Europe and North America, although operators considered the rival standards UMB and WiMAX. Eventually, LTE became popular and commercial networks were launched in the majority of countries and regions. It became the global leader of the standards, with the largest number of subscribers. Later, LTE evolved into LTE Advanced to fulfill the required data rates

specified in IMT-Advanced. LTE Advanced was standardized in 2011. The capable data rates were 1 Gbit/s downlink and 500 Mbit/s uplink. A number of test networks have been deployed to test LTE Advanced (or 4G+) in the field. Currently there are commercial networks available, but the coverage is limited; it is usually found around cities.

Mobile WiMAX or IEEE 802.16e is a standard that started off with peak data rates of 37 Mbit/s downlink and 17 Mbit/s uplink. The first commercial mobile network was released in 2006. As the data rates increased with new releases, the standard was also considered as 4G, even if it did not fully meet the requirements. With the coming of IEEE 802.16m or WiMAX 2 the IMT-Advanced criteria have been fulfilled by providing data rates of up to 1 Gbit/s. WiMAX networks have been deployed in North America and some regions in Asia; it is most dominant in South Korea.

2.2.6 5G

The upcoming Fifth Generation (5G) is supposed to provide an even greater connection speed. Data rates of up to 20 Gbit/s should be achieved by the use of higher frequencies, e.g. 26, 28, 38 and 60 Ghz bands. It will make use of both beamforming and MIMO. Since 2017, 5G is being actively developed by Samsung, Intel, Qualcomm, Nokia, Huawei, Ericsson, ZTE and others. The use of 5G was demonstrated at the 2018 Winter Olympics in South Korea. It is planned to be available worldwide by 2020.

2.2.7 Overview

Symbol	Standard	Name	Maximum downlink speed	Maximum uplink speed
2G	GSM	Global System for Mobile communications	9.6 Kbit/s	9.6 Kbit/s
G	GPRS	General Packet Radio Service	114 kbit/s	20 kbit/s
E	EDGE	Enhanced Data Rates for GSM Evolution	384 kbit/s	60 kbit/s
3G	UMTS	Universal Mobile Telecommunications System	384 kbit/s	64 kbit/s
H	HSPA	High-Speed Packet Access	14.0 Mbit/s	5.76 Mbit/s
H+	HSPA+	High-Speed Packet Access - Release 7-10	21.1-168.8 Mbit/s	11.5-23.0 Mbit/s
(Pre)-4G	WiMAX	Worldwide Interoperability for Microwave Access	6 Mbit/s	1 Mbit/s
(Pre)-4G	LTE	Long Term Evolution	100 Mbit/s	50 Mbit/s
4G	LTE-A	Long Term Evolution - Advanced	1 Gbit/s	500 Mbit/s
4G	WiMAX 2	Worldwide Interoperability for Microwave Access 2	1 Gbit/s	500 Mbit/s

Table 2.1: Overview of cellular technologies and their bandwidth.

2.3 Terminology

This section will explain some terminology and concepts that are used in the context of mobile telecommunication and 2G in particular.

2.3.1 Frequency Bands

As previously mentioned, a range of frequencies (subset of the spectrum) is called a (frequency) band. They are often referred to by their respective frequencies,

e.g. the GSM-900 band ranges from 880 to 960 MHz. A frequency band is usually divided into channels with a certain step size. For example, GSM-900 uses channels of 200 kHz: 890, 890.2, 890.4. To separate channels and/or bands, sometimes a guard band is used. A guard band is an unused part of the radio spectrum in order to prevent interference between two bands. The frequency bands that 2G operates on in Europe (and most parts of the world) are E-GSM-900 (Extended) and GSM-1800 MHz. The 1800 band is also referred to as DCS-1800. In North America the GSM-850 and GSM-1900 MHz bands are used. The 1900 band is also referred to as PCS-1900.

2.3.2 SIM

Subscriber Identity Module (SIM) refers to an integrated circuit which intends to securely store data that is used to identify and authenticate a mobile telephone device. The term used to refer to both the hardware and software, but with the release of UMTS the Universal SIM (USIM) was introduced and resulted in the old term being split. Therefore, modern SIM cards actually contain both the old SIM application and the newer USIM application. What is explained here, is relevant data of the older SIM application that is used to make communication with the a GSM network possible. The most important parameters stored by the SIM are mentioned:

- The Integrated Circuit Card Identifier (ICCID) is a unique identifier for the SIM card itself, it is read-only.
- The International Mobile Subscriber Identify (IMSI), which is composed of the Mobile Country Code (MCC), Mobile Network Code (MNC) and the Mobile Subscriber Identification Number (MSIN). The combination of MCC and MNC identify the network: they represent the country and provider respectively. The MSIN is an identifier for the subscriber to a particular provider in a country. Note that the combination of MCC, MNC and MSIN (and hence the IMSI) has to be unique. It is used to uniquely identify a subscriber; hence, this can be sensitive data.
- A 128-bit secret key k_i, which can be used to authenticate the subscriber.
- The session key k_c, used for encryption of the air interface, i.e. the communication between the mobile device and the cell tower. The session key can be updated, but the same key may be used for days.
- The Location Area Identity (LAI), which is comprised of the MCC, MNC and Location Area Code (LAC). It represents an internationally unique identifier used for location updating of subscribers. The LAC is used as unique identifier for a location area, allowing for 65534 unique areas within a network. When the device is power cycled, the LAI is taken from the SIM card in order to look for a previous location or perform a location update. When the location is changed by moving the device, the new LAI is stored on the SIM.

2.3.3 GSM Networks

A Public Land Mobile Network (PLMN) provides mobile telecommunication services to the public. It is composed of the MCC and MNC. The LAC and the Cell ID (CID) together define a unique cell within a PLMN. The CID identifies a cell tower or a sector covered by the cell tower. Note that the four parameters (MCC, MNC, LAC and CID) together provide a location estimate. In some contexts, this can be sensitive data.

With respect to cellular networks, the words uplink and downlink are often mentioned. Downlink refers to the communication from the cell tower (and the rest of the network) to the mobile device. Uplink refers to the communication from the mobile device to the cell tower (and the rest of the network).

2.4 2G Network Infrastructure

Since the 2G technology (and GPRS in particular) will be the main focus of this thesis, it is useful to get a basic understanding of the network infrastructure. The relevant different components of the network are explained below, and a simplified overview can be found in Figure 2.5.

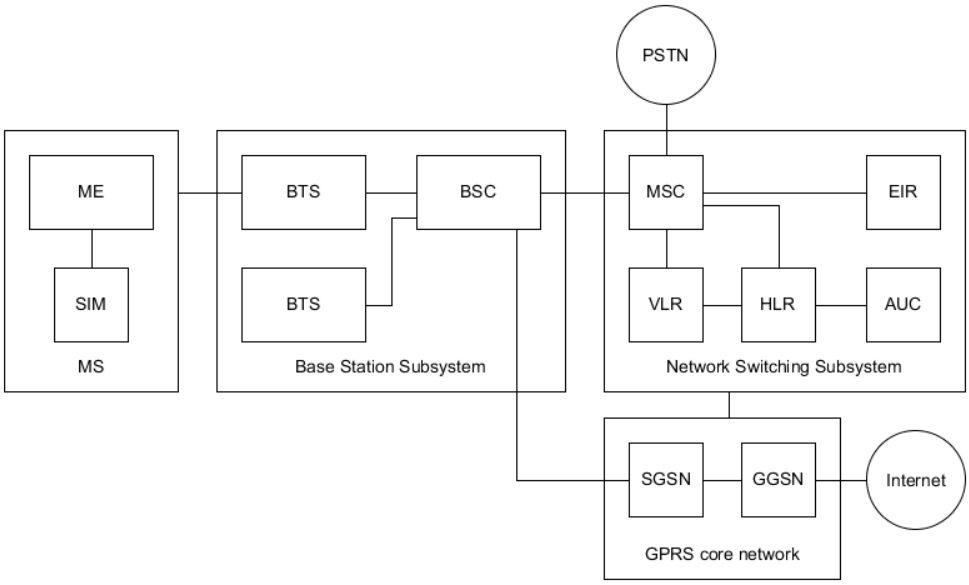


Figure 2.5: A (simplified) 2G network overview.

- The term mobile station (MS) is used to refer to a device that is able to communicate via the 2G network. This implies that the MS must contain a valid SIM card that is registered with the network operator. Therefore, the mobile station can be split into mobile equipment (ME) and the SIM. The ME contains a International Mobile Equipment Identity (IMEI) that is used to uniquely identify it.

- The base tranceiver station (BTS) is the wireless connection between the mobile station and the rest of the network. Informally, it relays communication from the MS and the rest of the network. A BTS is commonly referred to as a "cell tower" and covers a certain area (cell) in a cellular network. The size of the cell depends on the landscape and population. With large cells, repeaters can be used to relay the communication between the BTS and mobile stations. The network operator has to make sure that two adjacent base tranceiver stations do not use the same frequency in order to prevent interference.
- The Base Station Controller (BSC) covers multiple base tranceiver stations (tens or even hundreds). It makes sure that the transition of a mobile station between different cells under the same BSC is handled correctly. This process (switching cell towers) is also referred to as the 'handover'.
- The Mobile Switching Center (MSC) takes care of inter-BSC handovers. It is also responsible for authentication and the routing of calls and SMS messages. Therefore, it might need to interface with the Public Switched Telephone Network (PSTN). The PSTN is the aggregation of all public circuit switched telephone networks.
- The Home Location Register (HLR) is used to store subscriber information like the IMSI, phone number and which services are allowed to be accessed. There is one HLR for every provider, although this can be a distributed database.
- The Visitor Location Register (VLR) is used to keep active subscriber information for connections to the MSC. It holds the Temporary Mobile Subscriber Identity (TMSI), which is a temporary name for the IMSI. The TMSI is used to avoid subscribers from being identified, therefore it can be changed in the network at any time to make it harder for eavesdroppers to identify subscribers. Part of the information that is stored is retrieved from the HLR.
- The Authentication Center (AUC) is used to handle the authentication to the network. This means that the key (k_i) of every SIM belonging to a registered subscriber of the network is stored here. The AUC is also responsible for generating random challenges for authentication purposes.
- The Equipment Identity Register (EIR) is a service that holds the IMEI numbers of banned or stolen mobile equipment. The EIR is not implemented in every network.
- The Serving GPRS Support Node (SGSN) handles the packet switched data in the network. Its functionality can be compared to that of the MSC for circuit switched data. This means that the SGSN is responsible for authentication of GPRS devices and routing of data packets to and from mobile stations. It interfaces with the GGSN to relay data.
- The Gateway GPRS Support Node (GGSN) is responsible for the interfacing between external packet switched networks (like the internet, or a sub-network) and the rest of the GPRS network. The rest of the GPRS network is hidden from an outside perspective, it is viewed as a router. Once data comes in, it is forwarded via the SGSN to the corresponding

mobile station. If the mobile station is inactive, the data is discarded. Similarly, data coming from a mobile station is routed to the correct network via the GGSN.

Chapter 3

Related Work

Daehyun Strobel explained the weak point of one-sided authentication in GSM, which can be exploited by an IMSI catcher [8]. An IMSI catcher is a 'fake' base tranceiver station (BTS) which can be used to trick a device into connecting to it. Due to lack of mutual authentication in the 2G network, encryption on the radio traffic can be disabled. This allows for the IMSI, outgoing SMS messages and outgoing voice traffic to be captured.

In 2010 on DEF CON 18, Chris Paget demonstrated how an IMSI catcher can be operated with consumer affordable hardware and open source software [37]. IMSI catchers were previously only employed by intelligence and law enforcement agencies, because the professional hardware to operate them was expensive.

Early 2011, David Perez and Jose Pico extended the attack of an IMSI catcher to a practical attack on GPRS data services [40]. They were able to set up a rogue BTS to gain full control over a victim's 2G data communication. Both due to lack of mutual authentication in 2G and due to disabling GPRS encryption for devices.

In 2016, Kenneth van Rijsbergen has shown the effectiveness of an IMSI catcher using YateBTS and a bladeRF for Open Source Intelligence [45]. Kenneth concluded that IMSI catchers require specific conditions to be effective against modern phones, because newer generation cellular technologies are available.

There are a several parameters that increase the likelihood of a device connecting to an IMSI catcher, increase the frequency of location updates or make it unlikely for the device to switch to another network when connected to the IMSI catcher. Peter Ney and Ian Smith aggregated many of these parameters and use them as signatures for a cell-site simulator detection network [21]. While we do not intend to cover detection prevention for spoofed 2G networks, we do want to explore these parameters for device connectivity and information gathering.

Remarkably, the effect of IMSI catchers and spoofed 2G networks on IoT, M2M and embedded devices has been relatively unexplored. Where GPRS is not suited for casual browsing on mobile phones, these devices only require a small amount of bandwidth. Furthermore, these devices typically only support 2G and not any of the higher generations, which can make spoofed networks more effective than on modern mobile phones.

Chapter 4

Research Setup

This chapter describes the setup that was used to perform the research and to achieve the automation as described in Section 1.3. In short, the following steps were taken:

1. The selection of hardware and software required to configure and operate a BTS.
2. Configurations of the setup in order to:
   (a) Operate a test network that eases configuration and testing of tools.
   (b) Spoof a BTS belonging to a mobile operator.
3. Extension of the setup to ensure a MS connects to the spoofed BTS.
4. Extension of the setup to allow for brute-forcing of different networks.

4.1 BTS Software

By implementing the GSM and GPRS protocol in software, a major cost reduction for equipment can be achieved. Dedicated hardware is replaced by general-purpose hardware and software implementations. Additionally, a software implementation adds flexibility to the operation of a network. The software can be changed and upgraded rather than replacing or reprogramming hardware. Before looking at the available software, some requirements are specified.

1. The software should be convenient to setup and restart.
2. Capturing GSM and GPRS traffic should be possible.
3. The software should be compatible with affordable hardware (roughly €500).

The following three software implementations will be briefly discussed: OpenBTS, YateBTS and OsmoBTS. Do note that these packages are intended to set up small networks with only a handful of subscribers and that 100% uptime is not a requirement.

4.1.1 OpenBTS

OpenBTS is known to be the first free software implementation of a BTS. It includes support for GSM and GPRS, but not EDGE. Rather than forwarding voice and SMS traffic to a MSC (see Figure 2.5), it is delivered via Session Initiation Protocol (SIP) using a Voice over IP (VoIP) switch. This means that a GSM mobile station connected to an OpenBTS cell tower is turned into a SIP endpoint. Hence, it combines VoIP and the GSM air interface to create a new type of cellular network that can be operated at lower cost in various applications (e.g. private networks) compared to using other technologies. The VoIP switch can be operated on the same host that runs OpenBTS, allowing for the operation of a cellular network on one computer. In order to setup such a cellular network on one host, the following services will need to be run:

- Sipauthserve. The SIP authorization server for traffic related to device registration. It also maintains the subscriber registry database.
- Smqueue. The store-and-forward message service for SMS functionality.
- A VoIP soft switch like Asterisk, Freeswitch or Yate.
- OpenBTS.

While these services can run on one PC, a radio transceiver is still required for the air interface between the BTS and the MS. Initially, only Ettus Research USRP boards were supported. Over time, support grew for the Range Networks RAD1, Fairwaves UmTRX and the Nuand bladeRF.

OpenBTS was used in a demonstration at DEF CON (2010) [37], showing that a BTS could be spoofed to intercept GSM traffic.

4.1.2 YateBTS

YateBTS was first released in 2014 and used part of the OpenBTS project. This derivation of OpenBTS was dubbed MBTS, which allowed it to be licensed independently. MBTS is used as an interface between the radio transceiver and YateBTS, allowing for a layer of abstraction (see Figure 4.1). The goal of YateBTS was to create a software package that had less external dependencies and a modular design, such that the functionality can be extended with scripts. It relies on Yet Another Telephony Engine (Yate) to be used as soft switch for VoIP. Furthermore, YateBTS offers support for GSM and GPRS, but not EDGE. It does, however, offer a Network in a PC (NIPC) mode such that a cellular network can be setup on one PC without launching all services independently. Unfortunately, YateBTS has a rather limited hardware selection. It supports the same radio transceivers as public release of OpenBTS on which it was built and provides excellent support for the Nuand bladeRF.

4.1.3 OsmoBTS

OsmoBTS is another software implementation of a BTS and it is part of the Open Source Mobile Communication (Osmocom) project. It has support for both GPRS and EDGE. In order to operate, it requires interfacing with a BSC.

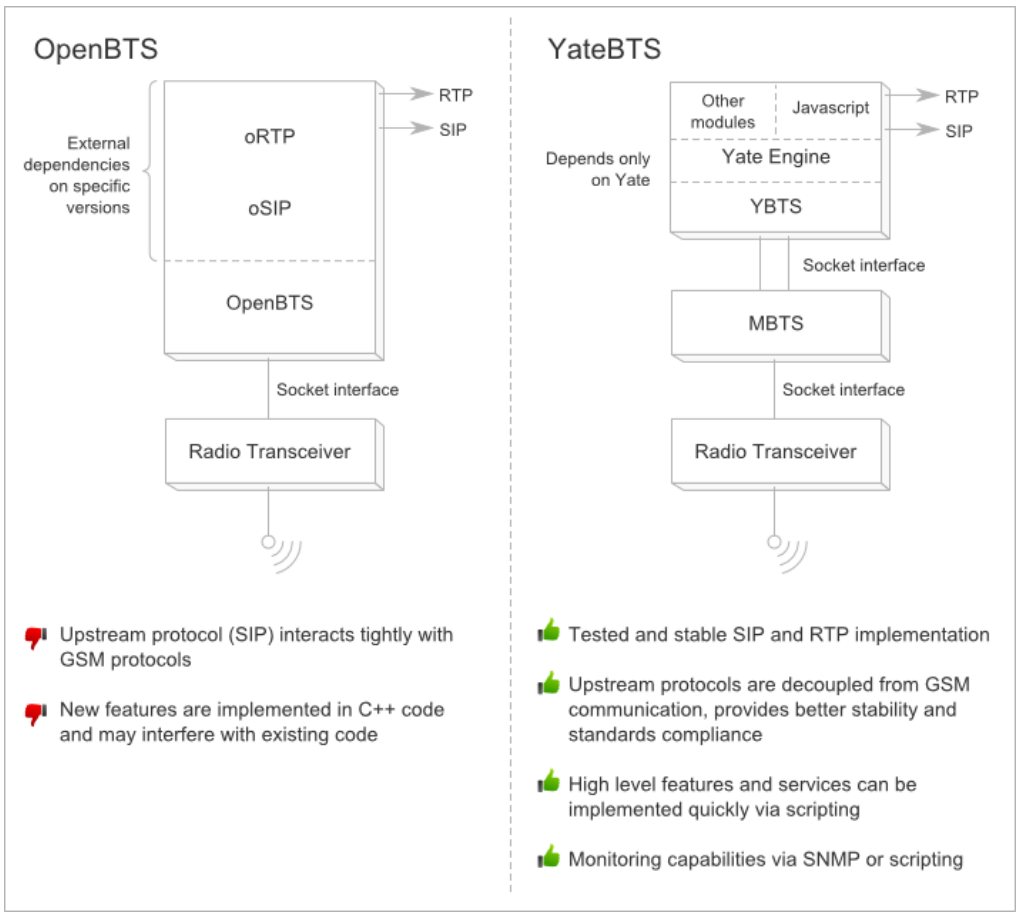


Figure 4.1: Improvements of YateBTS compared to OpenBTS [48].

Fortunately, it is modular and supports multiple back-ends. By using OpenBSC running in Network in the Box (NITB) mode, a cellular network can be operated on one PC. In order to do so, the following services must be run:

- OpenBSC running in NITB mode.
- OsmoBTS.
- OsmoTRX that is a bridge between the radio transceiver and OsmoBTS. When dedicated BTS hardware is used, sysmoPHY or octPHY can be used to interface with OsmoBTS.

OsmoBTS supports dedicated BTS models such as sysmoBTS, OCTPHY and LiteCell. However, it also provides support for a few SDRs such as the Ettus Research USRP, UmTRX and LimeSDR.

4.1.4 Selection

After consideration of the three software implementations (Table 4.1) and based on previous experience [20], YateBTS was chosen as the most suitable software

Software Implementation	Pros	Cons
OpenBTS	Wide variety of hardware supported. Extensive online documentation.	No EDGE support. Various services will have to be (re)started to operate.
YateBTS	Cheap hardware supported. Extensive online documentation. All-in-one-service that can be (re)started.	No EDGE support.
OsmoBTS	EDGE support.	Expensive hardware Limited documentation, aimed at GSM experts. Various services will have to be (re)started to operate.

Table 4.1: The pros and cons of the various GSM software implementations.

package. YateBTS is particularly convenient when the network has to be reconfigured and restarted, because it does not depend on other services or processes. Additionally, it supports affordable hardware: the Nuand bladeRF and the USRP. Furthermore, there are options available to tap both GSM and GPRS traffic in the configuration file [61].

4.2 Hardware Selection

Now that the software has been selected, radio transceiver hardware is required. More specifically, a hardware component is required to send and receive on the 900 and 1800 MHz bands. Previously, rather expensive and dedicated setups were required for radio communication. Nowadays, a selection of affordable Software Defined Radios (SDRs) are available. A host device is also selected to run the BTS software we selected (YateBTS).

4.2.1 SDR

An SDR is defined as a concept that achieves Radio Frequency (RF) communication by making use of software to perform signal-processing that is usually done by hardware. This can be for receiving, transmitting or both; the point is that software processing is used somewhere in the device. You might wonder what the advantage is of using a general purpose processor over dedicated hardware, since special hardware is much more efficient. An answer to this is flexibility: many different protocols, functionalities and applications can be supported. Function-

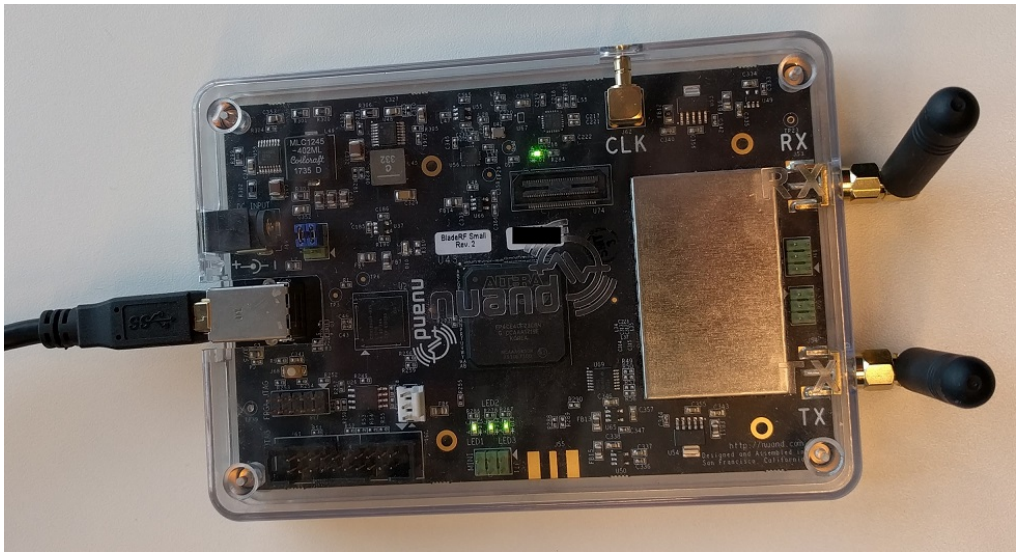


Figure 4.2: The Nuand bladeRF x40.

ality of the device can easily be adapted by modifying and writing new software to the device. A rather powerful processor is required to do signal processing on a software level. SDRs have become much more affordable in recent years due to innovation in hardware.

There are a fair amount of consumer SDRs available on the market with different specifications. However, the available options become rather limited when the following requirements are made:

- GSM is a full-duplex protocol (i.e. simultaneous transmission and reception), which implies that a full-duplex SDR is required.
- The 900 and 1800 MHz frequency bands must be supported.
- The SDR must be supported by the YateBTS software.

The SDR that is recommended by YateBTS is the bladeRF by Nuand [24]. This SDR has two variants: x40 and x115. The only difference being the price and the size of the FPGA. There are no intentions to write custom software for the FPGA and YateBTS comes with FPGA firmware images that work with both versions. This means that the bladeRF x40 will be sufficient, see Figure 4.2.

Since the bladeRF does not come shipped with antennas, two quad-band GSM antennas were acquired (model SPK-WA-WTH43003) [51]. One for Reception (RX) and one for Transmission (TX). These antennas are capable of operating on the GSM-850, E-GSM-900, DCS-1800 and PCS-1900 bands.

Clock Accuracy

According to the GSM 05.10 specification, a BTS has a rather high requirement for the accuracy of the clock: *"The BTS shall use a single frequency source of absolute accuracy better than 0.05 ppm for both RF frequency generation and clocking the timebase."* [12]. This implies an error of 45 Hz at 900 MHz and 90

Hz at 1800 MHz. In a lab setting with only a couple devices, this requirement can be relaxed to 0.2 to 0.4 ppm [7]. The problems associated with inaccurate clocks can be read here [34]; in the worst case, the network can not be found by a mobile station. Since the specification is rather strict, the clock on the SDR has to meet these high expectations. For example, in previous/stock versions of the USRP, an external clock had to be connected to meet the expectations. In fact, the developers of the bladeRF took the requirement of an accurate clock into account. The clock in the bladeRF comes factory calibrated within 1 Hz of 38.4 MHz at room temperature or 0.02 ppm. Should these circumstances change (e.g. the board sitting in the sun), then the accuracy could deviate up to 1 ppm. Additionally, the factory calibration might not be valid for more than a year due to component aging. This requires the user to recalibrate the clock, kalibrate-bladeRF [26] can be used for this. The software makes use of the Frequency Correction Channel (FCCH) broadcast by base tranceiver stations of GSM networks to update the voltage value of the clock.

4.2.2 Host Device

There are many different options to consider for a host device that will be interfacing with the SDR by running the BTS software. For that reason, first some requirements are stated:

- The setup should be mobile and battery powered.
- USB 3.0 should be supported.
- Enough processing power should be available to handle the data throughput from the SDR while still able to run the BTS software and other applications (e.g. to capture and/or analyze data).

A single-board computer is preferred over a laptop, because it allows the setup a high degree of mobility, remote access (e.g. via SSH) and headless operation. An SDR deals with high data flows to and from the device. The bladeRF was designed to work best with USB 3.0 to allow for the large amount of data to flow through. In addition to that, USB 2.0 might not deliver enough power to operate the bladeRF without malfunction [27]. Besides the operation of YateBTS on the host, tools will be run to for example capture, analyze and/or manipulate traffic as well as scan for services. Because of this, we would like to have enough processing power available to run all the necessary software.

The Odroid XU4 (Figure 4.3) [31] was chosen. It is a relatively cheap single-board computer (less than 100 dollars), but roughly 7 times more powerful than a Raspberry Pi 3. The Odroid has an octa-core processor (ARM Cortex A15 and A7), 2 GB of DDR3 RAM, USB 3.0 support and an eMMC flash storage socket. Rather than booting from micro SD, an eMMC module is used which provides roughly 5 times faster write speed and 7 times faster read speed [31]. It comes with extensive documentation and a wiki, which should ease the configuration. Finally, it offers General Purpose Input/Output (GPIO) pins, which are convenient when other devices have to be controlled.

In order to develop and test the tools and software, a laptop running Windows 10 with Kali Linux 2017.2 in a VM was used.

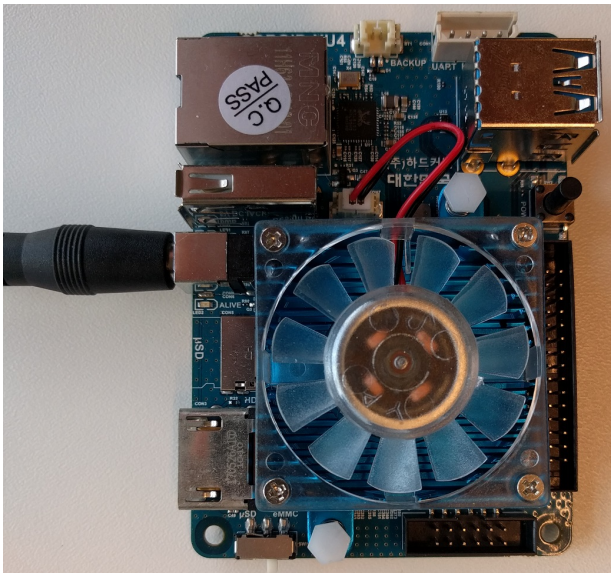


Figure 4.3: The Odroid XU4 by hardkernel.

4.3 Test Devices

In order to test the functionality of the spoofed BTS and the setup throughout the research, test devices were used. While the target of the setup is IoT devices, it is convenient to have mobile phones available as test devices. The reason for this is that proper functionality, testing and debugging of the BTS is convenient with a user interface that provides feedback and allows for configuration. For example, the signal strength, available networks, connected network and network configuration can be viewed. The following phone models were used:

- 1x LG Nexus 5X.
- 2x Samsung Galaxy J3 (2016, SM-J320FN).

A mix of SIM cards, both subscription and prepaid were used to cover the available 2G networks in the Netherlands: KPN, Vodafone and T-mobile. Besides these, Tele2 SIM cards were used in order to test the spoofing of a network that normally is not available.

Eventually, the goal is to investigate IoT devices. A small variety of devices have been selected (see Figure 4.4) to test the setup:

- Sonoff G1 GSM/GPRS switch [49]. A relay that communicates via GPRS and can be operated remotely using an Android app. The user has to provide a SIM card.
- Tractive GPS tracker [55]. A device used to track pets that communicates data over GPRS. The tracker contains an (embedded) SIM card.
- King Pigeon S271 GSM/GPRS Remote Terminal Unit (RTU) [18]. An RTU is an electronic device that interfaces with the physical world using sensors and actuators. It is typically used in SCADA systems. The user has to provide a SIM card.

(a) Sonoff G1 GPRS switch.

(b) Tractive GPS tracker.

(c) King Pigeon S271 GSM RTU

Figure 4.4: The IoT test devices that were used.

4.4 Setup Configuration

Now that the different components that are required to operate a 2G network have been selected, the setup is constructed and configured.

First of all, the operating system for the Odroid XU4 is installed. There are Linux OS images available by Odroid [33]. The Ubuntu MATE 16.04.3 (20171212) with kernel version 4.14 was selected (the latest release at the time) and installed on a 16 GB eMMC module. See Appendix B.1 for the details regarding the flashing and installation.

Secondly, the bladeRF software is installed [29] that is required to interact with the device. See Appendix B.2 for the installation. Using the included bladeRF-cli utility, we can probe for connected bladeRF devices, flash firmware, load FPGA images and adjust device parameters (see Listing 4.1).

```
odroid@odroid:~$ bladeRF-cli -p

Backend:        libusb
Serial:         ********************************
USB Bus:        4
USB Address:    3

odroid@odroid:~$ bladeRF-cli -i
bladeRF> info

  Serial #:                 ********************************
  VCTCXO DAC calibration:   0x90ce
  FPGA size:                40 KLE
  FPGA loaded:              no
  USB bus:                  4
  USB address:              3
  USB speed:                SuperSpeed
  Backend:                  libusb
  Instance:                 0

bladeRF> version
```

```
  bladeRF-cli version:        1.5.1-git-3095d995
  libbladeRF version:         1.9.0-git-3095d995

  Firmware version:           1.8.0
  FPGA version:               Unknown (FPGA not loaded)
```

Listing 4.1: Example usage of bladeRF-cli utility.

Thirdly, the functionality of the bladeRF is tested. The open source SDR receiver software Gqrx is used. See Appendix B.3 for the installation process. This tool can visualize part of the radio spectrum as a waterfall, see Figure 4.5. By default, no FPGA image is loaded when the bladeRF is connected. This can also be viewed in Listing 4.1. In order to receive or transmit with the device, such an image must be loaded. These FPGA images can be found here [25]. The image can be loaded with the bladeRF-cli utility as shown in Listing 4.2. YateBTS comes with the correct firmware image that is loaded by the software, so the steps mentioned here are only required when the bladeRF is used for other purposes (like using Gqrx).

```
odroid@odroid:~$ bladeRF-cli -l ~/Downloads/hostedx40-latest.rbf
Loading fpga...
Done.
odroid@odroid:~$ bladeRF-cli -i
bladeRF> version

  bladeRF-cli version:        1.5.1-git-3095d995
  libbladeRF version:         1.9.0-git-3095d995

  Firmware version:           1.8.0
  FPGA version:               0.7.1
```

Listing 4.2: Loading the FPGA image using the bladeRF-cli utility.

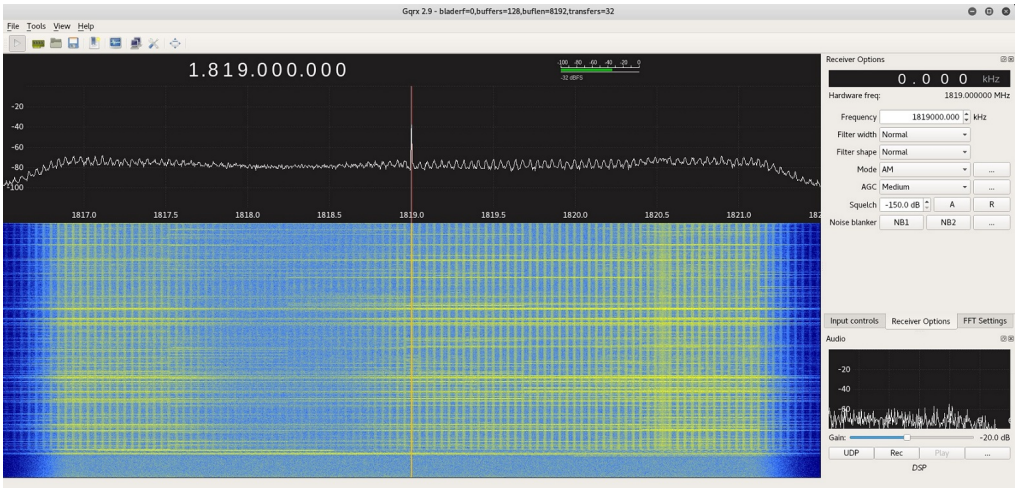


Figure 4.5: A waterfall view of the radio spectrum around 1819 MHz.

Fourthly, the YateBTS software is installed, see Appendix B.4. Before it can be used, the software will have to be configured. This can be done by changing

the configuration files in `/usr/local/etc/yate/` or by using the provided web interface. See Listing 4.3 to setup the web interface such that it can be accessed at `http://localhost/nipc/`. Note that this requires YateBTS to be installed.

```
cd /var/www/html
ln -s /usr/local/share/yate/nib_web nib
chmod -R a+w /usr/local/etc/yate
service apache2 start
```

Listing 4.3: Setting up the YateBTS Web GUI for NIPC management.

The YateBTS software restricts the registration of a mobile station by filtering on the transmitted IMSI. The subscribers can be configured individually (Figure 4.6) or a regular expression can be configured to accept subscribers. For example, `^204` will accept all subscribers from the Netherlands. Since a network will be operated that should accept arbitrary devices, all subscribers should be accepted. This can be achieved using the regular expression `.*`, see Figure 4.7.

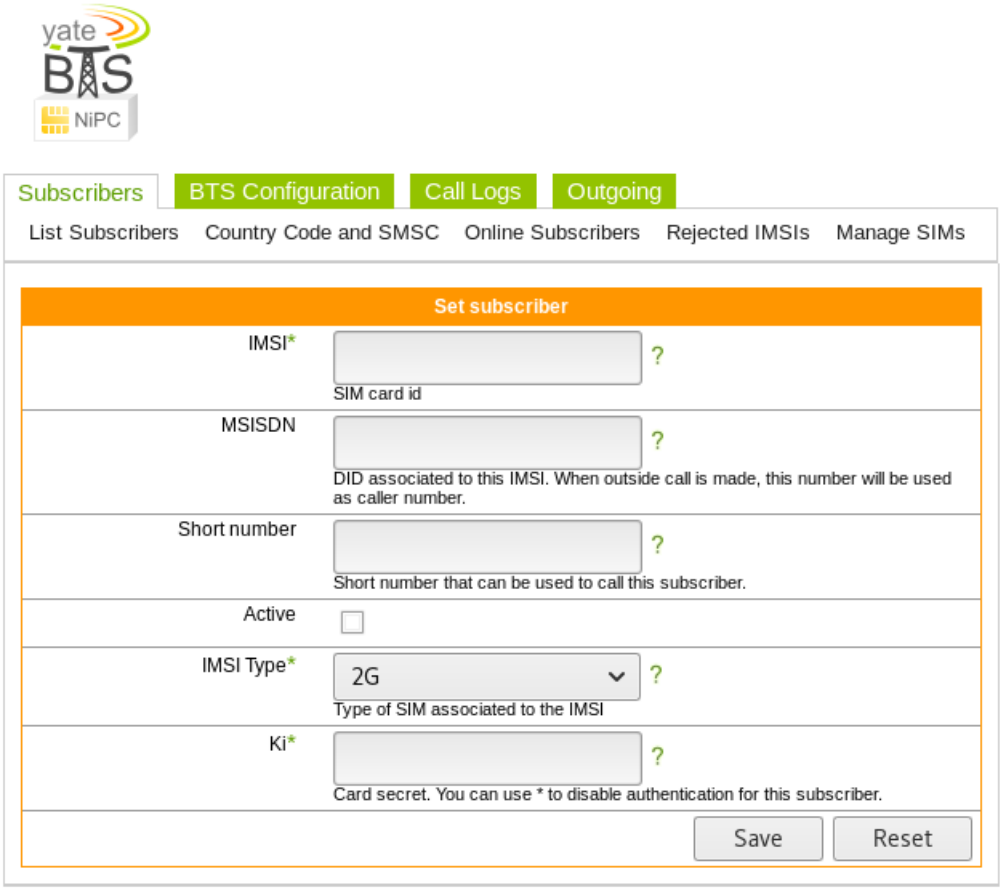


Figure 4.6: Configuration of an individual subscriber that is allowed to connect to YateBTS.

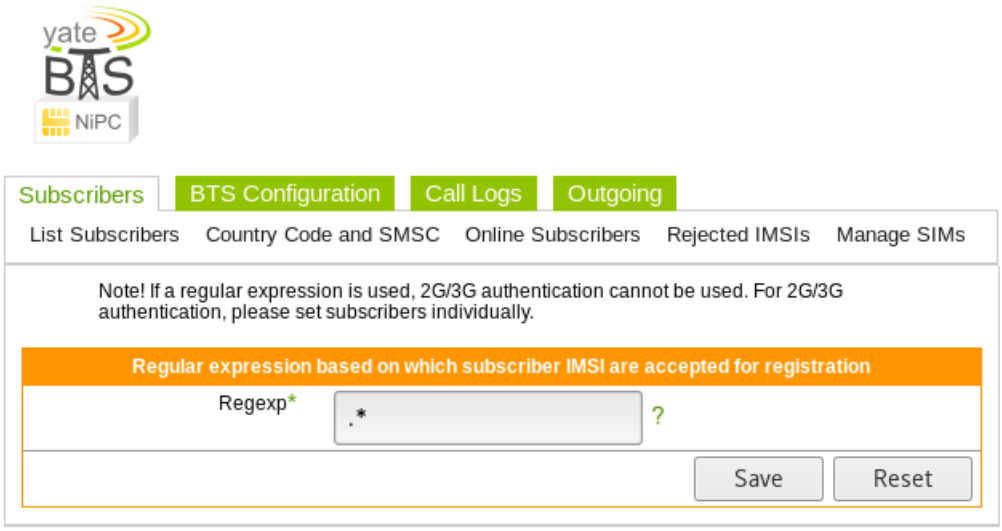


Figure 4.7: A regular expression that accepts all subscribers that connect to YateBTS.

Now that the allowed subscribers have been configured, the basic GSM network parameters are discussed (Figure 4.8):

- `Radio.Band`, the radio band on which the software will operate. GSM-850, E-GSM-900, DCS-1800 or PCS-1900 can be selected.
- `Radio.C0`, the Absolute Radio Frequency Channel Number (ARFCN) is a unique number for the channel of a GSM radio band. It actually defines a pair of channels: one channel for uplink and one for downlink. The calculation for the ARFCN can be viewed in Table 4.2.
- `Identity.MCC`, the Mobile Country Code is a three digit value that represents the country in which the PLMN (network) is located [2].
- `Identity.MNC`, the Mobile Network Code is a two or three digit value that identifies the PLMN within that country [2].
- `Identity.LAC`, the Location Area Code is a two byte value that identifies a location area within that PLMN [2].
- `Identity.CI`, the Cell ID is a two byte value that identifies the BTS or a cell/sector within the LAC [2].
- `Identity.BSIC.BCC`, the Base Station Identity Code contains the three bit Base station Color Code which is used to distinguish between BTSs that have overlapping coverage (this value must be different in that case) [2].
- `Identity.BSIC.NCC`, the BSIC Network Color Code is a three bit value used to distinguish between adjacent PLMNs which may use the same frequencies in neighboring areas [2].
- `Identity.ShortName`, this is used to display the network name on some phones/devices when the display space is limited. It was used in previous/-commercial versions of YateBTS and is now hard-coded into the software. Changing this parameter no longer has an effect, the source code will have to be adjusted instead.

- `Radio.PowerManager.MaxAttenDB`, the maximum attenuation of the transmitter in dB with respect to the full output power. With x representing this setting, the allowed range is $0 \leq x \leq 80$ with 0 representing no attenuation or maximum power and 80 represents full attenuation or minimum power. So, this sets the minimum allowed output power, e.g. 35 dB means -35 dB with respect to full output power.
- `Radio.PowerManager.MinAttenDB`, the minimum attenuation of the transmitter in dB with respect to the full output power. With y representing this setting, the allowed range is also $0 \leq y \leq 80$. This sets the maximum output power, e.g. 10 dB means -10 dB with respect to full output power. Do note that $y \leq x$. In short, the power parameters together determine the allowed range for output power p: $y \leq p \leq x$.

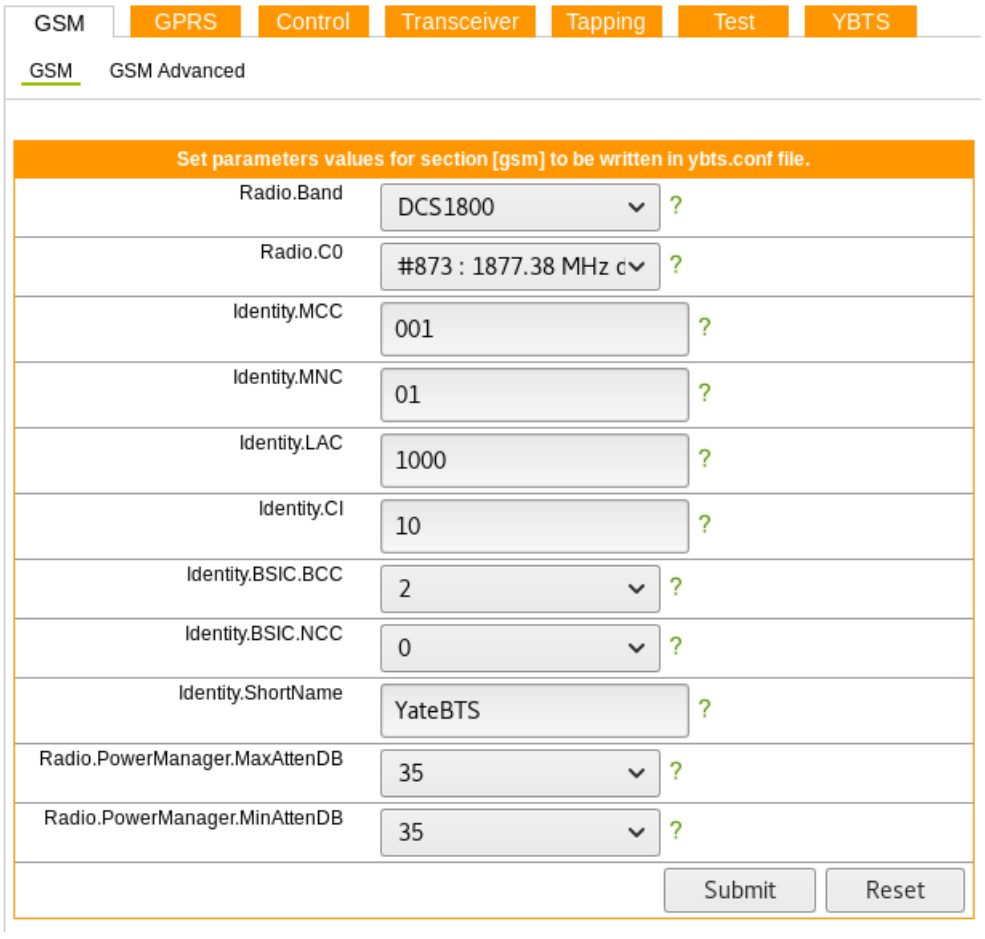


Figure 4.8: The basic GSM parameters for YateBTS.

4.4.1 Test Network Configuration

Throughout the research, it is useful to have a test network available that operates in 'open air' and allows us to test various tools and configurations with

Band	ARFCN	Uplink frequency (MHz)	Downlink frequency (MHz)
GSM-850	$128 \leq n \leq 251$	$f_{up}(n) = 824.2 + 0.2 \cdot (n - 128)$	$f_{dw} = f_{up}(n) + 45$
E-GSM-900	$0 \leq n \leq 124$ $975 \leq n \leq 1023$	$f_{up}(n) = 890 + 0.2 \cdot n$ $f_{up}(n) = 890 + 0.2 \cdot (n - 1024)$	$f_{dw} = f_{up}(n) + 45$ $f_{dw} = f_{up}(n) + 45$
DCS-1800	$512 \leq n \leq 885$	$f_{up}(n) = 1710.2 + 0.2 \cdot (n - 512)$	$f_{dw} = f_{up}(n) + 95$
PCS-1900	$512 \leq n \leq 810$	$f_{up}(n) = 1850.2 + 0.2 \cdot (n - 512)$	$f_{dw} = f_{up}(n) + 80$

Table 4.2: Formulas to calculate the ARFCN.

a mobile phone as well as functionality of the network. When configuring the GSM parameters for a test network, we have to take into account that YateBTS will be transmitting on a certain part of the radio spectrum and might interfere with other communication. Depending on local law, transmission on parts of the radio spectrum without a license is illegal. In the Netherlands, it is allowed to operate a BTS on the 1780-1785 MHz and 1875-1880 MHz radio bands without a license [36]. These frequencies are also known as the DECT guard band. This requires us to set the `Radio.Band` and `Radio.C0` parameters such that YateBTS operates within these specifications. `Radio.Band` will have to be set to DCS-1800. Using the formulas in Table 4.2, the range of allowed values $n_{lo} \leq n \leq n_{hi}$ for ARFCN n is calculated:

$$n_{lo} = \frac{(1780 - 1710.2)}{0.2} + 512 = 861, n_{hi} = \frac{(1784.8 - 1710.2)}{0.2} + 512 = 885$$

So, the `Radio.C0` can be set between 861 and 885.

`Identity.MCC` and `Identity.MNC` are configured for a test network, such that mobile stations that are close will not try to connect to our PLMN. Such a test network is also known as Test PLMN 1-1 and has MCC set to 001 and MNC to 01.

A small network that reaches only a couple of meters is desired, therefore the power parameters are adjusted to reduce the range of the network. Tests are performed to determine the desired value, this can be read in Section 5.1.1.

The other parameters are left at default, because they do not influence other networks. `Identity.LAC` and `Identity.CI` are used to distinguish between areas within our network, but there is only one area (our BTS), so any value can be picked. `Identity.BSIC` is only relevant to distinguish between PLMNs with overlapping frequencies. Fortunately, the configured frequency band is open for free use and not assigned to mobile operators [50]. Therefore, we do not expect to have other networks that overlap in frequency (this is also what we want to avoid).

4.4.2 Spoofed Network Configuration

In order to spoof a network by a wireless service provider, the `Identity.MCC` and `Identity.MNC` will have to be configured accordingly. The provider network identifiers can be viewed here [15]. However, this does not mean that every provider listed here also operates a network. In the Netherlands, there are only three providers that have a large GSM network with thousands of BTSs: KPN, T-Mobile and Vodafone. The provider Tele2 does not have their own 2G network, but roams on the network of one of the other providers. The rest of the listed providers are either inactive or are considered a 'virtual' provider, meaning that they resell the resources of an existing provider. To get an overview of the cell towers in the Netherlands, we can take a look at a database that contains information about cell towers around the world [57] and count the entries for the Netherlands, see Listing 4.4.

```
DATABASE FORMAT:
radio,mcc,net,area,cell,unit,lon,lat,range,samples,changeable,created,updated
UMTS,262,2,801,86355,0,13.285512,52.522202,1000,7,1,1282569574,1300155341
GSM,262,2,801,1795,0,13.276907,52.525714,5716,9,1,1282569574,1300155341
GSM,262,2,801,1794,0,13.285064,52.524,6280,13,1,1282569574,1300796207
...

CELL TOWER COUNT:
root@kali:~/cellid-db# grep GSM,204 cell_towers.csv | awk -F , '{print $3}' | \
sort -g | uniq -c
     17 2
     12 3
  45745 4
      2 7
  39877 8
      5 9
     26 12
  41081 16
      4 18
      5 19
     71 20
      2 21
      9 25
     17 67
      2 85
```

Listing 4.4: Obtaining a count of GSM cell towers for each provider in the Netherlands.

Indeed, this confirms that Vodafone (04), KPN (08) and T-Mobile (16) have large networks, each have about 40.000 entries of GSM cell towers in the database.

The database can also be used to obtain a rough location estimate when the MCC, MNC, LAC and CID are provided. This can be done, because the database contains the coordinates (longitude, latitude) of each cell tower. This process is also referred to as geolocation. Conversely, coordinates can be used to obtain nearby cell towers (also known as reverse geolocation). This can be useful when it is desired to not only spoof the network, but also a certain location within that network. A script has been written to do exactly this, see Appendix B.5.

See Listing 4.5 for an example output.

```
root@kali:~/cellid-db# ./find_towers.py
incorrect amount of arguments:
usage: python script.py [csv-database] [address] [radius in km]

root@kali:~/cellid-db# ./find_towers.py cell_towers_nl.csv Eindhoven 1
       Distance    MCC    MNC    LAC    CellID      lon      lat
---  ----------  -----  -----  -----  --------  -------  -------
  0   0.0229928    204      4    211     11882  5.46939  51.4416
  1   0.0261543    204     16   1005     39637  5.46959  51.4414
  2   0.0715911    204      4     35     11883  5.46906  51.4411
  3   0.0764571    204      8   4424     45496  5.47007  51.4423
  4   0.109267     204     16   1005      7848  5.46854  51.441
  5   0.13918      204     16   2001     30923  5.4705   51.4428
  6   0.13918      204     16   2001     31062  5.4705   51.4428
  7   0.162845     204      8   3390     46048  5.4716   51.4425
  8   0.17572      204      4    211     50911  5.46845  51.4403
  9   0.176472     204      8   3390     45495  5.47223  51.4419
 10   0.19579      204      8   4424     45495  5.47247  51.4412
...
```

Listing 4.5: Output of cell towers in a one kilometer radius around address "Eindhoven".

Now we have enough information to configure `Identity.MCC`, `Identity.MNC`, `Identity.LAC` and `Identity.CI` for the networks that we intend to spoof: Vodafone, KPN, T-Mobile and Tele2. Since YateBTS will impersonate an existing network now, arbitrary devices could try to connect to it. The transmitted signals should be contained to prevent this from happening. Additionally, as will be discussed in Section 4.5, other network signals should not reach the device that we intend to connect to the spoofed network. Once the signals transmitted by YateBTS are contained, then this allows us the freedom to set `Radio.Band` and `Radio.C0` to other values besides the previously mentioned DECT guard band. For example, the frequencies allocated to the providers as listed in [50] can be used, including the use of the GSM-900 band.

In Section 4.4.1, it was mentioned that BSIC is only relevant to distinguish between networks with overlapping frequencies. Also in the spoofed network configuration environment, a MS that we intend to connect should only receive the signals from the spoofed network, because the transmitted signals will be contained. As is mentioned in [45], changing the BSIC value did not have an effect on the spoofing of networks. Therefore, this value is left at default.

Finally, the power setting will be adjusted based on the size of the lab setting that the network will operate in. This is done in order to prevent reflection from the (small) environment and to prevent the leakage of the signal to the outside.

4.5 Extension - Ensuring Device Connection

Once the YateBTS software is started and a spoofed network is operational, the goal is to connect a mobile station. The core reason of why a MS would connect in the first place, is that there is no mutual authentication in 2G [3]. In fact, there

is one-way authentication: the subscriber is authenticated by the network. With YateBTS, even this one-way authentication can be disabled by not providing the SIM secret k_i (Figure 4.6 and 4.7). In any case, a mobile station will trust the network that is associated with the SIM (or other networks when roaming is enabled). This means that a MS should connect to the spoofed network when it impersonates the network associated with the SIM. Unfortunately, a mobile station tends to prefer higher generation networks, simply because they are faster. This in turn means that an EDGE network (2.75 G) will be preferred over a GPRS network (2.5 G), as was also confirmed by [45]. In order to force a MS to use the spoofed GPRS network, it is necessary to block the signals of higher generation networks. This can be achieved by for example: frequency jamming, modifying the device software, modifying the device antenna or placing the device in a Faraday cage. Since the goal here is to both prevent signals transmitted by YateBTS from going outside as well as preventing external signals from going inside, a Faraday cage was used.

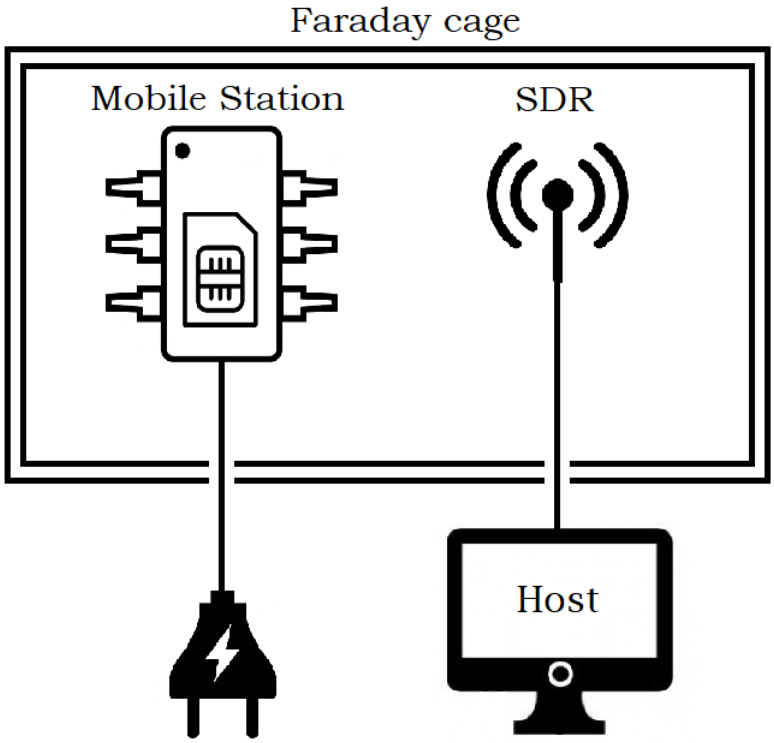


Figure 4.9: A simplified model of the setup with a Faraday cage.

4.5.1 Faraday Cage

A Faraday cage is an enclosure of a conducting material which blocks static electric fields. Electric charges within the material get redistributed when it is affected by such a field. This creates an opposing electrical field within the cage which cancels out the external field. Similary, this also works for static electrical

fields produced within the cage.

Electromagnetic (EM) radiation, however, consists of sinusoidal EM fields. Radiation is not completely blocked, but heavily attenuated based on (among others) the material used, thickness of the material and properties of the wave [13]. Since radio waves are EM radiation, a cage might sufficiently attenuate interior signals and exterior radiation. Even when the enclosure has holes in it, the cage might still sufficiently attenuate the radiation provided that the maximum diameter of the holes is 'significantly small'. No exact definition of this 'significantly small' could be found that also takes all the different variables into account (e.g. transmission power, frequency, material, thickness of material, distance to the receiver). Generally speaking, the rule of thumb for the 'significantly small' is that the holes in cage can be a maximum of 1/10th of the wavelength. This rule is used to find out whether it is feasible to use a Faraday cage in the context of the setup.

The goal is to block higher generation cellular technologies. This means that the Faraday cage should be able to block up to the 4G cellular technology. In the Netherlands, the highest frequency used in 2G, 3G and 4G is 2.6 GHz [50]. The wavelength of this frequency is calculated:

$$\lambda = \frac{v}{f} = \frac{3 \cdot 10^8}{2.6 \cdot 10^9} = \frac{0.3}{2.6} = 0.11 \text{ m}$$

Using the previously mentioned rule of thumb, holes of about 1 cm are allowed in the cage. Therefore, it might be feasible to construct such a cage which even allows cables to leave the enclosure. Because this is merely a guideline, experiments have been performed in order to find a suitable Faraday cage with the following requirements:

1. The signals transmitted by YateBTS are sufficiently attenuated such that the network can not be connected to outside the cage.
2. Signals from external cellular networks are sufficiently attenuated such that a mobile station inside the cage can not connect to the networks.
3. A power and USB cable can leave the enclosure while still keeping properties 1 and 2 intact, see Figure 4.9.
4. It should be convenient to swap out the mobile station and corresponding cables in order to test a different device.
5. The Faraday cage should be reliable and reusable.
6. The Faraday cage should be mobile.

First of all, the effectiveness of aluminum foil was investigated by performing a mobile phone experiment. A mobile phone (see test devices in Section 4.3) was called while being wrapped in aluminum foil to find out whether the call could still be received. It was also called without being wrapped for control. The results of this were that the foil could indeed be used to attenuate the signals sufficiently such that a call (and even the 5 GHz Wi-Fi signal) was no longer received. However, the foil had to be either wrapped really precisely (with no holes) or a couple of layers were needed.

As a follow-up experiment, a carton box was covered in two layers of aluminum foil in order to construct a Faraday cage, see Figure 4.10a. Again a similar test was performed where a mobile phone was called, both outside (for control) and inside the enclosure. The results are that the mobile phone was occasionally still able to receive the call: the enclosure was unreliable. This issue was most likely caused by the lid on the box not providing a tight seal. Additionally, the foil was rather fragile, which reduces the re-usability of the box. Once a hole for a USB cable was made in the box, the attenuation was even worse. Since the cable head is rather large, part of the hole needed to be filled up again. This let to the conclusion that aluminum foil proved to be unreliable.

Next, various metal canisters were tested in the same way. These worked perfectly, given that they had a lid with a tight seal. However, making holes for cables in the canisters (and filling them back up) was cumbersome. This also makes it inconvenient to switch devices and their cables.

After the previous experiences, it seemed like constructing a reliable Faraday cage that is also large enough to fit both the MS and SDR by hand is not trivial. Instead, professional Faraday cages were considered. Preferably, an RF test enclosure such as the one depicted in Figure 4.10c is used. Unfortunately, these are expensive (between $1000 and $2000). A cheaper option was considered: a Laptop Faraday bag [9] (Figure 4.10b). The bag advertised to block even WiFi signals (including 5 GHz) and is sealed with velcro. This makes it convenient to tightly wrap around cables leaving the enclosure. After performing the same mobile phone experiment, it was found to be reliable. When operating YateBTS for the experiments in Chapter 5, a USB cable could simply be run out of the enclosure. Unfortunately, as was found by connecting one of the test devices later on, the power cable to the device provided a really good reception to the device (it performed as antenna). This issue was solved in the next section.

(a) Aluminum foil prototype Faraday cage.

(b) A Faraday laptop bag by Disklabs.

(c) STE3000 by Ramsey Electronics [43].

Figure 4.10: Various Faraday cages.

4.5.2 Power Cable Leakage

A power cable leaving the enclosure might pose as an antenna for a (poor) insulated device. It can provide a signal to a device, allowing it to sufficiently receive an outside network and connect to it. This was found to be the case with the Sonar G1 smart switch. In order to solve this problem, there are several solutions. The power supply and the rest of the equipment can be put inside the container. Unfortunately, this is not a very ideal solution given that the space in the Faraday bag is rather limited and it might generate enough heat to be fire hazard. Instead, there are ways to prevent the signal from being carried through the wire to the device. One of such methods is galvanic isolation. In this method, the electrical circuits are separated to get rid of stray currents, possibly also the outside RF signals from the 2G networks. Since we are dealing with AC currents, an isolation transformer would have to be used. Before investigating this, a cheaper method was investigated first.

Ferrite cores can be used to provide enough impedance in the wires such that high frequency noise is suppressed. These cores come in various shapes and material mixes, which causes a different impedance on the wires at various frequency ranges. Many of these cores can be compared here [58]. Fortunately, these cores are rather cheap, costing only a couple of dollars. Therefore, a variety of these cores were purchased in order to test whether they would suppress the noise in the wire sufficiently. There was not enough time to properly test and select the most appropriate cores. The WE-TOF 74270191[1] and WE-AFB 74270096[2] were found to sufficiently suppress the noise. With these two cores in series around the cable (Figure 4.11), the device would no longer connect to external networks when placed in the Faraday bag. What was particular was that the amount of windings around the cores mattered. The impedance was found to decrease after too many windings, while two or three windings seemed to still block the signal. Any more than that and the device would still detect the signal of external networks.

4.5.3 GSM Parameters

When multiple 2G networks are available, the situation arises where the spoofed network 'competes' with an official network for connectivity of a mobile station. There are several GSM network parameters that can increase the likelihood of a MS choosing (or sticking to) the spoofed network over an official network. Those parameters were studied and are presented below. Do note that in this context it is assumed that the MS is competing between GPRS networks, implying that there is no reliable reception of higher generation networks (e.g. those are overloaded or unavailable).

A BTS constantly transmits information about the identity of the network. This broadcast is referred to as the Broadcast Control Channel (BCCH). Besides containing the identity of the network (i.e. the MCC and MNC), the

[1] http://katalog.we-online.de/en/pbs/WE-TOF
[2] http://katalog.we-online.de/en/pbs/WE-AFB

Figure 4.11: The WE-TOF 74270191 and WE-AFB 74270096 ferrite cores wrapped around the power supply cable of the Sonoff G1 GPRS switch.

broadcast also contains for example the ARFCN, CID, LAC and a list of neighboring cells. The parameters that are transmitted in the BCCH can be viewed in Chapter 9.1.31 onwards (System information Types) of the GSM 04.08 specification [4].

Connection Efficacy

The first relevant parameter is the transmission power of the BTS. By providing a better reception of the signal, the device might choose the spoofed network over the official network. While an official BTS has a much more powerful signal than can be achieved with an SDR, it also has to cover a larger area. A device can receive better signal quality from a spoofed BTS that has lower transmit power, simply because it is closer to the target.

A mobile station may prefer communication frequencies as advertised by the official network it was connected to. In order to increase the probability for the device to connect to the spoofed network, similar frequencies should be set for the spoofed network rather than some off-band frequency. This requires the ARFCN and band to be set accordingly.

Cell Entrapment

Once a connection with the spoofed network has been made, it is desired to keep the mobile station connected. By not configuring any neighboring cells, the network will try to prevent the MS from transitioning back to an official cell. This is because the network does not advertise any BTSs that are near and offer better signal quality. For that reason, the device might stick to the spoofed cell

until its signal level is below a certain threshold to still function correctly.

Similarly, configuration of a high cell re-selection offset might ensure that the mobile station stays on the spoofed cell. This offset is used together with the path loss exponent to determine the cell re-selection criterion, i.e. the value that is used to select the cell with the highest probability of successful communication. By configuring such a high cell re-selection offset, the signal quality of neighboring cells has to be exceptional in order for the device to transition.

Detection of a Spoofed BTS

To detect spoofed BTSs, one could check for out of place configurations of the aforementioned parameters as well as several others [17, 47]. Examples are:

- Unexpected broadcasting frequencies (ARFCN).
- Unusual LAC and/or Cell ID.
- Overlapping ARFCN with other BTSs in proximity, but different LAC and/or Cell ID.
- Same LAC and/or Cell ID as other BTSs in proximity but different ARFCN.
- Absence of supported features like EDGE.
- Absence of encryption.
- Lack of neighboring cell information (empty neighbor list).
- High cell re-selection offset.
- Low location update timer (T3212). This would decrease the time interval between location updates in order to gain more information about the device.

With respect to YateBTS, the short name of network (`Identity.ShortName`) is hard coded into the software. Furthermore, a welcome SMS is sent to the device once it registers to confirm that YateBTS is used. These are two signs specific to the software that can be used to detect a spoofed network. However, the `Identity.ShortName` can be changed in the source code after which YateBTS can be recompiled. The welcome SMS that is sent can be changed and/or removed by editing a JavaScript file.

4.6 Extension - Brute-forcing

The setup will be used on arbitrary IoT devices that are known to use GPRS. It might not be advertised what network is used. While some devices roam and are able to connect to multiple networks, there will also be devices that limit connectivity to one network (or a small group of networks). In order to find out which network is used by the device, a variety of networks can be spoofed until a correct one is found and a connection is made. It can be quite tedious for such an action to perform manually. When this process of brute-forcing the networks is automated, we can not assume the device is continuously looking for a network. Typically, a device increases the time between network searches when allowed networks can not be found. This in turn would mean that a single

network would have to be spoofed for several minutes before a device intends to connect. In the worst case, it might stop looking for a network and go to sleep mode. To avoid such headaches, we assume instead that a device looks for a network after booting. Therefore, a device can be power cycled to renew the frequency sweep. A time limit can be picked in which a device should connect, if it does not, the device is power cycled and the next network is spoofed. In this way, the networks will be brute-forced spoofed one by one until the mobile station connects or the list of networks is exhausted.

To power cycle a device automatically, some additional hardware is required. A relay switch was acquired and will be operated by our host device. The power supply of a test device can be connected to this relay. On the other hand, a power strip can be connected if the test device uses a 230 V AC power supply. In this way, also multiple devices can be connected and supported should that be necessary. Since the Odroid XU4 has GPIO ports, they can be used to power and control a Sunfounder 2-channel relay module [54]. The XU4 shifter shield [30] is acquired in order to raise current of the GPIO signal pins to 3.3 V, see Figure 4.12. For other hosts, a USB [10] powered relay may be used.

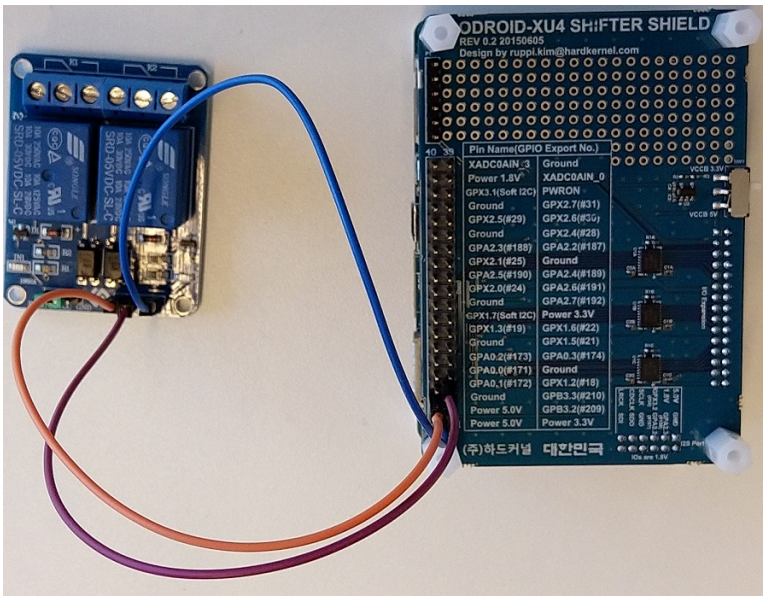


Figure 4.12: The Sunfounder relay module connected to the Odroid XU4 via the Odroid XU4 Shifter Shield.

Chapter 5

BTS Experiments

The setup and the configuration of both the test network and spoofed networks have been established. In this chapter it will be used to perform experiments with the test devices mentioned in Section 4.3.

The first section will contain experiments on the test network. This network will be used to determine a sufficient value for the transmission power, to test GPRS, SMS and call functionalities, assist in the finding of information (Chapter 6) and tool development (Chapter 7).

The second section will contain experiments concerning the impersonation of official networks.

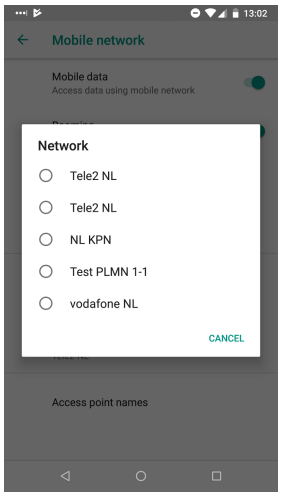
5.1 Operating the Test Network

After YateBTS has been configured for a test network (Section 4.4.1), the software can be started using the command `yate`. Once the software is initialized, a mobile phone can be used to search for the network. This is done by manually selecting the network after performing a search, see Figure 5.1a. When the connection to the network is successful (Figure 5.1c), the test network will be displayed as `00101`, `712 712` or `Test PLMN 1-1`, varying between devices. The `712 712` appears to be the `Identity.ShortName` mentioned in Section 4.4, which is hard-coded into the YateBTS software. When the time setting of the device is configured to use network-provided values, the time of the device might change.

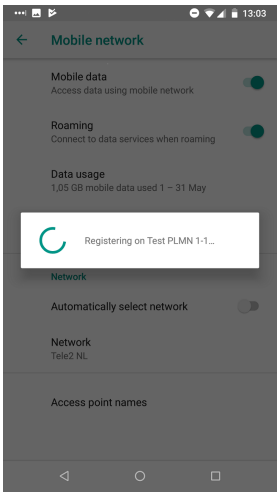
After successful registration, the device will be assigned a local phone number. An SMS is sent to the device to notify it successfully connected to YateBTS, see Figure 5.2a. Registration of the devices can also be found by sending commands to the YateBTS rmanager interface (Listing 5.1). The software offers an SMS chat bot (Figure 5.2b) and interactive voice response to test the connection.

GPRS functionality can be enabled in the YateBTS configuration file. It makes use of the existing internet connection on the host device. The DNS servers of the host device are used, but these can be changed in the configuration of the software. GPRS traffic will be output on a new interface: `sgsntun`. However, this new interface is not connected to the internet. Network Address

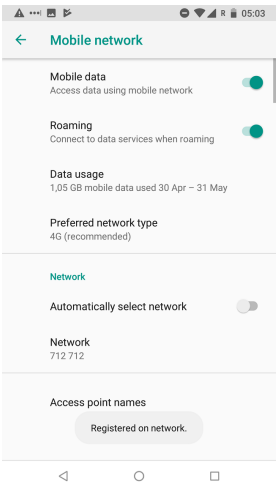
Translation (NAT) will have to be used in order to route traffic on `sgsntun` to the internet. Since IP addresses are dynamically assigned to the devices, IP masquerading is applied (Listing 5.2). The pool of IP addresses that can be assigned is also defined in the YateBTS configuration file.



(a) Networks found by the scan.



(b) Connecting to the test network.



(c) Connection successful. The network is displayed as 712 712. Notice the time changed.

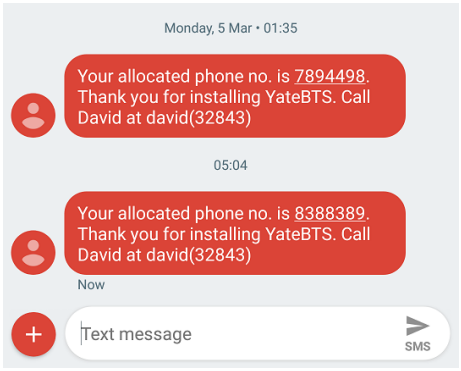
Figure 5.1: Figures depicting the connection to the test network on the LG Nexus 5X.

```
root@kali:~/yatebts# telnet localhost 5038
...
nipc list registered
IMSI            MSISDN
--------------- ---------------
20408**********   *******
20404**********   *******
mbts gprs list ms
MS#1,TLLI=c00a9001,800b7001 rrmode=PacketTransfer Bytes:242419up/545900down
    Utilization=127%
GMM Context: imsi=20408********** ptmsi=0xa9001 tlli=0xc00a9001 state=
    GmmRegisteredNormal age=252 idle=0 IPs=192.168.99.1
...
```

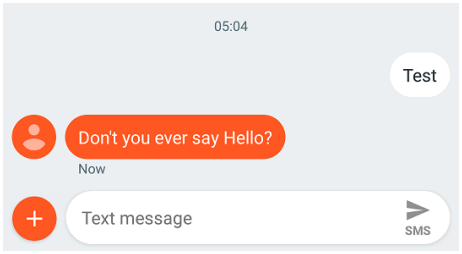
Listing 5.1: Querying about connected devices using the Yate rmanager interface.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
sudo iptables -A POSTROUTING -t nat -s 192.168.99.0/24 ! -d 192.168.99.0/24 -j
    MASQUERADE
```

Listing 5.2: Commands to enable forwarding of GPRS uplink and downlink traffic to and from the internet.



(a) SMS message that is sent upon registration of the device.



(b) YateBTS chat bot that can be used to test connection.

Figure 5.2: YateBTS registration and chat bot.

After these steps, a local 2G network with data services support has been constructed. It support a small amount of devices. The test phones were connected in order to check the functionality of the network. This concerned the SMS, phone calls and GPRS functionality. By making use of the allocated phone numbers, the phones are able to communicate with each other by sending SMS messages and making phone calls. However, the GPRS functionality has two additional requirements. First of all, the roaming option must be enabled on the phone in order to connect to a network that is not associated with the inserted SIM card (such as the test network). Secondly, an Access Point Name (APN) must be configured. The device will not request an IP address without one [60]. What is configured as APN does not actually matter, the value is not used by the software. The APN defines a GGSN (see Section 2.4) and optionally an operator identifier to define in which PLMN the GGSN is located. The GGSN is not accessible from outside an (official) operator network, which is why it is not used by the software.

Even after configuration of these two additional requirements, GPRS functionality might take time to establish (30 seconds or a minute). Sometimes the data services will have to be toggled on and off for the functionality to become available.

5.1.1 Transmission Power

Now that the test network base station is operational, the transmission power parameters (Section 4.4) will be tuned in order to reduce the range of the network. Inside a Faraday cage, the signal will not have to travel far. By configuring a lower transmission power, the effect of reflection in the Faraday cage during the spoofing of a network will be reduced. A powerful network will suffer from more reflection which might decrease the signal quality. With a small range network, the attenuation of the Faraday cage will be sufficient to stop the signal from interfering with the outside.

In order to measure the range of the signal, a test phone was connected

and the signal quality indicator was used in order to give a (rough) estimate of the range. Do note that the radiation pattern of the transmitting antenna is irregular. Furthermore, the signal travels further than the indicator shows (the interference region is larger than the connected region). With attenuation disabled (i.e. 0 dbm), the signal can reach roughly 20 meters inside a building. Once set to 35 dbm, the signal reaches roughly 5 meters. This is a comfortable setting that will be used for the test network. Configuration of an attenuation of about 55 dbm results in a small network with a range of about 0.5 meter.

Finally, the test network is operated in the Faraday laptop bag with only the USB cable leaving the enclosure. The previous transmission power experiments are repeated in order to find out whether the signal can still be received outside the Faraday cage. First, the phone is connected to the test network, after which the SDR is placed in the enclosure while the device will remain outside. The signal quality indicator on the mobile phone was used to measure connectivity. On the highest output power (attenuation of 0 dbm), a high signal could still be received outside the enclosure. With an attenuation of 35 dbm, the signal quality was extremely low and the device would occasionally disconnect. It would have to be placed right next to the USB cable to still receive the signal. At an attenuation of 55 dbm, the signal could no longer be received outside the enclosure.

Because the mobile phone provides a rather inaccurate measurement, a HackRF was used to look at the radio spectrum around 1877 MHz when the BTS was transmitting the test network on various output powers. With the noise floor at -76 dbm at the receiver, a signal with transmission attenuation of 35 dbm was noticeable at -60 dbm with full gain enabled on the receiver. With a transmission attenuation of 50 dbm, the signal could not be observed right next to the bag. It could, however, be observed right next to the cable. This was also the case at 55 dbm, the signal could be observed at -73 dbm right next to the cable.

5.2 Network Spoofing

The configuration(s) as defined in Section 4.4.2 will be used to spoof the official networks of KPN, Vodafone, T-Mobile and Tele2. Their corresponding frequencies [50] as well as nearby CID and LAC were used. A transmission power attenuation of 55 dbm was configured.

Connection to these networks will be tested using the corresponding SIM cards in combination with the test phones. The network spoofing was performed with mobile phones first in order to allow for a degree of configuration and more control of the MS as well as some feedback on the device before transitioning to embedded devices. A connection to a spoofed network is considered successful when an IP address is assigned to the mobile station and GPRS services are available. This was checked by using the commands in Listing 5.1. During the experiments mentioned in this section, the mobile stations and network will be contained in a Faraday cage.

5.2.1 Mobile Phone Experiments

In the first set of experiments, the mobile phones were configured to only use 2G networks, but still select one automatically. Roaming was disabled. This causes the devices to select the preferred network of the SIM, e.g. the KPN 2G network will be selected by the device when a KPN SIM is inserted.

The experiments were performed in the following way. First the bladeRF was placed in the Faraday cage. Next the mobile phone was configured accordingly, then also placed in the cage. Only then, when the cage was properly sealed, the software was started.

The spoofing of the network was successful for all tested SIM cards: KPN, Vodafone, T-Mobile and Tele2. However, a similar particularity as for the test network was observed: GPRS functionality might take some time to establish. Since no official Tele2 network existed, it was tested whether the devices would connect to both the T-Mobile and Tele2 network. This turned out to be successful: apparently (newer) Tele2 SIM cards allow for connection with the T-Mobile network without roaming enabled.

In a follow-up set of experiment, the devices were configured to prefer the highest available cellular technology, i.e. 4G over 3G over 2G. A similar set of experiments were performed in order to test whether a downgrade attack to 2G was possible by making the spoofed network the only available signal. This turned out to be the case: for all test phones a downgrade attack was successful, causing the devices to connect to the spoofed network corresponding to the SIM card.

5.2.2 Embedded Device Experiments

In the previous section it was shown that a network can be successfully spoofed using the setup. Since the setup is aimed to be used on IoT devices, the embedded test devices mentioned in Section 4.3 are used in order to put the setup to the test. With embedded devices, we have limited feedback that varies from device to device. Some devices will have LEDs to indicate some kind connectivity, while others will give no feedback. We also have limited configuration on the devices and this also varies based on the device. Because of this we assume that there is no knowledge of the configured network and on what band the device operates.

GPRS Switch

The Sonoff G1 GPRS switch (Figure 4.4a) requires the user to supply a SIM card. The PIN on the card will have to be disabled in order for the device to use it successfully. A mobile app can be used to control the switch. In order to do so, the device must first be registered with the app by means of a QR-code. When the switch is controlled using the app, an audible click can be heard every time it is switched on or off. When the device was placed in the cage and the cage was sealed, the device would still connect to the official cellular network and receive messages to switch on or off. This was caused by the power cable acting

as an antenna to the circuit. The problem was solved by the use of ferrite cores, as described in Section 4.5.2. In the end, it was possible to attenuate the signal enough such that the device would no longer connect to that network. However, this was rather unreliable, the user would have to shift the device and/or the ferrite cores around as well as double check that the cage is properly sealed. This can most likely be made more reliable by adding more attenuation to the wire (more cores). Another way to solve it would be to switch out the SIM card for one whose network does not have coverage in the current location. However, this would not be applicable to arbitrary devices whose SIM is not replaceable. Instead, the device could be taken to an area where there is no signal coverage, e.g. a tunnel.

Eventually, the device connected to the spoofed network on the GSM-900 band. It was advertised to work with the 800, 900, 1800 and 1900 GSM bands but seemed to have trouble to pick up the 1800 or 1900 MHz signal (it took considerable effort to make the device connect when operating on these bands). Registration of the device was showing up in the Yate rmanager interface using the commands in Listing 5.1. Unfortunately, the device did not seem to complete the registration process properly to obtain an IP address. Instead, it attempted to reconnect once the process failed. This resulted in many MS registrations, which would be listed as separate devices with the same IMSI in YateBTS. To investigate this issue further, the logs and radio traffic were analyzed in order to find the issue. There seems to be a problem with the GPRS attach procedure depicted in Figure 5.3. In the radio traffic of the mobile station the following messages can be observed:

1. Attach Request message with IMSI, hence no Identity Request messages are required.
2. Attach Accept message.
3. GMM Status message: Message type not compatible with the protocol state (0x62).

Currently, it looks like the device expects authentication to take place (which has been turned off) or hardwired a specific path through the protocol. Unfortunately, there was not enough time to investigate the issue further and debug the software to solve the problem.

GPS Tracker

The Tractive GPS/GPRS tracker (Figure 4.4b) is designed for location tracking of pets. It contains an embedded SIM. The device will have to be registered and a subscription will have to be purchased before the data services can be used. Fortunately, the spoofed network is able to provide data services for free. The tracker has no outgoing cables, which makes shielding the device from outside signals with the Faraday cage easier. However, this also means that the GPS signal will be blocked and no location data can be retrieved besides the use of GSM network parameters.

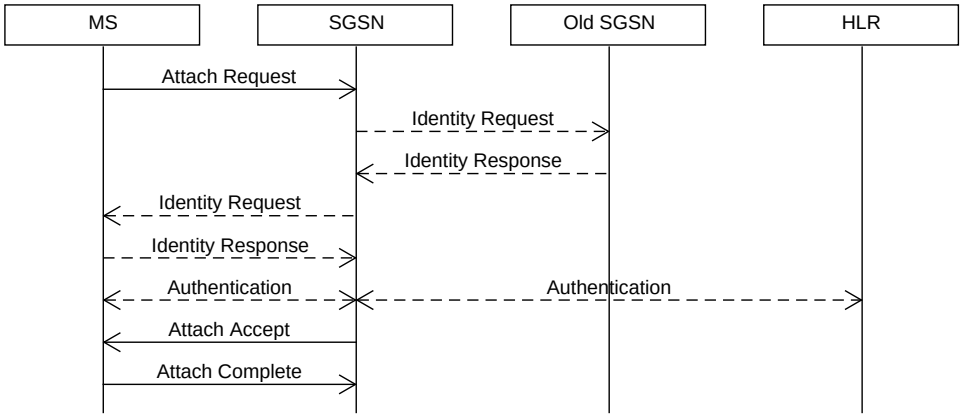


Figure 5.3: Simplified representation of GPRS attach procedure, the dashed lines are optional steps.

The KPN, Vodafone and T-Mobile networks were spoofed on the GSM-1800 band on separate occasions. The device connected to all of these networks successfully, this implies that it is allowed to roam on cellular networks operated by different providers.

GPRS RTU

The King Pigeon S271 GSM/GPRS Remote Terminal Unit (RTU) (Figure 4.4c) is a device that can be used to obtain data from sensors and control actuators, typically used in SCADA systems. The user will have to supply a SIM for the device to use the cellular network. It will also have to be configured by making use of custom software and drivers that are offered by the vendor. Once up and running, an Android app can be used to obtain information, send requests to the device and arm/disarm it. The device has an external antenna, which makes it convenient to be put into the enclosure together with the spoofed network. The device successfully connected with the network associated with the inserted SIM.

5.3 Discussion

In Chapter 4 we selected hardware and software to operate a 2G network. Furthermore, mobile phones and IoT devices were selected to test the network configurations defined in Section 4.4. Among these configurations, we distinguish between a test configuration for research experiments and spoof configurations to impersonate other networks. During network impersonation, we want to prevent the devices from connecting to a higher generation cellular network, i.e. we want to perform a downgrade attack to 2G. Additionally, the spoofing should not interfere with behavior of official networks. To achieve these two goals, we

used a Faraday cage as discussed in Section 4.5. Such a Faraday cage attenuates incoming and outgoing signals. We ran into a situation where the power cable leaked outside signals to the device. This caused the device to connect to an official network instead of one spoofed by us. In order to resolve this problem, ferrite cores were used to attenuate the signal in the power cable. To be sure we do not interfere with other signals on the radio spectrum, the transmission power of the BTS is adjusted as discussed in Section 5.1.1.

We have seen several connectivity experiments with spoofed networks. The first set of experiments concerned mobile phones, where we succeeded in impersonating four provider networks. One of these providers does not operate their own 2G network, but we were still successful in impersonating it. This might be a concern when 2G networks are eventually shutdown, because any remaining IoT devices with GPRS capability will find a spoofed network the only possible network to connect to. The feasibility of this scenario depends on the carelessness of the owners who should shut down and remove the devices. Whether it is an actual concern when an attacker can connect such a device to their own network once official networks are down, is questionable; but in the next chapter we will see certain information can be retrieved from the device which can give access to the internal network of a vendor. Lastly, we found out a downgrade attack from 4G to 2G is indeed possible with the setup.

The second set of experiments concerned IoT devices, where we managed to successfully spoof the network for all three devices. One device had difficulties during GPRS registration, but did function correctly on the official network. This registration issue is likely caused by the software implementation that is used to spoof the network (YateBTS). In order to resolve this, the software will have to be modified and recompiled or a different software package will have to be used (this might also require different hardware due to compatibility). Hence, two out of three devices connected successfully to our spoofed BTS.

Reflection on Research Questions

1. What circumstances are required to have a device connect to a spoofed cell tower? Absence of higher generation networks. This can be achieved by means of frequency jamming or a Faraday cage as discussed in Section 4.5. When only 2G networks are available, the spoofed network competes with them. Parameters that make a device prefer a certain network over others are discussed in Section 4.5.3.

2. How reliable is this connection? The reliability is directly related to the absence of higher generation networks. In our case, this would be the reliability of the Faraday cage. We did observe that the Faraday cage we selected is not a perfect solution. It is suitable in a lab setup, but in a professional environment for long penetration tests it would be advisable to purchase a professional Faraday cage such as the ones developed by Ramsey Electronics [43].

Chapter 6

Traffic, Vulnerability and Service Analysis

Once a mobile station is connected with the spoofed network, it is interesting to see what kind of information can be gathered about the device. A couple of things come to mind. First of all, the radio traffic can be analyzed. The registration of the mobile station to the BTS can contain sensitive information about the device and the services it intends to access within the cellular network. Secondly, after the MS registration is completed and an IP address is assigned, the GPRS traffic can be analyzed, intercepted and altered. Thirdly, DNS spoofing can be performed. Finally, a scan can be performed to look for available services on the device.

6.1 Radio Traffic

The first part of the information gathering on the connected device is the analysis of the radio traffic. This can be captured by enabling GSM tapping in the YateBTS configuration file. By doing so, frames from the GSM Um interface (air interface between the MS and the BTS) are wrapped into UDP packets with pseudo-header format GSMTAP[1]. In order to also view the establishment of the GPRS connection, a separate GPRS tapping option will have to be enabled. After enabling these two options, both the GSM and GPRS radio traffic will be replayed on the loop-back interface. This can then be captured by listening on UDP port 4729.

Since, communication will be going over the air interface, it could potentially be eavesdropped by anyone receiving the signal. 2G tries to achieve voice, data and sensitivity signaling confidentiality by means of encryption. The ciphering standard used for encryption of GSM calls and SMS is referred to as A5. For GPRS encryption the ciphering standard is GEA. The standards specify various different algorithms, the most commonly used variants for A5 in 2G are mentioned:

[1] http://osmocom.org/projects/baseband/wiki/GSMTAP

- A5/0: no encryption.
- A5/1: introduced in 1987, a stream cipher using the current frame number and session key K_c as input. It can supposedly be cracked in seconds using the tool kraken [22].
- A5/2: weakened A5/1 stream cipher for export regions, but has been deprecated. It can be successfully cracked in real-time [38].
- A5/3: stronger cipher, initially designed for 3G. Open design, based on block-cipher KASUMI.

For GPRS encryption, the following algorithms are the most common:

- GEA5/0: no encryption.
- GEA5/1: proprietary stream cipher using a 64-bit key and 96 bit internal state. Has been reported broken [23, 35].
- GEA5/2: proprietary stream cipher using a 64-bit key and 128-bit internal state. Has been reported broken [23, 35].
- GEA5/3: 128-bit block cipher using a 64-bit key.
- GEA5/4: 128-bit block cipher using a 128-bit key.

Do note that GSM encryption takes place between the MS and the BTS. GPRS encryption on the other hand, takes place between the MS and the SGSN. View Figure 2.5 for clarification. Because a whole 2G network is spoofed, we have full control over these components of the network and their communication. This should allow us to view the communication regardless of A5 or GEA encryption. However, downgrading the encryption to A5/0 and GEA5/0 allows the communication to be captured unencrypted by means of the GSM and GPRS tap options in YateBTS rather than requiring us to investigate the back-end of the software.

In order for a ciphering algorithm to be used, a session key k_c must have been established between the mobile station and the 2G network. This session key can only be established when the mobile station is authenticated to the network by the AUC. Authentication of the mobile station to the network is achieved by using the A3 standard, while the session key k_c is established using the A8 standard. The most commonly used algorithm for this is called COMP128 and merges both A3 and A8 together. It is implemented on two sides, the SIM card and at the AUC. Part of the input to this algorithm is the shared symmetric key k_i which is stored on the SIM card and on the back-end of the GSM network. This means encryption requires a session key k_c which can only be computed when k_i is known in the network. Unfortunately, we can not assume that it is trivial to extract the SIM secret k_i from an arbitrary IoT device. Therefore, unless a self-programmed SIM is inserted (and k_i is known), encryption will have to be disabled. While this makes it trivial to analyze the radio traffic, it is also one of the detection vectors for a spoofed BTS.

6.1.1 Analysis of Captured Radio Traffic

Before we proceed with the analysis, let us present a simplified overview of the GSM and GPRS protocol stack Figure 6.1. When analyzing the captured radio

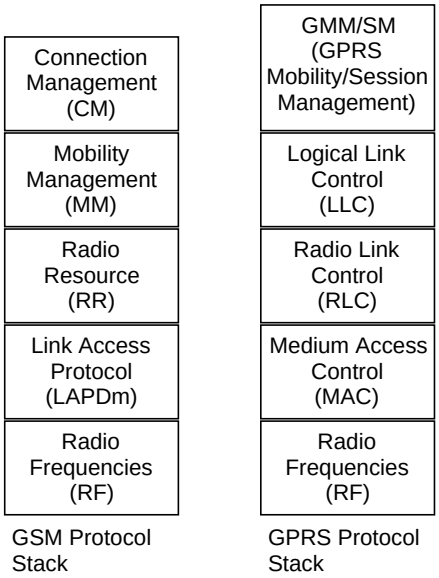


Figure 6.1: Simplified overview of GSM and GPRS protocol stack.

traffic, (part of) the messages in these protocols can be viewed and the overview helps to understand the purpose of the messages.

First of all, Broadcast Control Channel (BCCH) messages can be viewed (part of the Radio Resource (RR)). These system information type messages [4] contain the network identity, BTS configuration and features of the BTS. This can be used to check whether the parameters configured in YateBTS are actually transmitted. For example, part of the parameters mentioned in Section 4.5.3 are hard-coded into the source code (hidden in the configuration), but can be viewed in the system information type messages.

Secondly, some relevant parameters regarding the mobile station can be found. During registration of the MS to the network, both the IMEI and IMSI numbers can be captured. These can be found in the identity responses of the Mobility Management (MM) protocol. Frequent measurement reports of the mobile station on the received signal strength can be found in the RR management messages.

Thirdly, the mobile station radio access capability can be found in the routing area update request of the GPRS Mobility Management (GMM) protocol. The different radio bands that are supported by the MS are listed here, with their respective RF power capability. Additionally, it lists the supported encryption algorithms, whether EDGE is supported and many more technical parameters related to the radio capabilities of the device.

Fourth of all, in the routing area messages of the GMM protocol and the location messages of the MM protocol, information about the location of the device can be found. This data would be useful if we were to extend the spoofed network with multiple BTSs. For example, the old Routing Area Identification

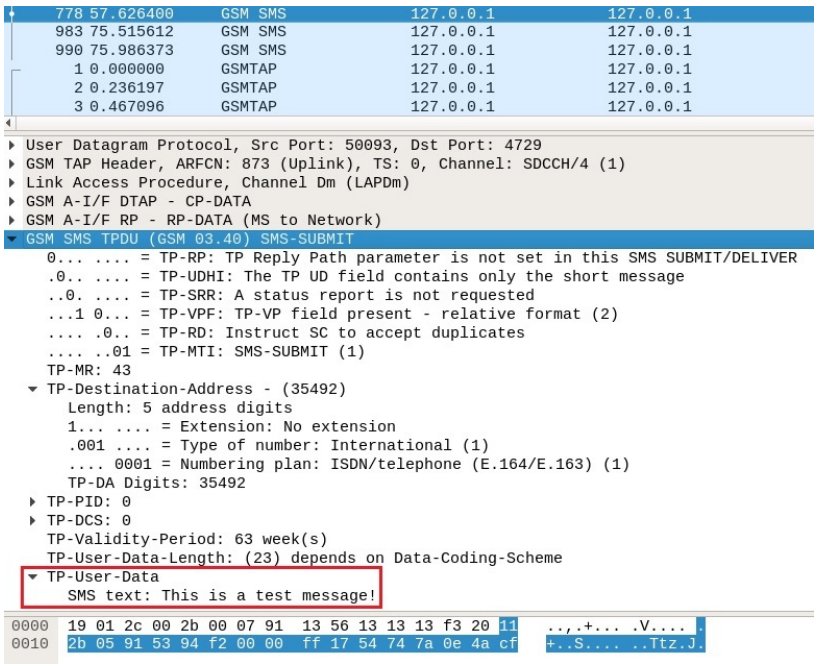


Figure 6.2: Example of a captured SMS message, which can be read in plaintext.

(RAI) can be found in a routing area update message. This parameter consists of the MCC, MNC, LAC and Routing Area Code (RAC) [2]. The RAC of GPRS is the counterpart of the CID for GSM and identifies a particular routing area within the location area.

Fifth of all, SMS messages that are sent can be viewed in plain text (Figure 6.2), including the sender and receiver. Furthermore, Call Control (CC) messages can be viewed whenever a call is made. These messages are used to establish, connect and disconnect a call.

Finally, a valuable piece of information that can be retrieved, is the Access Point Name (APN) and corresponding login credentials. This information can be found in the Session Management (SM) messages. The APN provides a reference to a GGSN [2]. The APN is translated into an IP address by means of an internal GPRS DNS in order to make inter-PLMN roaming possible. It is specified by means of labels which are concatenated by means of dots, e.g. `Label1.Label2.Label3`. It consists of a network identifier and an optional operator identifier. The network identifier must consist of at least one label. The operator identifier has the following format: `mncXXX.mccXXX.gprs`, where each `X` is a digit. An example of an APN is `testnet.mnc123.mcc456.gprs`.

As for authentication options, there are three ways:

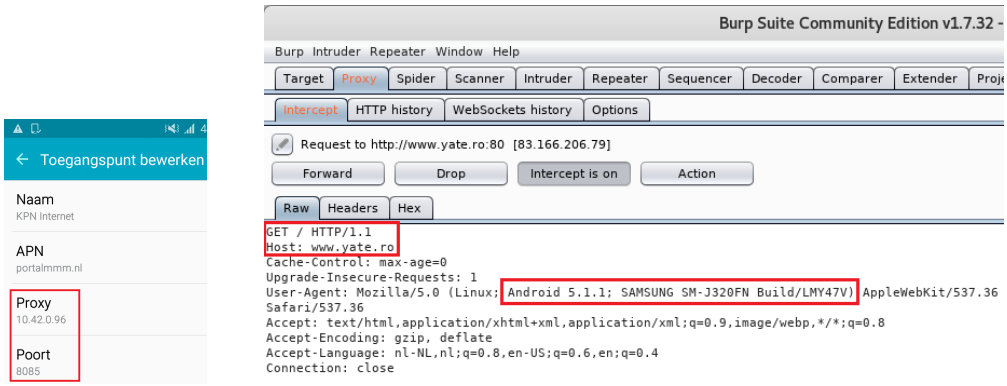
1. 'none'.
2. Password Authentication Protocol (PAP), the client provides username and password in plaintext. The server replies with an acknowledgment (ACK) if the credentials are correct or negative acknowledgment (NAK) otherwise.

3. Challenge Handshake Authentication Protocol (CHAP) uses a three way handshake. The server provides a challenge, the client calculates a response and finally the server replies whether the response matches the expected response (Success/Failure). While the APN information is ignored by YateBTS, the three way handshake still occurs. By observing the data in the messages, the challenge and response can be found. According to the RFC [46], "*The Challenge Value is a variable stream of octets. The length of the Challenge Value depends upon the method used to generate the octets, and is independent of the hash algorithm used.*" and "*The Response Value is the one-way hash calculated over a stream of octets consisting of the Identifier, followed by (concatenated with) the "secret", followed by (concatenated with) the Challenge Value. The length of the Response Value depends upon the hash algorithm used (16 octets for MD5).*" Hence the response consists of `MD5(id‖secret‖challenge)` which has a length of 16 bytes. Since the id, challenge and hash are known, a brute-force attack can be performed to try to recover the secret. MD5 has been demonstrated to be a weak algorithm and are unsuitable to protect a short secret. According to the RFC, it is recommended that the password is at least the length of the hashing algorithm used, e.g. 16 for MD5.

An IoT device might be using a private APN. This refers to the use of a service that captures the IP traffic leaving the device and routes it to an endpoint in a corporate network. The benefit of using such a service, is that the corporate infrastructure is not exposed to the whole Internet. Additionally, only devices that are also connected to that APN can route traffic to the device. This will help to protect against other users on the cellular network. When credentials for such a private APN have been obtained by an attacker (in combination with the SIM card), he can gain access to the internal network (e.g. by configuring the APN on another device). This would allow him to investigate the infrastructure of the internal network and other connected users.

6.2 Data Traffic

As previously mentioned, all traffic flowing through and from the device connected to YateBTS is directed through the `sgsntun` interface. This will make it trivial to capture the data traffic from the devices; we simple capture all traffic on this interface. As previously mentioned, Source Network Address Translation (SNAT) is applied to redirect the traffic from the private interface to the Internet. The traffic is captured before SNAT is applied, which keeps the source IP address intact for the capture. While the traffic flows freely through the interfaces and allows to be captured, it does not necessarily have to be plaintext. A device can setup an SSL/TLS tunnel and/or connect to a VPN. While analyzing the traffic is interesting, it would be useful to also be able to modify it. This can be achieved by means of a proxy server, which acts as a 'middle man' between the client and endpoint to which requests are made. The proxy can be on the client side, i.e. the client is aware of the proxy. This is known as a forward proxy.



(a) The proxy configuration.

(b) A get request to http://www.yate.ro is intercepted using Burp Suite.

Figure 6.3: Example of a HTTP interception by means of a forward proxy.

On the other hand, the proxy can be at the server side, i.e. the client does not know the whole chain towards the endpoint and is not aware of the proxy. This is known as a reverse proxy.

Burp Suite offers (among other functionalities) a proxy server for HTTP and HTTPS requests and can be configured to function as either a forward or reverse proxy. This allows the user to obtain a middle man position on the HTTP(S) requests: requests can be intercepted, dropped or modified. While it is nice to intercept HTTP(S) traffic, an IoT device is not very likely to connect to a web server. Therefore, we would like to obtain a middle man position for other traffic as well. This is where the software Scapy comes into play.

In order to approach this problem step by step, let us first setup a forward proxy using Burp and the mobile phones. Using Burp, we can setup a proxy server on the host machine. This should be an IP address that is accessible from the connected device, e.g. `10.42.0.92:8085`; the address assigned to the host device by the router of the connected network. Under the configuration for an access point of the cellular network on the test mobile phone, a proxy can be configured using the address of our proxy server, see Figure 6.3a. HTTP requests can then be intercepted with Burp (Figure 6.3b). It is also possible to intercept HTTPS traffic, but the browser will notify the user of an untrusted certificate. This SSL certificate is generated by Burp and is signed by its own Certificate Authority.

Now that the forward proxy is operating, we want to hide the use of the proxy to the device by setting up a reverse proxy. Rather than configuring the proxy settings (Figure 6.3a) on the device, DNAT is used to route the traffic on the host device to the proxy server. This can be achieved by means of an iptables rule (Listing 6.1) and by setting the proxy in Burp to invisible (Figure 6.4). Once this is configured, traffic can be intercepted similarly to the forward proxy (as in Figure 6.3b). However, the device is not aware of this happening. When the device checks for certificates, it will again notices the certificate used by Burp Suite is untrusted.

```
sudo iptables -t nat -A PREROUTING -i sgsntun -p tcp --match multiport \
--dports 80,443 -j DNAT --to 10.42.0.96:8085
```

Listing 6.1: iptables rule to redirect HTTP and HTTPS traffic to the proxy server on 10.42.0.96:8085.

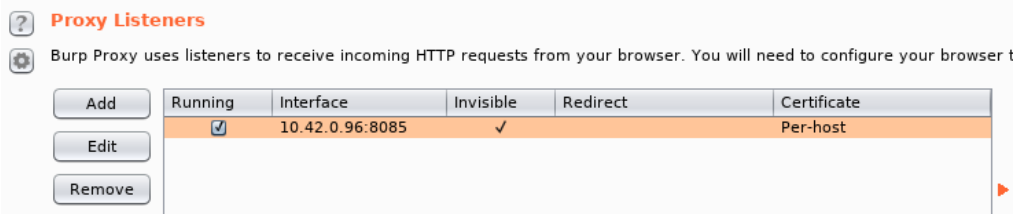


Figure 6.4: A transparent or reverse proxy configured in Burp Suite.

After setting up the reverse proxy, next we want to extend this attack to other traffic (besides HTTP/HTTPS) as well. In order to do this, we use Scapy to view and possibly modify the packets. Scapy is a packet manipulation tool written in Python. In order to feed traffic into Scapy, we use `libnetfilter_queue`. This is a userspace library that provides an API for packets that have been filtered by the kernel packet filter. The packet filtering will again be achieved by means of an iptables rule. In short, the traffic is placed in a queue, which can then be accessed by Scapy to view, modify, redirect, drop or forward the packets. A script to interface with and setup the queue can be found in Appendix C.1.

However, it is still possible for a device to setup an SSL tunnel. In such a case, the traffic is encrypted. Similar to how Burp Suite can intercept HTTPS traffic by means of a SSL-to-SSL proxy, we would like to achieve the same with an SSL tunnel for non-HTTPS traffic. This requires us to generate an SSL certificate and sign it ourselves. The certificate will then be used when we act as a server towards the traffic sent by the device. Because this certificate is self-signed, it is untrusted by the device. Hence this MITM attack rests on the assumption that the client device does not check the certificate used by our server. We can use the program `socat` (SOcket CAT) to setup a bi-directional data stream between sockets in order to create a (minimalistic) SSL-to-SSL proxy (Figure 6.5). The commands to perform this action are described in Appendix C.2.

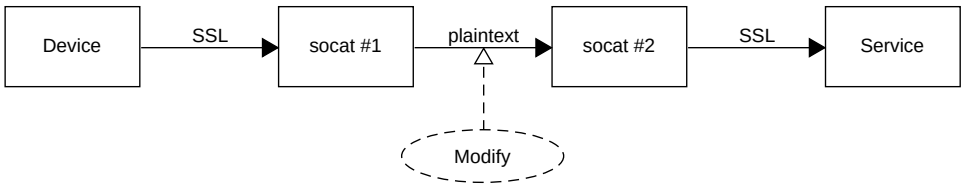


Figure 6.5: A simplified representation of the SSL-to-SSL proxy by using socat.

6.3 DNS Spoofing

The IoT device in question might request several domain names in order to connect to particular servers. It can be interesting to make the device connect to a server that we control. In such a case we want to redirect the device. Because we can modify the data traffic (as seen in the previous section), we can redirect the packets to a service that we can control. However, it is rather tedious to change the IP addresses in every packet that goes through our proxy. An easier approach is to setup a DNS service on our host machine and set this as the DNS used for the connected mobile stations in YateBTS. Once this is configured, we want to redirect all the DNS requests to an official DNS server (e.g. Google's) except for the domain names that we are interested in. `dnsmasq` is such a tool that can be used to achieve this goal. The program will offer DNS service on port 53 of our host machine. One or multiple DNS servers can be configured to be used as upstream DNS. A hosts file (e.g. `/etc/dnsmasq.hosts`) can be used to map IPs to domain names, which will be queried before the upstream DNS servers. Furthermore, the program will give us a clean output of all the DNS queries made by the device. See Listing 6.2 for an example of `dnsmasq` in action for a connected mobile station.

```
# /etc/dnsmasq.conf
no-dhcp-interface=
server=8.8.8.8

no-hosts
addn-hosts=/etc/dnsmasq.hosts

root@kali:~# dnsmasq --no-daemon --log-queries
dnsmasq: started, version 2.78 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6 no-Lua
    TFTP conntrack ipset auth DNSSEC loop-detect inotify
dnsmasq: using nameserver 8.8.8.8#53
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 8.8.8.8#53
dnsmasq: using nameserver 10.42.0.1#53
dnsmasq: read /etc/dnsmasq.hosts - 4 addresses
dnsmasq: query[A] www.google.com from 192.168.99.1
dnsmasq: forwarded www.google.com to 10.42.0.1
dnsmasq: reply www.google.com is 216.58.206.4
dnsmasq: query[A] www.dnstest123.com from 192.168.99.1
dnsmasq: /etc/dnsmasq.hosts www.dnstest123.com is 10.0.0.1
```

Listing 6.2: Example of dnsmasq configuration and output for DNS queries.

6.4 Services

In order to check for available services that are running on a connected mobile station device, a port scan can be ran. `nmap` is such a program that can be used to scan a network for available hosts and services. Using this program, we can launch a scan on the connected mobile station, because it is assigned a local

IP address by yateBTS. We have to take into account, however, that there are 65535 TCP and UDP ports and that one round-trip time (RTT) might take a considerable time (seconds) on 2G. This depends on many factors like the signal quality and other activities on the device. Therefore, scanning all ports on a device can take hours or even days. To reduce the time of these port scans, one might consider scanning only a subset of the ports. In particular, UDP scans are a hassle due to the nature of the protocol. Luckily, `nmap` has the possibility to scan for a list of most used ports. For example, the user can decide to scan for the top 1000 ports.

To test whether open services on a device can be reached through 2G with YateBTS, a test phone was used to offer an SSH service on a particular port. Next, `nmap` was used to verify the port was open and the service available. Lastly, it was tested whether SSH into the phone was possible through the connection offered via YateBTS. It turned out that this was indeed possible, but performance was rather dreadful (Listing 6.3).

```
root@kali:~# ping 192.168.99.1
PING 192.168.99.1 (192.168.99.1) 56(84) bytes of data.
64 bytes from 192.168.99.1: icmp_seq=1 ttl=63 time=12292 ms
64 bytes from 192.168.99.1: icmp_seq=2 ttl=63 time=12563 ms
64 bytes from 192.168.99.1: icmp_seq=3 ttl=63 time=12188 ms
^C
--- 192.168.99.1 ping statistics ---
16 packets transmitted, 3 received, 81% packet loss, time 15056ms
rtt min/avg/max/mdev = 12188.123/12348.156/12563.370/158.084 ms, pipe 13

root@kali:~# nmap -p 1111 192.168.99.1 --min-rtt-timeout 15000ms

Starting Nmap 7.60 ( https://nmap.org ) at 2018-09-07 11:07 CEST
Nmap scan report for 192.168.99.1
Host is up (2.7s latency).

PORT     STATE SERVICE
1111/tcp open  lmsocialserver

Nmap done: 1 IP address (1 host up) scanned in 6.55 seconds

root@kali:~# ssh 192.168.99.1 -p 1111
The authenticity of host '[192.168.99.1]:1111 ([192.168.99.1]:1111)' can't be
    established.
ECDSA key fingerprint is SHA256:feLhXLQSPt33VYV+PESoil/n/Pa7gH8QUIwHJQWstWM.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[192.168.99.1]:1111' (ECDSA) to the list of known
    hosts.
root@192.168.99.1's password:
user@j3xnlte:/data/data/org.galexander.sshd/files $ ls
dropbear.err
dropbear.pid
```

Listing 6.3: Pinging a connected mobile station, scanning for open port 1111 and connecting using SSH.

6.5 Discussion

Reflection on Research Questions

3. What communication can be observed from a connected device? The communication can be divided into radio and data traffic. Because, the 2G design has no mutual authentication, the GSM and GPRS encryption on this communication can be disabled. A device still has the possibility to setup an encrypted channel on the IP layer, e.g. a VPN connection or SSL tunnel. This could ensure data confidentiality and possibly integrity.

4. What discoverable information about the device is relevant? The most notable findings are listed. In the radio traffic, we saw the IMSI and APN credentials can be recovered. Furthermore, SMS messages and call setups can be found. Lastly, the radio access capability can be found, which lists supported radio bands, encryption algorithms and EDGE support. The information in the data traffic varies heavily based on the device tested. Typically, DNS requests are present. Finally, we have seen a port scan can be performed on devices to identify available services.

5. What active attacks can be performed on the device? SMS messages and calls can be generated and sent to connected devices. Next, we have demonstrated a MITM attack that can be performed on the data traffic, which allows the traffic to be altered. It is not possible to perform such an attack when a channel is setup between the device and endpoint that preserves data integrity. For example, we have shown an attack where we MITM an SSL connection. This attack relies on the assumption that the device does not check for trusted certificates. A vulnerable SSL implementation can also put the data integrity at risk and allow for such a MITM attack. Furthermore, we have seen a DNS server can be configured in the software. This allows for custom DNS responses to requests sent by the device. Lastly, open ports found on the device can be an interesting attack vector for active attacks.

Chapter 7

Automated Network Spoofing

Now that the research questions have been answered, the project goal will be further refined into specific requirements. Next, the software will be introduced. Finally, the software will be tested on the IoT devices and the findings will be reported.

7.1 Goals

The main goal was defined as follows.

Goal of the project: *Create software that can brute-force 2G networks and automatically performs reconnaissance once a device has successfully connected.*

We refine this into the following subgoals. The software will provide the ability to:

- Define and set different network profiles with the following parameters: GSM radio band, radio channel, MCC, MNC, LAC and CID.
- Set the transmission power.
- Detect connected devices.
- Print the IMSI, assigned IP and APN credentials of connected devices.
- Capture the radio and data traffic.
- Brute-force the defined network profiles until a device connects.
- Toggle power supply of a device during brute-force.
- Perform a port scan on connected devices.

7.1.1 Program Language Selection

When selecting the program language used for the software, we have to consider the requirements. The functionality that will be implemented does not require computationally intensive operations. In fact, it primarily exists of reading and writing configuration files, parsing captured data and sending data to sockets. The computationally intensive operations will happen in other processes that will be running simultaneously. Python was selected, because speed will not be

an important factor. It will cut down on the development time due to the many different libraries that are available and because less code will have to be written. It allows the code to be simpler and more compact. This will make it easier to understand and extend should another developer pick up the project. Python 3 was chosen over 2, because of the increased update frequency and improvements.

7.2 YateBTS Automation Tool

A command-line tool was built to achieve the previously specified goals, see Listing 7.1. This allows the tool to be used over SSH and allows the user to tie together multiple instructions or execute only one. Furthermore, the command(s) can be placed in a startup script in order to run the Odroid headless and automated. The configuration file is `config/settings.py`, the output can be found in `output/` and the Yate log file in `log/yate.log`.

Different network profiles can be defined in the configuration file, which can be selected by using the `-n` argument or listed by using `-L`. The power attenuation of YateBTS can be configured by using the `-p` argument, where a lower value will result in a higher transmission power. By selecting a network profile or configuring the transmission power, the associated parameters are written to the YateBTS configuration file by means of regular expressions. Individual subscribers can be added and listed, or a regular expression can be configured for IMSI numbers that should be accepted. YateBTS can be started, restarted and stopped by means of the `-y` argument. Additionally, the user can connect to the socket that Yate will be running on to execute commands.

With the `-b` argument, the networks defined in the configuration file will be spoofed for `-t n` seconds one after the other. When a device registers with the spoofed network, the brute-force will stop and the next command is executed (or the program is terminated). Instead of a brute-force, we can also simply poll for a device by means of the `-s` argument. When a device connects during a search (`-b` or `-s`), the IMSI and IP are printed. If traffic capturing is enabled, the APN credentials are also printed. By using the `-P` argument, the user can enable signaling of a GPIO pin which is also defined in the `config/settings.py` file. Hardkernel provided a WiringPi library for the Odroid boards that allows us to interface with the GPIO pins. It also comes with a Python 3 wrapper. see[1]. By connecting the ground, 5V and a GPIO pin to a relay, we can cut the power on the connected device if it uses an external power supply. The time for which the power will be cut can be specified by using the `--cut-time n` argument, with `n` the time in seconds.

The devices currently connected with the spoofed network can be listed using `--list-devices`. Furthermore, we can capture the radio and data traffic with `-c` for `-T n` seconds. With `-N` an `nmap` scan can be executed on the devices currently connected. The `nmap` command that will be executed can be specified in the configuration file. The captured traffic and the result of the `nmap` scans can be found in the output folder.

[1] `https://wiki.odroid.com/odroid-xu4/application_note/gpio/wiringpi`

```
root@kali:~/yatescript# ./ybts-mt.py -h
usage: ybts-mt.py [-h] [-n {0,1,2,3,4,5,6,7,8}] [-L] [-p n] [--regex REGEX]
                  [--add-subscriber n [n ...]] [--del-subscriber IMSI]
                  [--list-subscriber]
                  [-y {start,restart,stop,connect,startconnect}] [-b] [-P]
                  [--cut-time n] [-s] [-t n] [--list-devices] [-c] [-T n] [-N]

YateBTS configuration and automation multi-tool. The configuration can be
found in config/settings.py. Captured output can be found in the output
folder. Note: double check that the configured frequencies in the config file
are allowed according to local law or that transmission is contained in such a
way that no interference is possible. Disclaimer: this software comes with
absolutely no warranty.

optional arguments:
  -h, --help            show this help message and exit
  -n {0,1,2,3,4,5,6,7,8}, --net {0,1,2,3,4,5,6,7,8}
                        Configure YateBTS for a test network (0) or any of the
                        network profiles specified in the configuration file
                        (1..N).
  -L                    List the network profiles in the configuration file.
  -p n, --power n       Configure YateBTS power attenuation (0..80) dB, lower
                        value => higher output power.
  --regex REGEX         Set an IMSI regular expression rule to allow
                        subscriber to register. Wrap arguments like .* in
                        quotes. Example: ^204 allows all IMSI numbers starting
                        with 204. ".*" allows all numbers.
  --add-subscriber n [n ...]
                        Add/update a subscriber to the YateBTS subscriber
                        whitelist with the following format: "IMSI MSISDN
                        short_num ki" or "IMSI MSISDN short_num" if no ki
                        should be set. IMSI uniquely identifies the
                        subscriber, it should be 15 digits. MSISDN is the
                        number mapped to the subscriber by the network.
                        Recommended to keep between 1 and 15 digits. This
                        number is dialed when calling from the outside.
                        short_num is the number you dial to call/sms the
                        subscriber WITHIN the local network. Recommended to
                        keep between 1 and 15 digits. ki is the secret
                        symmetric key in the SIM and must be a 128-bit key in
                        hex format, i.e. 16 bytes in hex with no prefix.
                        Adding subscribers disables the IMSI regular
                        expression rule.
  --del-subscriber IMSI
                        Delete a subscriber based on IMSI from the YateBTS
                        whitelist, IMSI should be 15 digits
  --list-subscriber     List the accepted subscribers on the YateBTS
                        whitelist.
  -y {start,restart,stop,connect,startconnect}, --yate {start,restart,stop,
    connect,startconnect}
                        Start, restart, stop or connect to Yate.
  -b, --bruteforce      Brute-force all networks defined in the configuration
                        file until a device connects.
  -P                    Enable signaling of relay to cut power briefly after
                        every spoofed network during bruteforce. Uses
                        WiringPI. The signalling pin is defined in the config
```

```
                        file. Time for which the power will be cut can be
                        defined with the following argument.
--cut-time n            Sets the time in seconds for which power will be cut
                        during bruteforce. Default is 2 seconds which can be
                        changed in the config file.
-s, --search            Search on the network currently broadcast by YateBTS
                        for device connection.
-t n, --search-timeout n
                        Set the timeout value for device connection during a
                        search. Default is 60 seconds which can be changed in
                        the config file, minimum is 30 seconds.
--list-devices          List IMSI and IP address of devices currently
                        connected to YateBTS.
-c, --capture           Enable the capturing of radio and data traffic.
-T n, --capture-timeout n
                        Set the timeout value in seconds for the capturing of
                        traffic. Default is 60 seconds which can be changed in
                        the config file.
-N, --nmap              Execute the nmap command listed in the configuration
                        file on the currently connected devices. Output can be
                        found in the output folder.
```

Listing 7.1: The YateBTS automation tool.

7.3 Notable Findings on Test IoT Devices

The software was used to assist in discovering findings on the test devices. Unfortunately, we were unable to test the effectiveness of the brute-force due to the following problems. The smart switch failed to connect successfully to YateBTS. The tracker does not have an external power supply, hence can not be toggled. Furthermore, the device appeared to be roaming and connected successfully to any network. Lastly, the RTU does not turn off when the power is cut, but is designed in such a way that alert messages (by means of SMS and app notifications) can be sent to warn the user when power is cut. We were, however, able to use the tool to quickly switch between networks, capture the traffic, recover details and perform a port scan.

7.3.1 Sonoff G1

As previously mentioned, the Sonoff G1 smart switch failed to connect successfully to the network. We can, however, view the IMSI, IMEI and radio capabilities. The radio capabilities list the supported encryption algorithms, which lists support for A5/1, depreciated A5/2 and GEA/1. Furthermore, the radio traffic lists that the GSM-900 and GSM-1800 bands are supported, but entries for GSM-850 and GSM-1900 are lacking. The official specification lists [1] that either the 1800 or 1900 band should be advertised, but not both. While these bands might not be advertised, they are supported according to the specification and the device successfully connected on all bands. Finally, the device lists that EDGE is not supported.

7.3.2 GPS Tracker

In the radio traffic, we can find that the A5/1, A5/3 and GEA/1 are supported. It also correctly lists the supported bands. There is an absence of EDGE implementation. The IMEI number can be retrieved, as well as the IMSI number: it starts with [redacted].

More importantly, the APN credentials can be found. In this case, CHAP is used for authentication. An attempt was made to crack the MD5 hash with Hashcat. Hashcat is a brute-force password cracking tool, which can make use of GPUs to severely speed up the process. We were successful in doing so (Listing 7.2). This gives a suspicion about the password structure used in the devices: [redacted]. As previously mentioned, the RFC for the CHAP protocol recommends a password of length 16 for MD5 [46], which the developers clearly ignored.

```
Name: [redacted]
APN: [redacted]
Challenge: e096aa1edca55500
Response: [redacted]
Identifier: 01

> .\hashcat64.exe -w 3 -m 10 -a 3 -i --hex-salt --hex-charset -1 ?d?u?l
[redacted]:e096aa1edca55500 01?1?1?1?1?1?1?1

[redacted]:$HEX[01[redacted]e096aa1edca55500]

root@kali:~# echo -e "[redacted]"
[redacted]
```

Listing 7.2: Cracking the APN CHAP authentication hash using hashcat.

Regarding the data traffic, DNS requests to two domains can be found: [redacted] and [redacted]. The device tries to perform a TCP handshake with [redacted] but fails. However, some information can be gathered from UDP packets that are exchanged (Listing 7.3). The LEON-G100 is a GSM/GPRS Quadband by ublox [56]. It uses a positioning technology called CellLocate to derive a rough location estimate based on the BTS when GPS is unavailable. When we look at the server response, it looks like an HTTP-like protocol transmitted over UDP, because of the Content-Length and Content-Type headers. An `nmap` scan was ran to check for the top 1000 TCP and UDP ports, but no ports were found to be open. Unfortunately, there was no time to further investigate the device and protocol. A possible next step would be to scan the ports used in the data traffic. The device does not setup an SSL connection.

```
Sent by device (52 bytes):
0000   47 53 4d 43 45 4c 13 36 37 89 32 6c 6b 81 01 32  GSMCEL.67.2lk..2
0010   02 7e 04 03 08 04 02 81 1a 4c 45 4f 4e 2d 47 31  .~.......LEON-G1
0020   30 30 2f 55 72 62 65 31 30 30 5f 30 37 2e 39 32  00/Urbe100_07.92
0030   2e 30 32 00                                      .02.

Server response (3705 bytes):
```

```
0000   75 2d 62 6c 6f 78 20 43 6f 70 79 72 69 67 68 74  u-blox Copyright
0010   20 28 63 29 20 32 30 31 32 2d 32 30 31 33 20 75   (c) 2012-2013 u
0020   2d 62 6c 6f 78 20 41 47 0d 0a 43 6f 6e 74 65 6e  -blox AG..Conten
0030   74 2d 4c 65 6e 67 74 68 3a 20 33 36 30 38 0d 0a  t-Length: 3608..
0040   43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 61 70  Content-Type: ap
0050   70 6c 69 63 61 74 69 6f 6e 2f 75 62 78 0d 0a 0d  plication/ubx...
...
```

Listing 7.3: UDP packets exchanged between the GPS tracker and a server.

7.3.3 RTU

Regarding the RTU, the A5/1, A5/3, GEA/1, GEA/2 and GEA/3 encryption algorithms are supported. Again, the IMEI and IMSI can be viewed in plaintext. EDGE is not implemented for this device. We are able to obtain the old routing area identification, i.e. the old routing area the device connected to. Furthermore, we are able to sniff the APN credentials, however, we had to provide a SIM card and configure the APN ourselves.

The GPRS RTU can be controlled and programmed using SMS messages. The commands are executed based on the format: DDDD‖CC, where D is a digit and C a character in uppercase. This would imply that once an attacker finds the pin and knows the phone number, he can completely reprogram the device remotely (including arming and disarming the device). When the device is connected to the spoofed BTS, we are able to craft SMS messages and send them to the device with a plugin. If there is no limit on the tries for the pin code, we can brute-force it by crafting SMS messages.

The device performs a DNS request for [redacted]. On port `8001`, it opens a TCP connection. The device can be armed and disarmed by means of messages sent over GPRS. We are able to craft a TCP connection and get response on that port. With further protocol reversing, it might be possible to forge messages to remotely arm and disarm the RTU when the device is online. Lastly, a port scan can be performed in order to look for available services. Due to time constraints, the protocol could not be reversed and available services could not be analyzed. The device does not setup an SSL connection.

7.4 Discussion

We have seen three devices to test our setup and software. The circumstances that are required to perform automated reconnaissance on a device are specific. Because of this, we were unable to validate the brute-force network spoofing method of the software. It might, however, function under the right circumstances:

- Absence of other networks as discussed in Section 5.3.
- The device is powered by an external power supply and does not contain a backup power supply that is used when power is lost.
- Successful connection to YateBTS.

In practice, a security researcher might encounter a wide variety of devices on remote locations. Not all the right circumstances might apply for full automation to be effective. Therefore, it is likely that some manual actions will be required. The software allows to quickly switch between different network profiles, which reduces time spent configuring the BTS should full automation not be possible. Furthermore, automated reconnaissance can be executed by chaining together commands.

Reflection on the Goal

Goal of the project: *Create software that can brute-force 2G networks and automatically performs reconnaissance once a device has successfully connected.*

Software was developed that brute-forces 2G networks. We were not able to validate the effectiveness of the brute-force, because specific circumstances are required. The software can automatically perform reconnaissance, consisting of:

- Collecting the radio and data traffic;
- Listing the IMSI, assigned IP and APN;
- Executing a port scan.

7.4.1 Future Work

Ideally, we would like the automated process to return all relevant findings in a formal output report. This would include the other findings that were mentioned like the radio capabilities of the device. Furthermore, the effect of fuzzing the connected device can be investigated in order to find more vulnerabilities. A convenient addition to the software would be the setup of active attacks with a simple command, i.e. constructing SMS messages, setting up data MITM and DNS spoofing. Finally, we would like to investigate more devices and the OsmoBTS software with different hardware, because it has EDGE capabilities.

Chapter 8

Conclusions

There are still many embedded devices around that make use of the 2G network. However, it is well known that 2G networks can be spoofed. We constructed an automated tool to perform reconnaissance of such embedded devices by making use of YateBTS and the BladeRF. The tool intends to find the network used by a device by iterating over various network profiles (brute-forcing). Before doing so, we investigated the circumstances required to have these devices connect to a spoofed 2G network and investigate what information can be observed.

The circumstances appear to be rather specific and requires either frequency jamming or the use of a Faraday cage. We used a Faraday cage to shield such devices from external networks while also allowing us to impersonate a 2G network. With this cage, we were able to perform a downgrade attack from 4G to 2G. However, we found that a reliable Faraday cage is not trivial to construct. It is advisable to purchase a professional cage meant for interaction with devices that should be shielded from external signals.

Once such a device is connected to an impersonated 2G network, we investigated the information and communication that can be observed. The most notable findings were the APN credentials that can be recovered. Furthermore, we introduced several active attacks that can be performed on connected devices. These attacks consist of performing a Man-in-the-Middle (MITM) attack on the data services, DNS spoofing and scanning for open ports on the device.

After these investigations, we implemented software that can be used to automatically perform reconnaissance on devices. Three embedded devices were used to test the functionality. We were not able to validate the effectiveness of brute-forcing 2G networks. We found that full automation required specific circumstances which might not be present outside a lab setting. Nevertheless, the process of finding out the network used by the devices is severely sped up. The tool allows to automatically perform reconnaissance on a connected device. During brief investigation of two embedded devices, we already observed peculiarities that should be further investigated.

The contributions of this thesis consists of an aggregation of possible attacks on embedded devices that make use of GPRS and it introduces a tool intended to automate the reconnaissance of penetration testing on these embedded devices.

Bibliography

[1] 3GPP, "TS 05.14, Digital cellular telecommunications system (Phase 2+); Release independent frequency bands; Implementation guidelines," https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=266, pp. 9–13, 2001, accessed: July 2018. 66

[2] ——, "TS 23.003, 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Numbering, addressing and identification (Release 15)," https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=729, pp. 17–20, 30–32, 34–36 and 38–40, 2018, accessed: July 2018. 32, 56

[3] ——, "TS 43.020, 3rd Generation Partnership Project; Technical Specification Group Services and system Aspects; Security related network functions (Release 15)," 2018, accessed: July 2018. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2663 36

[4] ——, "TS 44.018, 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Mobile radio interface layer 3 specification; GSM/EDGE Radio Resource Control (RRC) protocol (Release 15)," pp. 245–258, 2018, accessed: July 2018. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2686 41, 55

[5] Aeris, "The Global State of 2G, 3G, 4G, and 5G," https://info.aeris.com/the-state-of-2g-3g-4g-lte-worldwide-whitepaper-website, 2015, accessed: July 2018. 1

[6] Antenna-Theory.com, "Cell Phone Antenna Design," http://www.antenna-theory.com/design/cellantenna.php, 2012-2016, accessed: March 2018. 9

[7] A. Chemeris, "Software Defined Radio hardware for Osmocom BTS," 2017, accessed: July 2018. [Online]. Available: https://osmocom.org/attachments/download/2616/SDR%20based%20BTS.pdf 27

[8] Daehyun Strobel, "IMSI Catcher," Ruhr-Universität Bochum, July 2007. 21

[9] Disklabs, "Laptop Shield (LS1) Faraday Bag RF Shielding," 2018, accessed: March 2018. [Online]. Available: http://faradaybag.com/products/laptop-shield-ls1-faraday-bag-rf-shielding/ 39

[10] DLP Design, "USB-Based Latching Relay Module," 2007, accessed: July 2018. 43

[11] Ericsson AB, "Internet of Things forecast," https://www.ericsson.com/en/mobility-report/internet-of-things-forecast, 2016, accessed: March 2018. 1

[12] ETSI, "Digital cellular telecommunications system (Phase 2+); Radio subsystem synchronisation (GSM 05.10)," GSM Technical Specification, 1996, accessed: March 2018. 26

[13] M. M. J. French, "A mobile phone Faraday cage," *Physics Education*, vol. 46, 2011, accessed: March 2018. 38

[14] GSM Map project, "Mobile network security report: Netherlands," Security Research Labs, Tech. Rep., June 2007, https://gsmmap.org/assets/pdfs/gsmmap.org-country_report-Netherlands-2017-06.pdf. Accessed: March 2018. 2

[15] interactive digital media GmbH, "Mobile Country Codes (MCC) and Mobile Network Codes (MNC)," 2013, accessed: March 2018. [Online]. Available: http://www.mcc-mnc.com/ 35

[16] M. Jaatun, I. Tøndel, and G. Køie, "GPRS Security for Smart Meter," in *Availability, Reliability, and Security in Information Systems*, ser. Lecture Notes in Computer Science, C. A., K. C., S. D.E., W. E., and X. L, Eds. Springer, 2013, vol. 8127, pp. 195–20. 1

[17] Joseph Ooi, "IMSI Catchers and Mobile Security," *University of Pennsylvania*, 2015, accessed: July 2018. 42

[18] King Pigeon Hi-Tech.Co.,Ltd, "4G RTU 3G Modem GSM RTU GPRS RTU,SMS Controller,4G DTU," 2018, accessed: May 2018. [Online]. Available: http://www.gprs-m2m.com/enproductShow.asp?ID=424 28

[19] H. Lecht, "2G and 3G networks are shutting down globally?!" https://1ot.mobi/blog/2g-and-3g-networks-are-shutting-down-globally, march, 2018, accessed: July 2018. 2

[20] R. Moonen, "Practical GPRS MitM attack- mobile setup with YateBTS," https://www.secura.com/blog-Practical-GPRS-MitM-attack, December 2017, accessed: March 2018. 2, 24

[21] P. Nay, I. Smith, G. Cadamuro, and T. Kohno, "SeaGlass: Enabling City-Wide IMSI-Catcher Detection," Proceedings on Privacy Enhancing Technologies, 2017, accessed: April 2018. 21

[22] K. Nohl, "Attacking phone privacy," Blackhat 2010, 2010, accessed: June 2018. 54

[23] K. Nohl and L. Melette, "GPRS Intercept: Wardriving your country," Security Research Labs, Tech. Rep., 2011, accessed: August 2018. 54

[24] Nuand, "Nuand — bladeRF Software Defined Radio," https://www.nuand.com/bladeRF, accessed: March 2018. 26

[25] ——, "bladeRF FPGA images," 2017, accessed: March 2018. [Online]. Available: https://nuand.com/fpga.php 30

[26] Nuand GitHub, "Nuand/kalibrate-bladeRF: kalibrate-bladeRF," 2010, accessed: March 2018. [Online]. Available: https://github.com/Nuand/kalibrate-bladeRF 27

[27] ——, "bladeRF Power Consumption Nuand/bladeRF Wiki," 2017, accessed: March 2018. [Online]. Available: https://github.com/Nuand/bladeRF/wiki/bladeRF-Power-Consumption 27

[28] ——, "Setting up Yate and YateBTS with the bladeRF," 2017, accessed: March 2018. [Online]. Available: https://github.com/Nuand/bladeRF/wiki/Setting-up-Yate-and-YateBTS-with-the-bladeRF 84

[29] ——, "Nuand/bladeRF: bladeRF USB 3.0 Superspeed Software Defined Radio Source Code," 2018, accessed: March 2018. [Online]. Available: https://github.com/Nuand/bladeRF 29

[30] Odroid Hardkernel, "XU4 Shifter Shield," 2013, accessed: July 2018. 43

[31] ——, "Odroid-XU4," 2017, accessed: March 2018. [Online]. Available: https://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825 27

[32] ODROID Wiki, "Linux Release note," 2017, accessed: March 2018. [Online]. Available: https://wiki.odroid.com/odroid-xu4/gettingstart/linux/install_linux 82

[33] ——, "Linux Release note," 2018, accessed: March 2018. [Online]. Available: https://wiki.odroid.com/odroid-xu4/os_images/linux/start 29

[34] OpenBTS.org, "Clocks - OpenBTS," http://openbts.org/w/index.php?title=Clocks, accessed: March 2018. 27

[35] Osmocom Github, "GPRS decoder for OsmocomBB," 2018, accessed: August 2018. [Online]. Available: https://github.com/osmocom/osmocom-bb/tree/master/src/host/gprsdecode 54

[36] Overheid.nl, "Regeling gebruik van frequentieruimte zonder vergunning en zonder meldingsplicht 2015," 2016, accessed: March 2018. [Online]. Available: http://wetten.overheid.nl/BWBR0036378/2016-12-28 34

[37] C. Paget, "DEF CON 18 - Chris Paget - Practical Cellphone Spying," 2010, accessed: March 2018. [Online]. Available: https://www.youtube.com/watch?v=fQSu9cBaojc 2, 21, 23

[38] N. Paglieri and O. Benjamin, "Implementation and performance analysis of Barkan, Biham and Kellers attack on A5/2," https://www.npag.fr/doc/papers/a52hacktool.pdf, 2011, accessed: June 2018. 54

[39] J. Pearce, "Operator will not switch off 2G network until 2025 in order to support legacy IoT connections," https://www.globaltelecomsbusiness.com/article/b14056696hc3lj/exclusive-vodafone-unveils-plans-to-switch-off-2g-but-not-until-2025, July 2017, accessed: March 2018. 3

[40] D. Perez and J. Pico, "A practical attack against GPRS/EDGE/UMTS/HSPA mobile data communications," Black Hat DC, 2011 (Jan. 18-19). 2, 21

[41] Qualcomm, "HSPA+ Advanced Taking HSPA+ to the Next Level," https://www.qualcomm.com/documents/hspa-advanced-taking-hspa-next-level-whitepaper, 2012, accessed: April 2018. 13

[42] ——, "The Evolution of Mobile Technologies: 1G to 2G to 3G to 4G LTE," https://www.qualcomm.com/documents/evolution-mobile-technologies-1g-2g-3g-4g-lte/, June 2014, accessed: April 2018. 9, 11, 12

[43] Ramsey Electronics, "STE3000 RF Test Shielded Enclosure - Ramseytest.com," 2018, accessed: July 2018. [Online]. Available: http://www.ramseyelectronics.com/product.php?pid=10 39, 52

[44] M. Y. Rhee, "Mobile communication systems and security," Kyung Hee University, 2009, wiley : IEEE Press. 2

[45] K. v. Rijsbergen, "The effectiveness of a homemade IMSI catcher build with YateBTS and a BladeRF," University of Amsterdam, 2016. 21, 36, 37

[46] W. Simpson, "PPP Challenge Handshake Authentication Protocol (CHAP)," Internet Requests for Comments, RFC 1994, August 1996, accessed: July 2018. [Online]. Available: https://tools.ietf.org/html/rfc1994 57, 67

[47] SnoopSnitch - opensource.srlabs.de, "IMSI Catcher Score - SnoopSnitch - SRLabs Open Source Projects," 2018, accessed: July 2018. [Online]. Available: https://opensource.srlabs.de/projects/snoopsnitch/wiki/IMSI_Catcher_Score 42

[48] I. Soican and Arhifane, "Yatebts vs openbts," https://wiki.yatebts.com/index.php/File:Yatebts_vs_openbts1.png, 2014, accessed: May 2018. xi, 24

[49] Sonoff, "Sonoff G1," 2016, accessed: May 2018. [Online]. Available: http://sonoff.itead.cc/en/products/sonoff/sonoff-g1 28

[50] spectrummonitoring.com, "List of Mobile Frequencies by Country (GSM, CDMA, UMTS, LTE)," 2017, accessed: March 2018. [Online]. Available: https://www.spectrummonitoring.com/frequencies/#Netherlands 34, 36, 38, 48

[51] S.P.K. ELECTRONICS CO. LTD., "SPK-WA-WTH43003," http://www.sparkfun.com/datasheets/Cellular%20Modules/GSM%20Antenna-850-1900.pdf, accessed: February 2018. 26

[52] Statista, "Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)," https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/, 2018, accessed: March 2018. 1

[53] ——, "Mobile technologies market share of subscriptions worldwide forecast from 2017 to 2022," https://www.statista.com/statistics/206655/forecast-of-the-distribution-of-global-mobile-broadband-subscriptions-by-technology-in-2016/, 2018, accessed: March 2018. 1

[54] Sunfounder, "2 Channel 5V Relay Module," 2017, accessed: July 2018. 43

[55] Tractive, "GPS Tracker for dogs," 2018, accessed: May 2018. [Online]. Available: https://tractive.com/eu_en/pd/gps-tracker 28

[56] u-blox, "LEON-G1," 2013, accessed: August 2018. [Online]. Available: https://www.u-blox.com/en/product/leon-g1 67

[57] Unwired Labs, "OpenCelliD - Largest Open Database of Cell Towers & Geolocation - by Unwired Labs," 2018, accessed: March 2018. [Online]. Available: https://www.opencellid.org/ 35, 84

[58] Würth Elektronik, "Ferrites for Cable Assembly," 2018, accessed: May 2018. [Online]. Available: https://www.we-online.com/redexpert/#/module/2/applicationbar/System/on 40

[59] Yate Documentation, "Compiling and installing Yate from SVN," 2014, accessed: March 2018. [Online]. Available: http://docs.yate.ro/wiki/Compiling_and_installing_Yate_from_SVN 84

[60] YateBTS Wiki, "GPRS Troubleshooting," https://wiki.yatebts.com/index.php/GPRS_Troubleshooting, 2016, accessed: July 2018. 47

[61] ——, "Wireshark monitoring traffic inside YateBTS," https://wiki.yatebts.com/index.php/Wireshark_monitoring_traffic_inside_YateBTS, 2018, accessed: July 2018. 25

Appendix A

Abbreviations

0G Zero Generation. 9
1G First Generation. 9
2G second generation. 1–5, 10, 11, 15–17, 21, 28, 29, 35, 36, 38, 40, 47, 49, 51–54, 61–63, 69, 71
3G Third Generation. 1–3, 11–13, 15, 38, 49, 54
3GPP 3rd Generation Partnership Project. 12
3GPP2 3rd Generation Partnership Project 2. 13
4G Fourth Generation. 1, 2, 13–15, 38, 49, 52, 71
5G Fifth Generation. 12, 14

APN Access Point Name. 47, 56, 57, 62–64, 67, 69, 71
ARFCN Absolute Radio Frequency Channel Number. 32, 34, 41, 42
AUC Authentication Center. 18, 54

BCC Base station Color Code. 32
BCCH Broadcast Control Channel. 40, 41, 55
BSC Base Station Controller. 18, 23
BSIC Base Station Identity Code. 32, 36
BTS base tranceiver station. 18, 21–28, 32, 34, 35, 40–42, 48, 52–55, 67–69

CC Call Control. 56
CDMA Code Division Multiple Access. 12
CDMA2000 Code Division Multiple Access of the IMT-2000 standard. 13
cdmaOne Code Division Multiple Access One. 10, 12, 13
CHAP Challenge Handshake Authentication Protocol. 57, 67
CID Cell ID. 17, 32, 35, 41, 42, 48, 56, 63

D-AMPS Digital-Advanced Mobile Phone Service. 10
DCS Digital Cellular System. 16, 26, 32, 34
DECT Digital Enhanced Cordless Telecommunications. 34, 36
DNAT Destination Network Address Translation. 58

E-GSM Extended GSM. 16, 26, 32, 34
EDGE Enhanced Data Rates for GSM Evolution. 11, 12, 15, 23, 25, 37, 42, 55, 62, 66–69
EHF Extremely High Frequency. 8

EIR Equipment Identity Register. 18
ELF Extremely Low Frequency. 8
EM electromagnetic. 7, 8, 38

FCCH Frequency Correction Channel. 27
FDMA Frequency Division Multiple Access. 9, 10
FPGA Field-programmable gate array. 26, 29, 30

GGSN Gateway GPRS Support Node. 18, 19, 47, 56
GMM GPRS Mobility Management. 55
GPIO General Purpose Input/Output. 27, 43, 64
GPRS General Packet Radio Service. 10–12, 15, 17, 18, 21–23, 25, 28, 29, 37, 40–42, 45–56, 62, 68, 71
GPS Global Positioning System. 1, 12, 28, 29, 50, 67
GSM Global System for Mobile communications. 1, 10–12, 15, 16, 21–23, 25–29, 32–36, 40, 41, 50, 51, 53–56, 62, 63, 66, 84

HLR Home Location Register. 18
HSDPA High-Speed Downlink Packet Access. 12
HSPA High-Speed Packet Access. 12, 15
HSPA+ Evolved High Speed Packet Access. 12, 13
HSUPA High-Speed Uplink Packet Access. 12

ICCID Integrated Circuit Card Identifier. 16
IMEI International Mobile Equipment Identity. 17, 18, 55, 66–68
IMSI International Mobile Subscriber Identify. 16, 18, 21, 31, 50, 55, 62–64, 66–69
IMT-2000 International Mobile Telecommunications for the year 2000. 12
IMT-Advanced International Mobile Telecommunications-Advanced. 13, 14
IoT Internet of Things. 1, 2, 4, 21, 28, 29, 42, 49, 51, 52, 54, 57, 58, 60, 63
IS-136 Interim Standard 136. 10
IS-54 Interim Standard 54. 10
IS-95 Interim Standard 95. 12
ITU International Telecommunications Union. 8, 12, 13

LAC Location Area Code. 16, 17, 32, 35, 41, 42, 48, 56, 63
LAI Location Area Identity. 16
LTE Long Term Evolution. 12–15

M2M machine to machine. 21
MCC Mobile Country Code. 16, 17, 32, 34, 35, 40, 56, 63, 84
ME mobile equipment. 17, 18
MIMO Multiple-Input Multiple-Output. 13, 14
MITM Man-in-the-Middle. 2, 59, 62, 69, 71
MM Mobility Management. 55
MMS Multimedia Messaging Service. 10, 11
MNC Mobile Network Code. 16, 17, 32, 34, 35, 40, 56, 63
MS mobile station. 17–19, 22, 23, 27, 31, 34, 36–43, 48, 50, 53–55, 60
MSC Mobile Switching Center. 18, 23
MSIN Mobile Subscriber Identification Number. 16

NAT Network Address Translation. 45, 46
NCC Network Color Code. 32
NIPC Network in a PC. 23, 31
NITB Network in the Box. 24

OFDMA Orthogonal Frequency Division Multiple Access. 13
OpenBSC Open Base Station Controller. 24
OpenBTS Open Base Transceiver Station. 22, 23, 25
OsmoBTS Open Source Mobile Base Transceiver Station. 22–25, 69
Osmocom Open Source Mobile Communication. 23
OsmoTRX Open Source Mobile Transceiver. 24

PAP Password Authentication Protocol. 56
PCS Personal Communications Service. 16, 26, 32, 34
PLMN Public Land Mobile Network. 17, 32, 34, 47, 56
PSTN Public Switched Telephone Network. 18

RAC Routing Area Code. 56
RAI Routing Area Identification. 55, 56
RF Radio Frequency. 25, 39, 40, 55
RR Radio Resource. 55
RTT round-trip time. 61
RTU Remote Terminal Unit. 28, 29, 51, 66, 68
RX Reception. 26

SCADA Supervisory Control and Data Acquisition. 28, 51
SDR Software Defined Radio. 24–27, 30, 39, 41, 48
SGSN Serving GPRS Support Node. 18, 54
SIM Subscriber Identity Module. 16–18, 28, 37, 47–51, 54, 57, 68
SIP Session Initiation Protocol. 23
SM Session Management. 56
SMS Short Message Service. 2, 10, 11, 18, 21, 23, 42, 45, 47, 53, 56, 62, 66, 68, 69
SNAT Source Network Address Translation. 57

TDMA Time Division Multiple Access. 10, 11
TMSI Temporary Mobile Subscriber Identity. 18
TX Transmission. 26

UHF Ultra High Frequency. 8
UMB Ultra Mobile Broadband. 13
UMTS Universal Mobile Telecommunications System. 12, 13, 15, 16
USIM Universal SIM. 16
USRP Universal Software Radio Peripheral. 23–25, 27

VLR Visitor Location Register. 18
VoIP Voice over IP. 23

WiMAX Worldwide Interoperability for Microwave Access. 13–15

Yate Yet Another Telephony Engine. 23, 64
YateBTS Yet Another Telephony Engine Base Transceiver Station. 21–27, 30–34, 36–39, 42, 45–47

Appendix B

Setup Configuration

B.1 Odroid XU4 Installation

In order to flash an OS image on the eMMC module, an eMMC module reader is required. The reader converts the interface of the module to micro SD, this allows a USB SD card reader to be used to flash the image using your preferred tool. For more information, see [32]. After installation, the eMMC module can be inserted into the socket. Be sure to set the switch from 'μSD' to 'eMMC'. Now a USB mouse, USB keyboard, Ethernet cable and HDMI monitor can be connected. The Odroid will boot when the power supply is connected (otherwise press the power button) and will take a couple of minutes to setup on a fresh image. The default login credentials are `root:odroid` or `odroid:odroid`. Do note that SSH is enabled by default, so change the password. Consider configuring a static IP so SSH can be used instead of connecting a mouse, keyboard and monitor. After logging in, update the software and system:

```
sudo apt update
sudo apt upgrade
sudo apt dist-upgrade
sudo reboot
```

B.2 BladeRF Software Installation

Using PPA

```
sudo add-apt-repository ppa:bladerf/bladerf
sudo apt-get update
sudo apt-get install bladerf
```

Building from Source

```
sudo apt -y install git libusb-1.0-0 libusb-1.0-0-dev build-essential cmake \
automake libncurses5-dev libtecla1 libtecla-dev pkg-config

git clone https://github.com/Nuand/bladeRF.git ./bladeRF
```

```
cd ./bladeRF/host/
mkdir -p build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local \
-DINSTALL_UDEV_RULES=ON ../
make
sudo make install
sudo ldconfig
```

B.3 Installing Gqrx

```
sudo apt-get purge --auto-remove gqrx
sudo apt-get purge --auto-remove gqrx-sdr
sudo apt-get purge --auto-remove libgnuradio*

sudo add-apt-repository -y ppa:bladerf/bladerf
sudo add-apt-repository -y ppa:ettusresearch/uhd
sudo add-apt-repository -y ppa:myriadrf/drivers
sudo add-apt-repository -y ppa:myriadrf/gnuradio
sudo add-apt-repository -y ppa:gqrx/gqrx-sdr
sudo apt-get update

sudo apt-get install gqrx-sdr
```

B.4 Installing YateBTS

```
apt -y install subversion telnet apache2 php libapache2-mod-php7.0 \
libusb-1.0-0 libusb-1.0-0-dbg libusb-1.0-0-dev libgsm1 libgsm1-dev cmake \
automake build-essential

svn checkout http://yate.null.ro/svn/yate/trunk yate
svn checkout http://voip.null.ro/svn/yatebts/trunk yatebts

cd ./yate
./autogen.sh
./configure --prefix=/usr/local
sudo make install
sudo ldconfig

cd ../yatebts
./autogen.sh
./configure --prefix=/usr/local
sudo make install
sudo ldconfig
```

Troubleshooting

- `make: *** [install-api] Error 1`. You do not have the `kdoc` and `doxygen` package. These are necessary for documentation and can be skipped by using `make install-noapi` instead.

- `MSInfo.cpp:641:86: error: no match for operator<< ...` You have to download version 5 of the `g++` compiler:

```
sudo install g++-5
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-5 1
sudo update-alternatives --config g++
```

  Do not forget to switch back to the default `g++` compiler after installation.

Also see [59] and [28].

B.5 Reverse Geolocation of Cell Towers

The following script is written in Python 2. It returns a sorted (by distance) list of cell towers around a given address for a certain radius in km. The OpenCelliD database [57] was used, with all entries stripped out except for GSM towers located in the Netherlands (MCC 204).

```
#!/usr/bin/python
# be sure to pip install tabulate geocoder geopy
import csv, sys, math, tabulate, geocoder
from geopy.distance import great_circle

if (len(sys.argv) != 4):
    print "incorrect amount of arguments:\nusage: python script.py [csv-database
    ] [address] [radius in km]"
    sys.exit(1)

latlon = geocoder.google(sys.argv[2]).latlng
if (latlon is None):
    print "no response: retry or the given address was not found"
    sys.exit(1)

with open(sys.argv[1], 'rb') as csvfile:
    out = []
    reader = csv.reader(csvfile, delimiter=',')
    next(reader, None)
    for row in reader:
        distance = great_circle(latlon, (float(row[7]), float(row[6]))).km
        if (distance < int(sys.argv[3])):
            out.append([distance, int(row[1]), int(row[2]), int(row[3]), int(row
    [4]), float(row[6]), float(row[7])])
    out = sorted(out, key=lambda l:l[0])
    headers = ["Distance", "MCC", "MNC", "LAC", "CellID", "lon", "lat"]
    print tabulate.tabulate(out, headers, showindex="always")
```

Appendix C

Intercept

C.1 Using Scapy and NetfilterQueue

This interception method requires an iptables rule that places packets in a NetfilterQueue. An example of this can be found below. The bytes of the packets can be modified accordingly, this is left to the user to specify.

```
sudo iptables -t mangle -A PREROUTING -i sgsntun -j NFQUEUE --queue-num 1
```

```
#! /usr/bin/python3
# be sure to pip3 install scapy netfilterqueue
from scapy.all import *
from netfilterqueue import NetfilterQueue

def modify(packet):
    pkt = IP(packet.get_payload()) #converts the raw packet to a scapy
    compatible string
    print(pkt)

    # possible to modify the packet
    # packet.set_payload(bytes(pkt))

    packet.accept() #accept the packet

nfqueue = NetfilterQueue()

# 1 is the NFQUEUE number
nfqueue.bind(1, modify)
try:
    print("[*] waiting for data")
    nfqueue.run()
except KeyboardInterrupt:
    nfqueue.unbind()
    pass
```

C.2 SSL-to-SSL Proxy

The following commands can be used to redirect `tcp` traffic on port `443` from interface `sgsntun` to `localhost:8085`.

```
#!/bin/bash
# retrieve the local ip, e.g. 10.42.0.164
addr=$(ip addr show eth0 | grep 'inet ' | cut -d/ -f1 | awk '{print $2}')

# iptables rule
iptables -t nat -A PREROUTING -i sgsntun -p tcp --dport 443 -j DNAT --to \
 $addr:8085
```

The following commands generates a server certificate that we sign ourselves (this requires some user interaction).

```
#!/bin/bash
filename=server

openssl genrsa -out $filename.key 1024
openssl req -new -key $filename.key -x509 -days 3653 -out $filename.crt
cat $filename.key $filename.crt > $filename.pem
```

The following processes must be started separately. The first process listens on `localhost:8085` with certificate `server.pem`, then redirects the traffic to `localhost:6500`. The second process listens on `localhost:6500` and redirects the traffic to ip `172.217.17.36:443`. Note that we configure the ip and port of the target service, hence the proxy is not dynamic.

```
# first process
socat -v openssl-listen:8085,cert=certs/server/server.pem,verify=0,reuseaddr, \
 fork tcp4:localhost:6500

# second process
socat -v tcp4-listen:6500,reuseaddr,fork ssl:172.217.17.36:443,verify=0
```