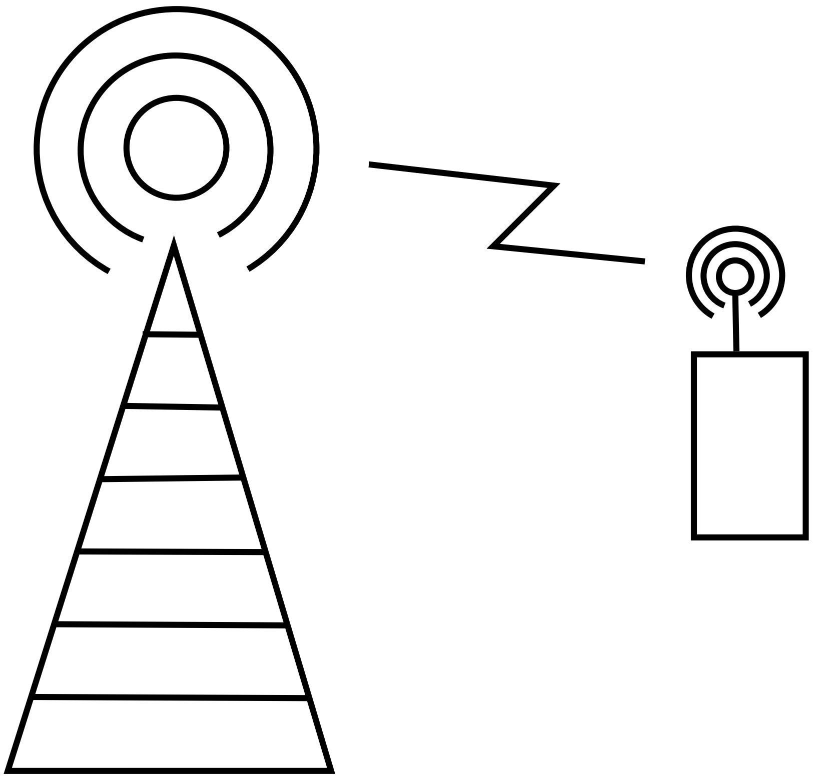
RADBOUD UNIVERSITY NIJMEGEN

Master's Thesis Computer Science

Fuzzing the GSM Protocol

by Brinio Hond



Thesis number: 654
Date: July 29, 2011

Supervisors:
ir. Stan Hegt[a]
ir. Ronald Heil[b]
dr.ir. Erik Poll[c]

[a]Advisor ICT Security & Control at KPMG
[b]Manager ICT Security & Control at KPMG
[c]Associate Professor Digital Security at Radboud University Nijmegen

Abstract

In our current society GSM can be considered a critical infrastructure as it is used by over 3.5 billion people worldwide. And even though the protocol is already over twenty years old most serious scrutiny on it stems from the last couple of years. This is due to the rapid evolution of Software Defined Radio (SDR) five years ago, which allowed most of the signal processing to take place in software instead of hardware. Several open source projects emerged that used the principle of SDR to implement a GSM stack in software using relatively cheap radio hardware.

In this thesis one part of the security of GSM is analysed using an SDR based on the open source project OpenBTS and a hardware device called USRP-1. With a technique called protocol fuzzing the robustness of the implementation of the GSM protocol on different cell phones is tested. In this thesis it is first described which parts of the protocol stack are most suitable for fuzzing, as well as which fields in GSM communication are most likely to result in strange behaviour on the receiving end. Then this theory is put to the test and two parts of the GSM stack (SMS and Call Control) are fuzzed on actual cell phones. Using a test set of just over 900 messages seven out of sixteen tested phones could be caused to reboot remotely. On two smart phones it was possible to cause a Denial-of-Service on receiving SMS messages until the phone was switched off and back on. Even though the discovered attacks can not be sent over real, commercial networks they serve to illustrate how vulnerable most phones are.

Acknowledgements

First and foremost I would like to thank my supervisors Erik, Stan and Ronald. Thank you for all the time you put into helping me with problems I encountered, giving me suggestions to improve both my research and this document and providing me with feedback regarding everything I wrote.

My gratitude also goes out to Fabian van den Broek and Ronny Wichers Schreur, who built the fake basestation at the university and helped me out with any technical problems on the way.

I also want to thank those that lent or donated me phones for this research. Bram, Erik, Fabian, Ineke, Pieter, Ronald, Ronald, Ronny, William and others from ICIS whom I missed in this list, thank you all. My thanks also go out to Lejla for watching over the key.

Last but not least I would like to thank everyone from KPMG team ISC for the great time I had there during my graduation. Their openness, professionalism and great social skills made the time fly every day again. I had such a blast that I took the opportunity to work there after graduation with both hands.

Contents

Chapter 1

Introduction

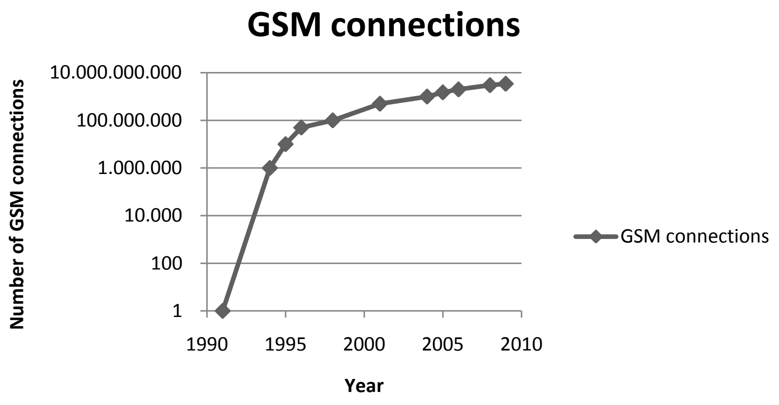
Hacking the Global System for Mobile Communication (GSM) is hot. Although the protocol was already published in 1990 by the European Telecommunications Standards Institute (ETSI) [1] most attacks on GSM date from the last few years. This is strange, especially considering the fact that GSM has grown incredibly fast. The first GSM network was online in 1991 and in 1994 GSM already had one million users. In 2004 GSM over one billion people used GSM, with the latest numbers being nearly three and a half billion users worldwide in 2009 (see Figure 1.1). For reference, the number of Internet users was only just over two billion in March 2011 [2].

One of the main reasons GSM was mostly left alone by security researchers is because it was simply too expensive to attack practically. Third party equipment that is accurate enough for GSM communication was expensive, while it was unknown how to use equipment designed for GSM communication (i.e. cell phones) in ways not specifically allowed by the vendors (e.g. sniffing or sending custom data). In addition people were unfamiliar with the protocol. The GSM protocol specification consists of nearly two thousand documents, each being anywhere from a dozen to over six hundred pages long. And last but not least the GSM implementations on nearly all cell phones were (and still are) closed source, so these could not be used to learn more about the protocol.

Of course the above problems did not stop all researchers and some flaws in different parts of the GSM protocol were revealed as early as the end of the 90's. Two of the most significant flaws of GSM that were already discovered back then are the stream cipher that is still used in most GSM networks (called A5/1) and that only the cell phone is authenticated when connecting to a GSM network while the network is not. However, practical attacks to exploit these issues were not found at that time.

In the last couple of years Software Defined Radio (SDR) started to emerge, which allows a large part of the signal processing involved in radio communication to be handled in software instead of hardware. This not only allowed software experts to enter the world of radio communication, but also made the equipment with sufficient accuracy that is required for GSM traffic a lot cheaper. As a result several open source projects have emerged to implement parts of the GSM network based on the principle of SDRs. These community efforts provide a lot of insight into the GSM protocol and have also enabled researchers to discover and implement practical attacks on GSM, although that was not the

Figure 1.1 Development of GSM connections over the years. Data taken from the GSM Association [3, 4].



main purpose of these projects.

There are two independent open source SDR projects that both aim to be a complete GSM network: OpenBTS [5] and OpenBSC [6]. These projects allow anyone to run their own GSM network for calls and Short Message Service (SMS) on relatively cheap SDR devices (starting at around €1500), but can be also used for GSM security reviews. One possible attack is to mount a man-in-the-middle attack against a real network to control all communication between the cell phone and the network and attack confidentiality, integrity and availability. This is possible due to the lack of mutual authentication in GSM. Besides these two GSM network implementations there is also an open source solution for certain cell phones called OsmocomBB [7]. OsmocomBB can completely replace the commercial closed source firmware and thus allows the user to have a cell phone running open source software only. Besides this goal OsmocomBB has also been successfully used to implement an attack to sniff on GSM communication. Using a few cheap cell phones with OsmocomBB, a desktop computer and rainbow tables of two terabytes in size it is possible to crack the session key of the A5/1 cipher in less than five seconds [8], subverting the confidentiality of GSM.

The above attacks focus on weaknesses in the GSM protocol itself, but there is another point at which security issues can be introduced in the GSM equipment: the implementation of the protocol. The specification with its tens of thousands of pages is hopelessly large and some parts are very complex as well. This large pile of paper describes many different features of GSM, with a large part of those features unknown by most and barely ever used. An example of these obscure features of GSM is sending a fax or email over Short Message Service (SMS). Because of the sheer amount of features and room for future extensions in all aspects of GSM it is highly likely programming errors occurred in the implementation, which can be exploited by an attacker for Denial-of-Service (DoS) attacks or possibly even worse (for example remote code execution in case of a buffer overflow).

The focus of this research is exactly these programming errors introduced when implementing the protocol specifications. Through the use of a technique called protocol fuzzing programming errors in GSM protocol implementations in commercial cell phone brands can be identified, giving an indication of the robustness of GSM implementations. This research describes how this protocol fuzzing can be done on GSM, from which hardware and software is required to do this, up to which parts of the protocol are most interesting to fuzz and how this can be done both efficiently and thoroughly. These ideas are then put to the test using a fake basestation, made with a USRP-1 in combination with OpenBTS, to fuzz sixteen different cell phones on parts of the SMS protocol and Call Control (CC) service. A total of just over 900 test cases generated by us were executed on all phones and with this test set we could induce a remote reboot on seven and an SMS DoS on two phones.

The remainder of this chapter is organised as follows. First related work regarding protocol fuzzing attacks on the GSM protocol is described, followed by a section on the contributions and scope. In the third section the research methodology is described. Section 4 discusses the relevance of this research and the final section of this chapter describes the organisation of the rest of this thesis.

1.1 Related Work

Due to the popularity of GSM has become a popular topic for research now that equipment is affordable and widely available. Many different aspects of GSM have been scrutinised, from protocol specifications and encryption algorithms to implementations of certain features on the handsets and in the network. This research focuses on the implementation of GSM, but is not the first to do so.

Two basic attacks on the robustness of an implementation exist, one based on source code review and the other based on testing. Because most cell phone Operating Systems (OSs) are closed source the former is difficult to realise and, to the best of our knowledge, has not been performed on commercial cell phones by independent researchers. The attacks taking the latter approach were mostly fuzzing attacks, since that is a well-known way to test robustness of software (see also Chapter 3).

Protocol fuzzing on GSM has a problem that has to be overcome. Making calls or sending SMS messages over a commercial network costs a fee. The fee is small for a single call or message, but protocol fuzzing usually requires a large test set making it add up very rapidly. As a result all previous research on GSM protocol fuzzing used some way to get the messages to the phone without having to pay a fee, either by using a private network or otherwise.

One of the first robustness tests of GSM dates from 2006. Mulliner and Vigna used protocol fuzzing on the Multimedia Messaging Service (MMS) feature of GSM [9]. MMS is an extension to SMS for the exchange of multimedia content. When an MMS message is sent the recipient receives an SMS message with a Uniform Resource Identifier (URI) to a server where the MMS content can be retrieved from using the Wireless Application Protocol (WAP). Mulliner and Vigna built a virtual (malicious) MMS server using open source software and retrieved content from it on different cell phones. They found several weaknesses in various implementations, among others buffer overflows in the Synchronised

Multimedia Integration Language (SMIL) parser, the part that takes care of the presentation of the content on the cell phone to the user. Some of these buffer overflows could be used for arbitrary code execution, which resulted in a proof of concept of "... the first [exploit] to perform a remote code execution attack against a mobile phone using an MMS message as the attack vector" [9].

In 2009 Welte presented three ways how one could perform protocol fuzzing on GSM [10]. The first approach is to use a cell phone to fuzz the network. This requires full access to the radio part of the GSM phone, which was not yet possible at that time (but is now with the OsmocomBB project). The second approach uses a rogue network endpoint (called basestation) to test the cell phone, which can be achieved with OpenBTS or OpenBSC. This is also the strategy of choice for this research. The last approach works by using a proxy between the basestation and the rest of the network to inject messages to test both cell phone and the network. This could be done using a combination of Scapy[1], an interactive packet manipulation program, and OpenBSC.

Another target of earlier GSM protocol fuzzing has been SMS. Mulliner and Miller targeted SMS on smart phones in 2009 [11] and Mulliner and Golde targeted the same feature on feature phones in 2010 [12]. The approach in these two researches was quite different. To fuzz the smart phones Mulliner and Miller built a custom injection framework to inject messages on a software level in the phone itself[2]. They found several DoS attacks for three popular smartphone OSs: iOS, Android and Windows Mobile. Mulliner and Golde used the second approach described by Welte: a rogue basestation based on OpenBSC. Furthermore they used a J2ME[3] application for monitoring on the cell phones. They found DoS attacks for six different popular feature phone brands using SMS messages that can even be sent over commercial (real) networks. After consultation with the phone manufacturers Mulliner and Golde did not publish the actual messages that cause the DoS.

1.2 Contributions and scope

1.2.1 Contributions

This research presents a way to use protocol fuzzing on GSM enabled devices (i.e. cell phones) using a fake basestation made with a USRP-1[4]. The USRP-1 made it possible to send arbitrary messages using OpenBTS from a notebook hooked up to the Universal Software Radio Peripheral (USRP) to any connected phone. The research question that formed the basis for this research is the following:

> On which parts of the GSM protocol can protocol fuzzing be used to cause unexpected behaviour of a Mobile Station?

The following contributions are made in this research:

[1]`http://www.secdev.org/projects/scapy/`

[2]Note that this was an approach not foreseen by Welte in his presentation.

[3]Java 2 Platform, Micro Edition; a Java platform designed for embedded systems like mobile devices.

[4]`http://www.ettus.com`

- It is described which layers, sublayers and services of the GSM protocol are interesting targets for protocol fuzzing and how this could be done (Chapter 4).
- The messages for two of these services (SMS and CC) are described in detail, including pointers to which fields in messages are most likely to trigger unexpected behaviour (Chapter 4 and Appendices D, E and G).
- A working proof of concept was built for easy creation of GSM protocol fuzz test cases that fuzz the aforementioned fields that are most interesting.
- We are, to the best of our knowledge, the first to use OpenBTS for GSM protocol fuzzing and extended OpenBTS version 2.6.0 to allow fuzzing[5].
- We generated just over nine hundred test cases for a practical analysis of sixteen cell phones. For nine of the sixteen phones strange behaviour on certain messages was found and the fields that cause this are discussed. Most of these messages can only be sent when controlling the basestation and are thus academical in nature. However, they serve to illustrate the potential of these kind of attacks and how much effort cell phone manufacturers put into robustness of their products.

1.2.2 Scope

In this research protocol fuzzing is only used against cell phones. There were two reasons why it was not used against networks:

1. Commercial basestations are not our property and attempting to hack or disrupt them without permission is unlikely to be allowed[6]. It was also deemed unlikely that GSM providers would grant us permission to find weaknesses (that would get published) in their commercially used basestations.
2. OpenBTS and OpenBSC are not based on implementations of real commercial basestations, thus fuzzing them would not grant us insight in how a real basestation would respond to invalid messages. In addition they are still under development and at least OpenBTS does not handle most encountered errors gracefully (i.e. simply shuts down with an assertion failure).

Another scoping decision was that of which layers of the SMS protocol are fuzzed. In [12] the Short Message Application Layer (SM-AL) was fuzzed extensively, so we did not fuzz this SMS layer.

Last but not least for the practical part of this research we were dependent on donations of cell phones, otherwise it would cost too much. As a result mostly older types of phones could be tested, but a couple of smart phones are included as well. Even though older, 'dumb' phones are not used often anymore,

[5]We had access to a USRP-1, which is only supported by OpenBTS.

[6]We did not look deeply into the legal issues this would cause, but our guess is that it is not forbidden by criminal law, but could be covered by the Terms and Conditions of the telecom provider.

they pose the biggest threat. Most of these phones can not have their firmware updated. And even if they can, they have no means to notify the user that an update is ready. In addition performing an update requires the user to hook up the phone to a computer to perform the update. This means that the problems are unlikely to ever be fixed on all older phones. Smart phones on the other hand can often be updated more easily, either with over-the-air update methods or simply using the Internet connection.

1.3 Research methodology

- Literature study. We try find out as much as possible about GSM, protocol fuzzing and the combination of the two. For the GSM standards the most recent release (at the start of this research in February 2011) was used. Even though GSM is not explicitly named in the title of this release, it still contains information to which GSM should comply as well[7].
- Proof of concept. We build a proof of concept implementation for a program to generate fuzz tests. In addition we extend the current OpenBTS version (2.6.0) with methods to send GSM arbitrary Layer 3 messages specified in raw hexadecimal format.

1.4 Relevance

GSM is a critical infrastructure in many countries and downtime of a network or user can result in financial losses or even worse. Previous work has shown that the resilience of cell phones is nothing to write home about and it is likely that the resilience of networks is just as bad. The tools to test both parts of the infrastructure are within reach of researchers and thus also of attackers. As a result it is important that issues are identified and fixed before a malicious person or organisation exploits them on a large scale.

Despite the introduction of the newer and more secure 3G and 4G protocols for mobile communication GSM is still widely used. In Q2 of 2009 there were almost 3.5 billion GSM connections against only just over 0.5 billion 3G connections [4]. In addition few phones offer an option to turn usage of GSM off completely, which would be the only way to prevent fuzzing attacks on GSM (since most do not require user interaction from the receiver). And even then attacks on SMS could still occur, because SMS is a separate protocol and SMS messages can also be sent over 3G and 4G networks.

1.5 Outline

This thesis is split up in two parts. The main contributions and results are conveyed in the chapters, while the technical details of this research are discussed in the appendices. The chapters are organised as follows.

Chapter 2 is a short introduction to GSM with an overview of the network and the protocol stack. It provides a basic understanding of the part of a network we attack and what layers, sublayers and services can be distinguished

[7]See `http://www.3gpp.org/Specification-Numbering`.

within the GSM protocol. Chapter 3 is an introduction to protocol fuzzing and what protocol fuzzing strategies exist. In Chapter 4 it is described which protocol (sub)layers and services of GSM are interesting targets for fuzzing and which of the introduced protocol fuzzing strategy was chosen for this research. Chapter 5 contains the results of protocol fuzzing on sixteen cell phones with six different types of unexpected behaviour found on nine different phones. Chapter 6 describes future work and this thesis is concluded in Chapter 7.

Chapter 2

Introduction to GSM

This first chapter is dedicated to the Global System for Mobile Communication (GSM) network infrastructure and protocol stack. This chapter is a only short introduction to GSM, more details can be found in Appendix A.

In the first section of this chapter an overview is given of the GSM network infrastructure. The different subsystems of GSM are briefly discussed, as well as some of the entities within the network. In the second section a short description of the air interface between a cell phone and the network is described. Appendix A contains a more thorough description of this air interface, as well as a description of the messages that are transmitted over the air on Layer 3 of the GSM protocol stack. These messages will be the target of fuzzing in this research, as will be discussed in Chapter 4.

2.1 GSM network infrastructure overview

A GSM network (officially called a Public Land Mobile Network (PLMN)) is a complex infrastructure with many different entities. Each network operator has its own PLMN in an area and a single PLMN can span an entire country. The entities in a PLMN are spread over three subsystems, which are called:

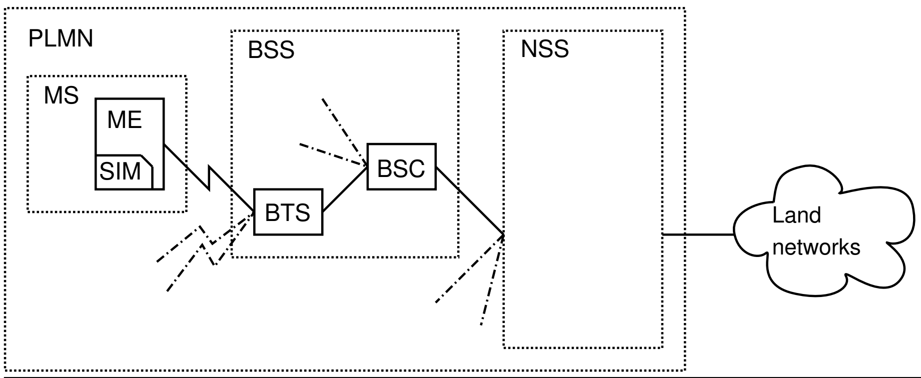
1. Mobile Station (MS)
2. Base Station Subsystem (BSS)
3. Network Switching Subsystem (NSS).

An overview of these subsystems and some of the entities within the subsystems is given in Figure 2.1. Only the parts relevant for this research are shown and described in this section. For more information about the PLMN infrastructure see [13, 14].

2.1.1 Mobile Station (MS)

The Mobile Station (MS) is what most GSM users are familiar with, since it is the phone they hold in their hand. An MS is actually the combination of two parts:

Figure 2.1 Overview of a Public Land Mobile Network (PLMN).



1. Mobile Equipment (ME)
2. Subscriber Identity Module (SIM).

The Mobile Equipment (ME) is the cellular phone that is used for incoming and outgoing calls. The Subscriber Identity Module (SIM) is a small smart card supplied by the GSM service provider which fits in the ME. The SIM contains, among others, a (globally) unique identifier called International Mobile Subscriber Identity (IMSI) that identifies that particular SIM/MS[8]. The SIM also contains information about the default frequencies used by the provider that issued the SIM and a symmetric encryption key shared with the provider that it can use to authenticate to the network and to derive a session key to encrypt part of the GSM signal travel path (see [15, 14]). Evidently there are many MSs in a single PLMN.

2.1.2 Base Station Subsystem (BSS)

The Base Station Subsystem (BSS) is the part of the PLMN that connects to MSs over the air on one end and to the NSS (usually over land) on the other end. The main purpose of each BSS in a PLMN is exactly connecting the other two subsystems; it uses several many-to-one interfaces to combine the traffic of a lot of different MSs that was sent over the air to a single connection to the NSS. Most, if not all, PLMNs have multiple BSSs, each serving their own region. Within a BSS two different entities exist:

1. Base Transceiver Station (BTS)
2. Base Station Controller (BSC).

Base Transceiver Station (BTS)

Base Transceiver Station (BTS) is the name for a transceiver on a cell tower. The BTS is the entity that an MS connects to and is thus the first hop in GSM

[8]It is important to note that the IMSI is not the same as the phone number (officially called Mobile Subscriber Integrated Services Digital Network number (MSISDN)). These numbers are linked by the provider, allowing a user to switch provider (which requires switching the SIM) but keeping the same phone number.

traffic originating from the MS (called uplink) and the last hop in GSM traffic terminating at the MS (called downlink).

BTSs are connected over the air with MSs on one hand and with a BSC over 'land' on the other[9]. The BTS has to take care of all the details of the radio (air) interface with the MS, like frequencies, voice encoding/decoding (for compression), multiplexing and ciphering. The air interface between the MSs and BTS is officially called Um interface and is further described in Section 2.2.

Base Station Controller (BSC)

The Base Station Controller (BSC) is the heart of a BSS and usually serves many BTSs at the same time. It is concerned with the radio channel setup and handovers of MSs between BTSs connected to it [14]. Each BSS has exactly one BSC. Since this entity is not relevant for this research, it will not be discussed here further.

2.1.3 Network Switching Subsystem (NSS)

The Network Switching Subsystem (NSS) is the core of a PLMN and consequently every PLMN has exactly one NSS. An NSS contains many different entities, but these are not relevant for this research and therefore will not be described here. For further reading on the NSS see [13, 14].

2.2 Um interface

In this section the radio interface between the MS and BTS is described: the Um interface. The name of this interface was picked because it is very similar to the U interface between the network and a client in a land network (officially called Public Switched Telephone Network (PSTN), see [16]), but adapted for *m*obile communication. The Um interface is based on the bottom three layers of the Open Systems Interconnection (OSI) reference model [17] and shown in Figure 2.2. From bottom to top the layers for GSM are:

1. the Physical Layer
2. the Data Link Layer
3. Layer 3[10].

Each of these layers takes care of a part of the connection between an MS and BTS. The following subsections give an overview of the three layers.

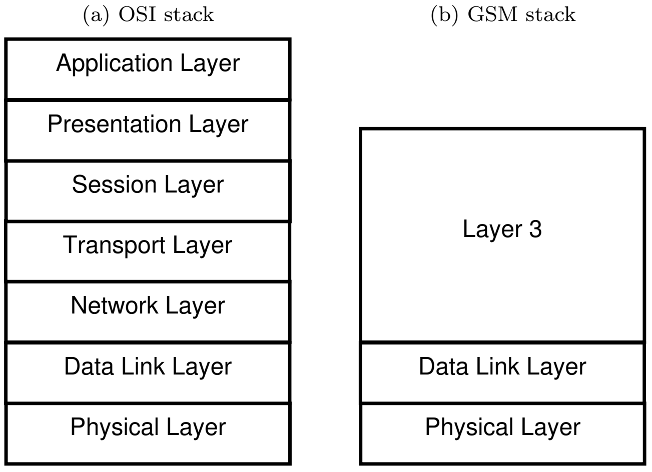
2.2.1 Physical Layer

The lowest layer of the Um interface is the Physical Layer, which is specified in [18]. This layer serves as an abstraction of the radio communication for the

[9]Land should not be taken literally, sometimes a microwave directional antenna is used for this connection [14].

[10]In the Open Systems Interconnection model the third layer is called the Network Layer, but in the GSM standards it is called Layer 3, because it is much more than only a Network Layer.

Figure 2.2 The OSI and GSM protocol layer stacks.



higher level layers. It offers several channels to the Data Link Layer on which data can be transmitted. The Data Link Layer then only has to specify which channel and which bits have to be transmitted and the Physical Layer will ensure that this happens, using error detection/correction and also encryption when necessary.

A channel in the Physical Layer has two important parameters: frequency and timing. To make optimal use of the available bandwidth both parameters can be varied, so traffic of different users can be multiplexed together both in frequency and time. This is further discussed in Appendix A.1.1.

2.2.2 Data Link Layer

The second layer in the Um interface is the Data Link Layer. This layer is the interface between Layer 3 and the Physical Layer. Its main purpose is to offer two optional services to Layer 3, which are specified in [19]:

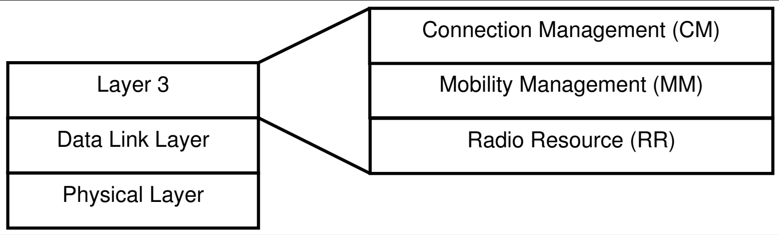
1. acknowledgements and retransmits
2. segmentation.

When acknowledgements are used at the Data Link Layer messages are retransmitted until the receiver acknowledges delivery. This guarantees to Layer 3 that a message will be delivered. Segmentation can be used to deliver Layer 3 messages that are too long to fit into a single Data Link Layer message, allowing Layer 3 to send messages of arbitrary length. Segmentation can only be used in combination with acknowledgements.

2.2.3 Layer 3

Layer 3 is the highest layer in the GSM stack. It utilises the services offered by the Data Link Layer (and thus indirectly also the Physical Layer) to perform its functions. Layer 3 itself consists of several sublayers, each offering their

Figure 2.3 Layer 3 of the GSM protocol consists of three sublayers.



own services and having their own responsibilities. The different sublayers are defined in [20] and depicted in Figure 2.3:

1. Radio Resource sublayer
2. Mobility Management sublayer
3. Connection Management sublayer.

Each of the sublayers are briefly described below, with a more detailed description in Appendix A.1.3.

Radio Resource (RR) sublayer

The Radio Resource (RR) sublayer is the bottom sublayer inside GSM Layer 3, which means it forms the basis for the rest of Layer 3. It has as main purpose the setting up and tearing down of logical channels for point-to-point communication between the network and a single MS. More on logical channels can be found in Appendix A.1.1.

Mobility Management (MM) sublayer

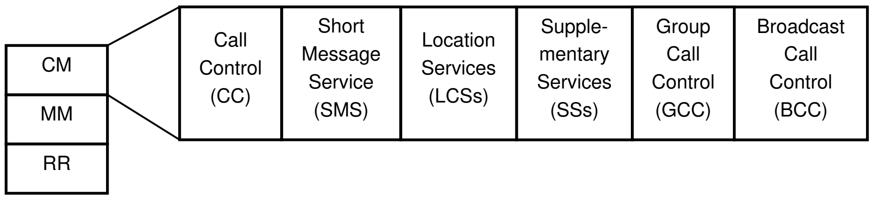
The Mobility Management (MM) sublayer is the sublayer that is used to make sure the network knows exactly in which cell an MS is at any one time. It defines several different messages that can be used to update the location information of an MS or connect an MS that is new in the PLMN. Authentication of the cell phone to the network is also carried out on this sublayer.

Connection Management (CM) sublayer

The Connection Management (CM) sublayer is the name for all services running on top of the MM sublayer. Six of these services are distinguished in [20] and depicted in Figure 2.4:

1. Call Control
2. Short Message Service
3. Location Services
4. Supplementary Services
5. Group Call Control

Figure 2.4 The six services distinguished in the Connection Management sub-layer.


CM
MM
RR
Call Control (CC)
Short Message Service (SMS)
Location Services (LCSs)
Supple-mentary Services (SSs)
Group Call Control (GCC)
Broadcast Call Control (BCC)

6. Broadcast Call Control.

Each of these services are briefly discussed below.

- Call Control (CC) is the service GSM was originally designed for: making and receiving phone calls. CC manages everything related to phone calls, from call establishment to call tear down and everything in between. It allows the user to make, accept and hold calls. The specifics of CC can be found in [21].
- The Short Message Service (SMS) was added shortly after the initial release of GSM and the first SMS message was sent in 1992 [3]. The first version of SMS allowed the exchange of short text messages between GSM users, but SMS has gone a long way since then. Not only can SMS be used to exchange text messages, but at this time pictures, sounds and many other types of data can be sent over the SMS. The current SMS standards also allow segmentation of messages that are too long to fit into a single message, enabling users to transmit much longer messages. Finally SMS is no longer only linked to GSM, but is also available for the more recent protocols 3G and 4G protocols. The current SMS specification is found in [22, 23].
- A GSM user can subscribe to Location Services (LCSs) to receive interesting services based on his/her current location. To this extent applications to which the user is subscribed can request a more accurate estimate of the location of the MS and use this information to give localised services, for example weather forecasts or directions to the nearest bank. These services are run by a Serving Mobile Location Centre (SMLC), which is an optional entity in the network. Upon activation of an LCS the MS can determine its location more accurately using the Global Positioning System (GPS) for the location estimation (if available in the cell phone) or by measuring the distance to different BTSs in the area (using timing measurements). LCSs are specified in [24].
- Supplementary Services (SSs) are additional services optionally offered by GSM providers. There are just over a dozen SSs currently specified. Among others SSs give the possibility to show the phone number (MSISDN) of the caller, block incoming or outgoing calls or temporary put calls on hold. The SSs are split up in two categories, those that are call related and those that are not. The call related SSs, like showing the phone number of the caller, are officially integrated into the CC service. An exhaustive list of all SSs is given in [25].

- Group Call Control (GCC) is an optional feature for both the network and the MS. When implemented it allows a group of subscribers to make a group call when they are together in an area (cell or PLMN). Among others the GCC sublayer takes care of handovers of active participants of the group call. GCC is described in [26].
- Broadcast Call Control (BCC) is another optional feature and very similar to GCC. The only difference is that with BCC only a single user can speak, while the other subscribers can only listen. BCC is described in [27].

Chapter 3

Introduction to protocol fuzz testing

This chapter will give a short introduction to protocol fuzz testing (or protocol fuzzing for short). Protocol fuzzing is the testing of a protocol implementation for robustness, i.e. testing how well it copes with invalid or unexpected input. In other words it tests whether a program accepts input it should not accept and fails while processing it. A typical example of a robustness issue is a buffer overflow, meaning the program stores input that is too long while it should either truncate the input, process it in smaller parts or report an error before storing it.

In this chapter a brief overview of protocol fuzzing is given. Appendix B holds more details regarding protocol fuzzing: some history on protocol fuzz testing as well as fuzz testing in general, several techniques that exist for fuzz testing and which tools can be used to assist in protocol fuzzing. This chapter will start with an explanation of protocol fuzz testing and then describe which three main strategies of protocol fuzzing exist. In Chapter 4 the strategy chosen for this research is discussed.

3.1 What is protocol fuzz testing?

Protocol fuzz testing is a fairly new technique to test the robustness of a protocol implementation. It is important to note that protocol fuzz testing tests the implementation; protocol fuzz testing can not be used to directly test the protocol specification itself. However, it is always possible that a problem in an implementation is related to a more fundamental issue with the protocol specifications.

The way protocol fuzzing works is actually quite simple: feed a protocol implementation input that does not conform to the specifications and see whether the implementation behaves correctly, e.g. does not hang or crash. Example protocol fuzzing input is a length field specifying the wrong length for an Internet Protocol (IP) packet or starting with the non-existent request method FOOBAR in a HyperText Transfer Protocol (HTTP) request line. Sometimes the specification might state how an implementation should respond to such malformed input, but even if it does not specify any behaviour the implementation should gracefully handle those input values, for example by ignoring them or returning an error message to the sender. Misbehaviour of the implementation on invalid input could be used for DoS attacks (e.g. making a web application crash thus rendering it unreachable for genuine users), or even worse, it might be exploitable for arbitrary command execution (e.g. in case of a buffer overflow).

There are two different parts of the implementation that can be fuzzed with protocol fuzzing, either separately or together. The most obvious part is fuzzing the content of the exchanged messages, like invalid types or wrong length fields. But there is another part that can often be fuzzed: the state machine. Most protocols have some sort of set sequences in which messages are exchanged and which messages are expected at any one time is tracked in a state machine. When the wrong message is sent at some point in time and still accepted by the implementation it shows a problem with its state machine. The impact of this is hard to estimate, because it depends on what states can be skipped, but for some protocols it might allow one to bypass authentication steps, posing a serious security risk.

3.2 Protocol fuzzing strategies

Protocol fuzzing has three different strategies, which of these can be chosen depends on the knowledge of two aspects of the protocol implementation. The first aspect that influences the strategy one can take is the knowledge of the protocol specification. Fuzz testing using the protocol specifications is called smart fuzzing, while fuzz testing without the specifications is called dumb fuzzing [28]. The second aspect is the knowledge of the source code of the implementation. The terms used to describe using the knowledge of the implementation source code is called white box, while not using said knowledge is called black box. Because knowledge of the source code usually implies also knowing the protocol (either because it is also available to the tester or because it can be deduced from the source code) white box fuzzing generally implies smart fuzzing.

Each of the three strategies has its advantages and disadvantages, therefore one should carefully decide which strategy to pick for a particular fuzz test. Of course when neither the protocol specification nor the implementation source code is known only a single strategy is possible. But if one or both of these aspects are known, using them is not necessarily a good thing. Below an overview of the three strategies is given.

- Dumb black box fuzzing, the simplest of the strategies that can be picked, but often also the least effective. Because neither the specifications nor the source code is known the tester knows essentially nothing. So the only option is to (pseudo-)randomly generate input and feed it to the implementation (called generation-based fuzzing). The advantage of this method is that input is incredibly easy to create and is easily portable between different implementations. The disadvantage comes from the fact that in many protocols only very specific message formats are accepted (for example only a handful of methods in an HTTP request header are valid) or only one specific value (for example when using a nonce). Randomly generated messages are unlikely to comply with these protocol rules, so the number of attempts required to proceed further down the message sequence quickly explodes.

- Smart black box fuzzing, a refinement of dumb black box fuzzing. With this strategy the protocol specification is used to create more intelligent test cases to penetrate deeper into the protocol. An often employed way of work with this strategy is to take a correct message and change different parts of the message one by one (mutation-based fuzzing). For example using the correct HTTP request line “HEAD / HTTP/1.1” and to first try “head / HTTP/1.1”, then “HEAD \ HTTP/1.1” and finally “HEAD / HTTP/1.2”.

  The advantage over dumb black box fuzzing is pretty clear from this example, the test cases can penetrate deeper into the protocol by first testing the parsing of earlier parts and then using correct values for those parts (in this case “HEAD” and “/”). On the other hand smart black box fuzz tests take more time to create then dumb black box fuzz tests, because each protocol rule has to be implemented and then parts have to be mutated. In addition the tester can (accidentally) make the same assumptions as the programmer, missing test cases because they are simply not tried. For example the above header might result in a buffer overflow if the length of the request method is more than 20 characters, regardless of actual characters in the header. A dumb fuzz test might have found this, while a smart one might not because the tester forgot to test very long request methods. The last disadvantage of smart black box fuzzing is that the resulting fuzz tests are only portable to other protocol implementations, but not to other protocols, unlike dumb black box fuzzing.

- White box fuzzing, the strategy that uses knowledge of both the protocol and source code to test as thoroughly as possible. By having access to the source code white box fuzzing theoretically allows one to find all robustness problems. Test cases can be made to test every possible execution path through the program and thus all erroneous paths can be discovered. Unfortunately this will often be infeasible in practise due to the size of the search space for larger code bases.

  Because testing every execution path is generally always infeasible the usual approach with white box protocol fuzzing is to first execute smart black box fuzz tests and then use the source code either to measure how much of the program code was actually covered by the tests or to monitor what happens exactly with each test. The former strategy is aimed at increasing the code coverage of the tests, i.e. how much of the code is actually tested. The higher the code coverage the more likely it is that an erroneous instruction is performed. The latter strategy on the other hand is aimed at finding even the slightest issues that do not show to the user, for example a very small memory leak.

  White box fuzzing has a big disadvantage, it takes a great investment in both time and effort to design all test input and execute them with the source code in hand. The more information that has to be implemented in the test cases the longer it takes to make them. In addition the resulting test set is only useful for that protocol implementation and is not portable at all.

Chapter 4

Considerations on GSM protocol fuzzing

In this chapter it is described for each GSM (sub)layer and service introduced in Chapter 2 how effective protocol fuzzing of that part would be. In addition we point out why SMS and CC, both at the CM sublayer of Layer 3 of the GSM protocol, are the target of protocol fuzzing in this research. At the end of this chapter the protocol fuzzing strategy and tooling used in the practical analysis of this research are described.

4.1 Fuzzing on the GSM (sub)layers

Unfortunately not all of the three layers, nor Layer 3 sublayers, are equally suitable for protocol fuzz testing, as is described in this chapter. The lower layers are more difficult to monitor, because nothing about what is happening is displayed on the screen. For some of the higher sublayers the interesting messages are only sent in mobile originating transactions, which often have to be started manually. This makes fuzzing those sublayers very time-consuming, because manually calling a number or sending an SMS message takes a lot of time compared to automatically doing this from a computer.

4.1.1 Fuzzing the Physical Layer

The Physical Layer of the GSM protocol is difficult to fuzz. Sending messages with the wrong timing or on the wrong frequency is relatively easy, but tracking what happens on the MS side is much more difficult, because usually nothing will ever appear on the screen. A hang or reboot can be seen quite easily, but smaller problems, like the MS being in a wrong state, can not. In addition it is very likely that the MS makes the hardware only listens to the correct frequencies at the correct time to safe as much battery power as possible, meaning that the fuzzed messages might not even reach the cell phone OS. But again that leaves the problem of how one can observe whether the radio is even listening at all.

4.1.2 Fuzzing the Data Link Layer

The second protocol layer is not very suitable for fuzz testing either. The acknowledgements and retransmits of messages are difficult to influence with OpenBTS without major time investments. And just like messages in the Physical Layer it is difficult to track what exactly happens inside the cell phone. Message segmentation offers a lot more options for fuzzing and is easier to monitor, but requires a lot of in-depth knowledge of the Data Link Layer. Due to the limited time that was available for this research this did not seem very efficient. However, it could be an interesting topic for further study.

4.1.3 Fuzzing Layer 3

The third and highest layer of the GSM protocol OSI stack is the most suitable layer for fuzz testing. It offers a lot of different services and monitoring is significantly easier than for the lower layers.

Fuzzing the Radio Resource (RR) sublayer

The RR sublayer has some interesting features that can be fuzzed, for example there are 75 different valid RR Message Types, all with their own content format and peculiarities. However, the RR sublayer suffers from the same problem as the Physical and Data Link layers: it is difficult to monitor unless something really bad happens. Testing whether a channel request is handled correctly and a channel is established is possible by sending genuine data over the channel. But what if it the genuine data is not received? Did the MS ignore the bogus RR message and does it therefore not receive the genuine message? Or did it allocate the wrong channel, which costs battery power? Or did something even more exotic happen? The difference is impossible to tell on most cell phones.

Fuzzing the Mobility Management (MM) sublayer

The MM sublayer only has a handful of tasks and most of them will prove difficult to fuzz. OpenBTS does not (yet) support authentication and ciphering, so fuzzing those messages is rather difficult without major changes to the code. In addition, one would hope that authentication and ciphering are more thoroughly tested for robustness, because errors there would make the phone completely unusable. So that leaves location updates and CM service control as the only remaining options for fuzzing on the MM sublayer.

MM messages for the control of CM services are only sent for CM services initiated by the MS. CM services initiated by the network do not explicitly use these MM messages, but instead either instruct the MS to set up an MM connection via paging or simply start sending CM messages to implicitly set up an MM connection. So in order to fuzz these messages one has to start a mobile originating CM transaction, but this is highly inefficient for the reasons discussed at the start of this chapter. Another option would be to start a mobile terminating CM transaction that forces the MS to set up an MM connection, but besides (optional) authentication and ciphering message exchanges only an acknowledgement (ACK) is sent, making the options for fuzzing close to zero.

The last function of the MM sublayer is the location update procedure. Just like the CM control messages activity regarding location updates is mostly initiated by the MS. It does happen automatically from time to time (unlike CM service control), but too infrequently for efficient fuzzing. This makes the entire MM sublayer difficult to fuzz and this is therefore not attempted in this research.

Fuzzing the Connection Management (CM) sublayer

The highest sublayer in the GSM protocol stack, the CM sublayer, is the most suitable (sub)layer for protocol fuzzing. The main reason (again) is that this highest sublayer is the easiest one to track, because you can be sure that the MS received the message and that it was parsed somewhere in the MS. When no ACK was received one can nonetheless be certain that the at least part of the intended message was parsed by the OS of the cell phone, unlike for example fuzzed RR messages. As a result all messages that are sent really test the GSM protocol implementation in the MS and none are lost in transit because the hardware was not tuned in to listen.

However, not all of the services in the CM sublayer are easily fuzzable. Below for each of the services possibilities for protocol fuzzing are described.

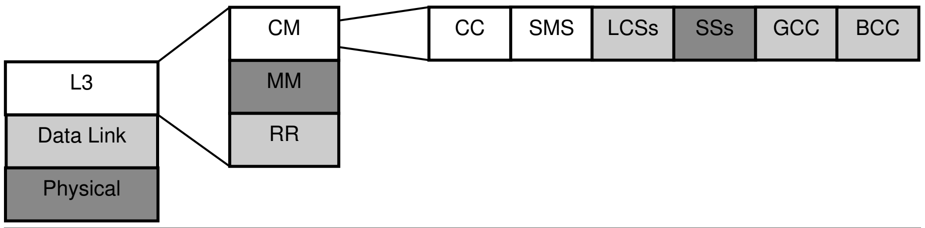
- Call Control (CC) is an interesting service from a fuzzing perspective. There are quite complex state machines defined for CC and at the same time many different types of messages exist. Of course there are still some downsides to fuzzing CC, e.g. many state transitions can not be influenced by the network in a mobile terminating transaction. For example the transitions from call received by the phone to alerting the user (i.e. ringing) to call accepted/rejected by the user all happen at the MS side and can not be altered from the network side. Still, this sublayer has a lot of options for fuzzing and was the second (sub)layer to be fuzzed in this research. More on the specifics of fuzzing of Call Control can be found in Appendix E.

- Short Message Service (SMS) is a great target for fuzzing attacks, especially for fuzzing of messages. Very few protocol states exist for SMS traffic, but this is made up for by the fairly large and complicated header included in SMS messages. This was the first (sub)layer being fuzzed for this research exactly for that reason. However, because some parts were already fuzzed in earlier research only lower layers of the SMS protocol stack were fuzzed in this research. More details on the SMS protocol stack and SMS fuzzing can be found in Appendix D.

- Location Services (LCSs) seem like a nice target to fuzz. Although the number of messages exchanged is limited to three, resulting in few protocol states, the messages do have many optional fields with a lot of possible values. Unfortunately the exact message structure is poorly described in the standards and we could not figure out how it worked[11]. In addition LCSs are not supported (yet) by OpenBTS, thus a large time investment would be required to implement LCS fuzzing. Because of these reasons

[11]This is of course very interesting from a fuzzing point of view, because it might mean the cell phone manufacturers did not understand it either, possibly resulting in a higher number of implementation errors.

Figure 4.1 Fuzzing of the different GSM protocol (sub)layers. White layers are fuzzed in this research, light grey are recommended for future research, while dark grey (sub)layers will prove difficult to fuzz.



fuzzing of LCSs was not attempted in this research, but would be nice to do in future work.

- Non-call related Supplementary Services (SSs) are the fourth type of CM service. SSs are rather hard to fuzz, because all but one are only used in mobile originating transactions. The only SS that is not initiated by the MS is "barring of incoming calls", which blocks all calls terminating at the MS. But when a call is blocked nothing gets reported to the MS, so no messages are exchanged at all making fuzzing those transactions impossible.
- Group Call Control (GCC) and Broadcast Call Control (BCC) are optional features in both the network and the MS. It is not supported by OpenBTS, but initial message exchange is similar to regular CC and hence should not be hard to add. However, both GCC and BCC are optional features of GSM and do not have to be implemented on the MSs. Fuzzing these services might prove to be a waste of time and was due to time constraints not attempted. Nonetheless it could be interesting in further research.

An overview of the evaluation in this section is given in Figure 4.1.

4.2 GSM protocol fuzzing strategy and tooling

Besides choices on which layers to fuzz another choice had to be made regarding the protocol fuzzing strategy. In this research cell phones of different brands will be tested and the OS of most cell phones is closed source. This makes a white box approach impossible. In addition even if the source code was available white box fuzzing would still be undesirable, because then for each implementation a different test set has to be created. This would simply cost too much time. The GSM protocol specifications on the other hand are publicly available, so smart protocol fuzzing is possible. To ensure that the phone actually parses the message, at least the lower layers of the protocol should be used correctly. And the first few fields in a messages determine the message type so these are probably the first to get parsed by the MS. Therefore to get the best results by being able to pass these checks black box smart fuzzing was the strategy of choice. The test cases can then be used on all different brands of cell phones and can target any desired part of the protocol with ease.

In this section the decisions regarding what aspects of SMS and CC are fuzzed more extensively than others are explained first. In the second part a short overview of tools is given that can aid with fuzzing, as well as what tools were used in this research.

4.2.1 Fuzzing SMS and CC

SMS and CC both have two messages sent from the network that can be fuzzed. For both the first message is complex with many fields that can be changed, while the second is a simple ACK with only three fields. Recall that besides fuzzing the content of these messages there is also the state machine that can be tested. The latter does require much more coding in OpenBTS, but can show interesting results as well. Below both of these options are explored.

Message content fuzzing

The number of possible invalid SMS or CC messages is too large to test exhaustively, which means concessions had to be made on which fields to fuzz more extensively than others. Luckily some fields are more likely to have insufficient error checking than others, making those the prime targets. In addition some fields are barely ever used in practise and thus received little real world testing. Those fields are very interesting targets for fuzzing as well.

Fields that specify the type of the message or the service the message belongs to are probably always verified, since these fields determine which part of the OS should parse the rest of the message. As a result extensively testing these fields is not very useful and it is more efficient to only test a handful of invalid or wrong values.

On the contrary fields that specify the length of a block in the message are much less likely to always be checked thoroughly. Using a very small or very large value for this field can easily trigger buffer overflows or similar issues if verification is insufficient. Also values that are just smaller or just larger than the actual length are also likely to cause a problem in an implementation. A different but similar approach is to leave the length field alone, but instead vary the size of the block it encodes the size of. Additionally most blocks have a minimum and maximum length specified in the GSM specifications. Deviating from these values is also more likely to result in strange behaviour.

Other interesting fields are those that are deprecated, reserved for future use or can be used as extension mechanism in future use. Changing these values might trigger undocumented or untested features.

Last but not least are the parts of the specifications that are barely ever used. An example was already given in Chapter 1: sending email or fax over SMS. Because these parts are not used often, they have barely been tested in the real world. This might also mean that it has a lower testing priority for cell phone manufacturers and thus potentially contains more bugs. Bugs in these parts might also exist for a longer time over generations of phones/firmware, simply because they were not yet discovered.

State machine fuzzing

Fuzzing the state machine in the phone requires more effort than fuzzing the content of messages. OpenBTS is made to send the messages in the correct order and on the correct times, so this is where changes have to be made to fuzz the state machine. It is possible to implement this generically, but that does take time.

Due to the time constraints of this thesis only state machine fuzzing of the SMS protocol was done and only in a limited way. The tests included both sending a correct message when it was not expected by the phone and sending a correct message when a different message was expected by the phone. Fuzzing the state machine for CC is more interesting because more states exist, but this was much harder to implement in OpenBTS. Therefore this option is left open for future research.

4.2.2 Protocol fuzzing tool

Several protocol fuzzing tools exist, for example SPIKE [29], Sulley [30] and mangleme [31], some of which are described in Appendix B.2. The tools range from simple fuzz test input generators or mutators for a single protocol, to more advanced fuzzers where the protocol can be specified by the user. Some fuzzers also include monitoring of the target implementation via Virtual Machines (VMs) and even automated restarting of the target in case of a crash.

Unfortunately none of these tools were ideal for GSM fuzzing. Advanced fuzzers with target monitoring were out of the question, because that would require VMs with all the cell phone OSs on it, which are difficult to get when the OSs are closed source. In addition one would also have to simulate the air interface over a different interface, something that might not work as expected. These two factors together made the use of advanced fuzzers very difficult, if not impossible.

But even simple fuzzers missed a couple of necessary features, mostly because they all seemed to work at the byte level. None of the tools that were tried supported bit level fuzzing, something which was required to be able to fuzz bit flags used in GSM. This would require a lot of recoding to add. In addition, some length fields in GSM do not count in octets (sequences of eight bits), but in septets (sequences of seven bits) or half-octets (sequences of four bits) instead. In the existing tools only counting in bytes was supported, but other granularities would be fairly easy to add by simply converting the current result. Other encountered problems with existing fuzzers included lack of support for writing output to files and lack of support to exclude certain values in tests. Because of these issues it was decided that building a tool from scratch, taking good features from established tools and adding the remaining required features at the same time, was the way to go for this research and a proof of concept was built in the Python programming language. The features of this proof of concept are described in Appendix C.

Chapter 5

Practical protocol fuzz testing on GSM cell phones

Part of this research was a practical analysis of the implementation of the GSM protocol stack on different cell phones. The cell phones we used for this were either lend or donated to us, so our selection was limited. However, interesting results could be found in the test set nonetheless.

In this chapter a summary of the results is given and the most interesting findings are highlighted. In Appendix G a more detailed description of the results is given, including the exact messages that causes them.

5.1 Scrutinised cell phones

A total of sixteen different cell phones have been tested for robustness using almost 550 SMS content fuzzing messages and over 350 CC content fuzzing messages on the message parts described in the previous chapter and also eleven SMS message sequence fuzzing tests. The SMS tests took on average one and a half hour per phone, while the CC tests took approximately one hour per phone.

Unfortunately we could not get an even distribution of different popular cell phone brands for our practical analysis, but different types of the same brand did give different results. In Figure 5.1 fifteen of the tested cell phones are displayed. Table 5.1 gives an overview of all the phones, their firmware versions and OS version if applicable[12]. For older phones the firmware version and OS version are basically one and the same, but on smartphones there is a clear difference (e.g. Android running on smartphones of different manufacturers).

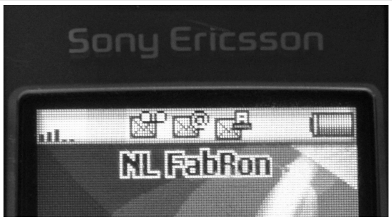
5.2 Summary of the results

Out of the sixteen tested cell phones nine had at least one message that triggered strange behaviour. Nearly all of the messages that caused this can only be sent when the attacker controls the BTS, because the headers involved are built by the BTS. However, the values might get copied from the message submitted to

[12]It is unclear to us at what level the CC and SMS stacks are implemented and whether the firmware or cell phone OS cause the observed behaviour.

Figure 5.1 Fifteen of the sixteen tested phones. Only the HTC Legend is missing.

Figure 5.2 Sony Ericsson T630 showing three icons for new types of messages, from left to right voice recording, email and fax.



the network by the sender, but this was not tested because we do not know how the network will respond to fuzzed messages (nor did we have the capabilities to easily send fuzzed messages from a phone).

In this section the discovered problems are summarised by type, from smallest impact to largest impact, and for each type the affected phones are listed. In Table 5.2 at the end of the chapter an overview is given of the problems for each phone. In Appendix G details about the exact fields that caused the issues can be found.

5.2.1 Icons

SMS offers the ability to notify a user that a voice, fax or email message is waiting to be retrieved. According to the specifications every cell phone has to show an icon on the screen when this happens. Problem is that these icons are hard to remove when they were activated illegitimately. Normally these icons are removed by another SMS sent from the network with flags set such that the phone will clear the icons. But these messages can not be forced from the network. The only way we found to remove them as a user is to insert a different SIM into the phone and power it on.

Even though this is not an actual security risk it can be quite annoying. The icons on most phones are not very clear, thus an unexpecting user might lose

Table 5.1 Cell phones tested in this research.

Brand	Type	Firmware version (Operating System)
Blackberry	9600	5.0.0.743
HTC	Legend	3.15.405.3 (Android 2.2)
iPhone	4	04.10.01 (iOS 4.3.3)
Nokia	1100	6.64
Nokia	2600	4.42
Nokia	3310	5.57
Nokia	3410	5.06
Nokia	6610	4.18
Nokia	6610	4.74
Nokia	7650	4.36
Nokia	E70-1	3.0633.09.04
Nokia	E71-1	110.07.127
Samsung	SGH-A800	A80XAVK3
Samsung	SGH-D500	D500CEED2
Samsung	Galaxy S	FROYO.XWJS5 (Android 2.2.1)
Sony Ericsson	T630	R7A011

a lot of time trying to figure out what these icons mean and even more time to get rid of them. This is unfortunately exactly how it is specified, but we nonetheless recommend an option for the user to remove the icons from the cell phone, even though this is not required by the specifications.

5.2.2 No notification

Five phones would accept invalid SMS messages and store them on the SIM or in the phone memory but not actually notify the user of the new message. As a result an attacker can silently fill up the phone memory (or in one case only the SIM) of the victim to temporarily make the user unreachable for SMS messages. Affected phones are both Nokia 6610s, the Samsung SGH-A800 and SGH-D500 (SIM card memory only) and the Sony Ericsson T630. All of these phones do notify the user when a message is received while the memory is full, so even when the memory is full the user will still get notified for new (genuine) messages and can clear the memory to receive it. The affected Nokia phones would already alert the user when 80% of their memory capacity was full.

5.2.3 Read memory

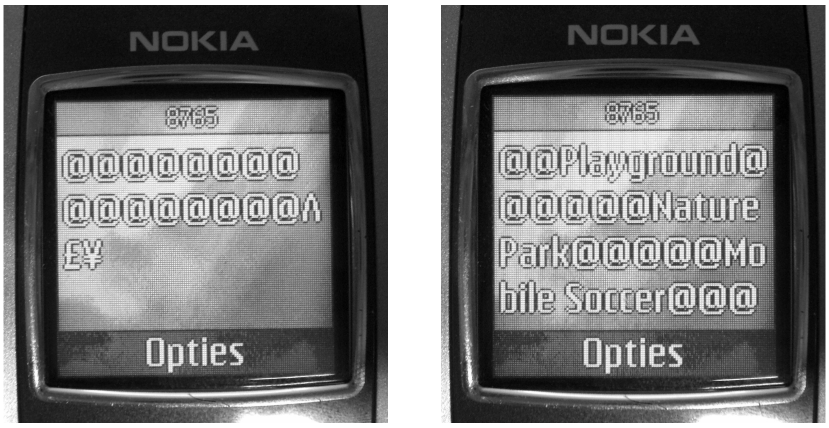
On two different phones it was possible to read out (part of) the phone memory. The most interesting of these phones was the Nokia 2600, where a text message would get stored that shows a seemingly random part of the phone memory upon opening (see Figure 5.3(a)). Closing and reopening of the same message would display a different part of the memory (Figure 5.3(b)), sometimes also causing a reboot of the phone.

On the Samsung SGH-D500 certain messages would show a strange sequence of characters when opened, but it was unclear to us where it came from. The same message would show up differently when sent multiple times, so we expect

Figure 5.3 The same SMS message on the Nokia 2600 opened twice showing part of the phone memory. The @ symbol in the GSM 7 bit default alphabet is encoded as only zeroes, explaining why it occurs quite often.

(a) Showing garbage

(b) Showing the name of a wallpaper and two games



it came somewhere from memory. The content shown in the SMS was actually also the content of the message (when opened on a different cell phone), unlike the issue on the Nokia 2600 where the displayed content was not stored in the message itself.

Problems like these can be quite dangerous. It is likely these issues are caused by a buffer overflow on some buffer, causing the following part of memory to be read. It might be possible to not just read this part, but actually overwrite parts as well. When this happens one is very close to remote, arbitrary code execution at the phone.

5.2.4 Reboot

Seven of the sixteen phones could be forced to reboot remotely. When rebooting the network connection would be lost temporarily. With two exceptions discussed below all phones would go back to standby mode and restore the network connection without asking for the Personal Identification Number (PIN) after the reboot, making these messages cause a short DoS. In all but two cases reboots were caused by a discrepancy between a length field and the actual length of that field in the message, making it likely that the behaviour is caused by a buffer overflow.

Two phones did, under certain circumstances, require the user to enter the SIM PIN again when being forced to reboot. The Nokia 2600 would sometimes reboot when receiving or opening the SMS messages that allows one to read parts of the phone memory (see above). When this happened the PIN would be required to use the phone again. The Samsung SGH-A800 on the other hand would reboot when it was not connected to a power outlet, but would shut down completely while charging. When this happened the phone had to be switched on again and the PIN had to be reentered.

5.2.5 Unable to delete messages

A rather annoying bug manifested itself on two cell phones, the Sony Ericsson T630 and Samsung SGH-D500. On the Sony Ericsson it was not too bad, but some messages could not be deleted when using the “delete all new messages” option. However, manually deleting them one by one or using “delete all messages” did work. On the affected Samsung phone certain messages would be stored as invisible messages. They could not be viewed or deleted in any way, but they still occupied space on the SIM. The only way to delete these messages was to put the SIM in a different phone and delete them there.

5.2.6 Long time DoS

For the iPhone 4 and HTC Legend the attack with the highest impact was found. By sending a carefully crafted SMS message the phone would not display anything and also stop receiving any SMS messages altogether. In addition on the iPhone it was impossible to change network after the attack, when trying to change network the phone would notify the user that it was unable to load the network list. Both smart phones returned to normal when turned off and on again or when out of range of our basestation[13].

[13]Because we were testing on a single BTS it was impossible to verify whether one has to go out of range of the BTS or of the entire network. This would be interesting to verify in future work.

Table 5.2 Summary of the discovered problems.

Phone	Protocol	Type of problem	Solution
All	SMS	Icons on screen	Insert other SIM
Blackberry			
HTC			
Legend	SMS	SMS DoS	Reboot/Change network
iPhone			
4	SMS	SMS DoS	Reboot
Nokia			
2600	SMS	Read memory	n/a
2600	SMS	Reboot	n/a
6610	SMS	No notification	n/a
6610	SMS	Reboot	n/a
7650	SMS	Reboot	n/a
Samsung			
SGH-A800	SMS	Reboot	n/a
SGH-A800	SMS	No notification	n/a
SGH-D500	SMS	Reboot	n/a
SGH-D500	SMS	Read memory	n/a
SGH-D500	SMS	Message deletion	Delete on other phone
SGH-D500	CC	Reboot	n/a
Sony Ericsson			
T630	SMS	No notification	n/a
T630	SMS	Message deletion	Delete one by one
T630	CC	Reboot	n/a

Chapter 6

Future Work

This research was just the start in extensive protocol fuzzing of all aspects of GSM. Together with some previous work many parts of the SMS protocol and the some parts of the CC service have been tested on several phones with protocol fuzzing. But many questions remained unanswered, both regarding the results found so far and regarding the untested parts. In this chapter several pointers for future work are given.

First of all protocol fuzzing can be carried out on different phones or the same phones with different firmware and OS versions. Interesting questions are whether certain attacks work across different firmware or phones. Another interesting idea is to try and combine attacks on different phones to a single attack that works on multiple types of phones at the same time. This would greatly increase the power of an attack and thus also the severity of the problems.

Another possibility is to fuzz other (sub)layers and services of GSM or other parts of SMS or CC. Especially the (sub)layers and services discussed in Chapter 4 are promising targets. Based on the results with SMS and CC it is likely that other (sub)layers or services contain bugs as well. And if those attacks can be performed over commercial networks practical attacks are close.

One of the most interesting targets for protocol fuzzing, which to the best of our knowledge have not been tested yet, are commercial basestations or even complete networks. Being able to make a certain type of phone of a certain brand reboot is nice, but the actual impact is rather small. It is very well possible that the same thing can be done with a commercial network, which can have a much higher impact. A promising fact is that many phone manufacturers also build and sell GSM network components, so bugs are likely to be found there as well. The only problem is that (most likely) permission is required from the service provider to fuzz the network and getting permission could prove quite a challenge.

Another option for future research is using OsmocomBB to fuzz other cell phones over commercial networks. Because OsmocomBB is open source it should be fairly easy to modify parts of the GSM stack to let the phone send arbirary messages over the network. One problem is that it is unknown how the network will respond, but fuzzing at the highest layers might be fairly safe.

A fifth starting point for future study is trying to actually exploit the attacks found so far. Some crashes are likely to be caused by buffer overflows and these might be exploitable for remote code execution (for example as shown by [9]).

Finding out what exactly happens on the phone and how this can be exploited is probably very tedious because the source code is not available, but can be very rewarding especially when performed on popular (smart) phones.

Chapter 7

Conclusion

During this research we encountered several strange, funny, annoying and interesting things. In this chapter we discuss some of the highlights, both positive and negative, we found. In addition we summarise the results of this research and draw conclusions based upon these results.

7.1 Interesting observations

An interesting observation in this research was that different types of phones of the same brand more often than not reacted differently to the same fuzzed message. For example the three different types of Nokia phones on which a DoS attack was found misbehaved on completely different fields and would not fail on the fields that brought the others in trouble. This seems to imply that either they did not reuse the firmware code for newer phones or that they did reuse it, but broke parts that worked while fixing parts that were broken. This is contrary to what others (like Welte [10]) observed.

An annoyance we encountered is that OpenBTS is still rather fragile. It does check for errors where they may occur, but often simply with an assertion. The result is that every now and then on unexpected input from the user or phone (the latter often happened with the Samsung SGH-D500) OpenBTS would crash with an assertion failure. Initially when this happened we thought we had to completely reboot the notebook running OpenBTS, because it could not fire up OpenBTS again due to a used socket. But later it was pointed out to us by Ronny Wichers Schreur that killing the process called "transceiver" also stopped the use of that socket. Either way, after firing up OpenBTS it took a good minute for the basestation to have enough signal to be found by cell phones. This has cost us a quite a lot of time over the course of our analysis.

One of our findings that still seems rather strange to us is that a certain SMS message causes the Samsung SGH-A800 to shut down while charging, but to reboot when not charging. This probably means there is something weird going on in the error management and recovery (i.e. crash and restart) module. Either this module checks whether it is charging and changes behaviour based on that (unlikely) or there is something wrong with when the phone considers itself on: showing that it is charging or being in standby.

The most promising finding is probably the possibility to read out the phone memory of the Nokia 2600. This bug seems the easiest to exploit for remote code execution, but nonetheless it will probably cost a lot of time to implement.

7.2 Summary and conclusions

This research focused on the GSM protocol and how protocol implementations could be tested using protocol fuzzing. For each (sub)layer and service of the Um interface it was determined how feasible protocol fuzzing of that part would be. The lower (sub)layers were more challenging than others because of the difficulty of monitoring, while some of the higher level services were problematic because the transactions get initiated by the MS resulting in an inefficient fuzzing process. Besides discussing fuzzing of each piece of the GSM protocol stack the interesting fields of Layer 3 messages are also described. The length fields, the length of the blocks encoded by length fields and reserved values were pointed out as the prime targets, while type fields are most likely to be parsed safely.

All this information was brought together into a practical analysis of sixteen cell phones using a USRP-1. We built a fuzzer to create just over nine hundred ‘smart’ black box test cases and extended OpenBTS to send these test cases. With these test cases nine of the sixteen phones could be caused to reboot, to show parts of their memory, to not notify the user when a certain SMS message was received, to store SMS messages that could not be deleted on that phone or to stop receiving SMS messages altogether until switched off and back on.

From these results we can conclude that the robustness of GSM implementations is rather lacklustre on many phones. Even with a relatively small test set of just over 550 SMS messages and 350 CC messages already more than half of the sixteen tested phones could be caused to reboot or to refuse to accept SMS messages until restarted. These attacks described can not be sent over a commercial network, because the affected headers are built by the network, but they are only the tip of the iceberg. For example we found nothing for the Nokia E70-1 and E71-1 although at least one attack against those phones is known [32], which can actually be sent over commercial networks.

How regrettable it might be, it is not strange issues exist on this many phones. The GSM standards are hopelessly large and complex with tens of thousands of pages full of features and extension mechanisms for future use. Bugs are likely to exist in implementations of these standards, especially considering the fact that many specified features are never used, and thus also never tested, in practise. In addition only just a few years ago bugs could not be found, let alone exploited, by the public without major investments in both time and money, thus these bugs used to be of low risk.

Now with the emerging of cheap SDR systems like the Universal Software Radio Peripheral-1 (USRP-1) and the open source projects OpenBTS, OpenBSC and OsmocomBB attacks are becoming feasible and as a result these bugs more critical. With this research we hope to make people more aware of the issues and create an incentive for manufacturers to pay more attention to robustness of their products.

Appendix A

Details of the GSM protocol

In this appendix the GSM protocol is described in more detail. The first section is a more thorough description of the GSM Um interface than that of Section 2.2. The remainder of this appendix describes the messages sent on Um Layer 3 in detail, providing a better understanding of the exact fuzzing parameters introduced in Appendices D and E.

A.1 Um interface details

In this section more details regarding the Um interface are described. The structure of this section is similar to that of Section 2.2, describing the protocol stack layers from bottom to top.

A.1.1 Physical Layer

The bottom layer of the GSM protocol stack is the Physical Layer. It provides abstractions to the Data Link Layer right above it in the form of logical channels. Logical channels are defined by frequency, timing and purpose. In this section details regarding the frequencies, timing and different purposes of logical channels are described.

GSM frequencies

Every PLMN operates in a certain frequency range or band. In most parts of the world the GSM-900 (890-960 MHz) and GSM-1800 (1710-1785 MHz) are the frequencies for GSM, however in the United States and Canada the GSM-850 (824-894 MHz) and GSM-1900 (1930-1990 MHz) bands are used.

Each GSM band is split up in frequency channels of 200 kHz that can be used simultaneously. This is called Frequency Division Multiple Access (FDMA). Each frequency channel is used for uplink or downlink, but never both. Each band has a specific offset between uplink and downlink for all frequency channels, resulting in the lower frequency half of the band always being used for uplink and the higher frequency half for downlink.

Communication on a logical channel between an MS and BTS does not necessarily stay on the same frequency channel; to reduce interference the BTS can decide to use frequency hopping. In that case the BTS communicates parameters to the MS so both parties can derive the same sequence of frequencies to jump through at a set rate (around every 4.615 ms). This process is completely transparent to the higher level layers.

GSM timing

Besides FDMA GSM traffic is also multiplexed in time, called Time Division Multiple Access (TDMA). Each frequency channel is divided into eight time slots, creating eight logical channels out of one frequency channel. Traffic between an MS and BTS only uses one time slot on a certain frequency, so eight MSs can communicate on the same frequency channel to the same BTS. There is much more going on with regard to timing in the GSM protocol, but this will not be discussed here. For further reading see [33, 14].

Logical channels

Each logical channel created through FDMA and TDMA can be used as Traffic Channel (TCH) or one of the many different control channel. Nearly all traffic on the TCH is speech data, while all the traffic on any of the control channels is to let the network function properly.

There are ten different types of control channels, aggregated in three subtypes:

1. Broadcast Channels
2. Common Control Channels
3. Dedicated Control Channels.

Each of these subtypes and the channels belonging to these subtypes are briefly described below. Only the channels relevant for this research are described in more detail.

- The Broadcast Channels (BCHs) are used by the BTS to broadcast the network parameters and synchronisation information to all MSs in its cell, even those not yet connected to the BTS. These channels are used continuously in the downlink direction and ensure that all MSs can determine the correct frequencies and timing to communicate with the BTS and initiate a connection to the network. The channels in this subtype are the:

  1. Broadcast Control Channel
  2. Frequency Correction Channel
  3. Synchronisation Channel
  4. Cell Broadcast Channel.

  Because these channels are not relevant for this research they will not be described. For further reading about these channels see [14].

- The Common Control Channels (CCCHs) are shared between the BTS and all MSs in the cell. The main use of these channels is to set up a Dedicated Control Channel between the BTS and a single MS. The three CCCHs that exist are the:

1. Paging Channel
2. Random Access Channel
3. Access Grant Channel.

The Paging Channel (PCH) is used by the BTS to inform MSs of an incoming call or message. This process is called paging. Messages on this channel are addressed to a single MS by IMSI or Temporary Mobile Subscriber Identity (TMSI) (see Section A.1.3).

The Random Access Channel (RACH) is the only uplink CCCH. This channel is shared between all MSs and used to request a Dedicated Control Channel (DCCH) from the BTS. When an MS wants to transmit a message on this channel it just sends it and listens whether any other MS transmitted a message at the same time. If multiple MSs sent a message at the same time they all wait a random delay before they try sending their message again.

The last CCCH is the Access Grant Channel (AGCH). Just like the PCH it is a downlink channel sending a message to an MS addressed with the IMSI or TMSI. Responses to requests on the RACH are sent on this channel and contain information about the allocated Dedicated Control Channel (DCCH).

- The last subtype of control channels are the Dedicated Control Channels (DCCHs). Unlike the other two subtypes these channels are not shared by all the MSs in the cell, but used by a single MS and the network. DCCHs are, among others, used for location updates, call setups, handovers and SMS messaging. All of the DCCHs are both uplink and downlink channels. There are three types of DCCHs:

  1. Standalone Dedicated Control Channel
  2. Slow Associated Control Channel
  3. Fast Associated Control Channel.

  The Standalone Dedicated Control Channel (SDCCH) is in most cases the first channel that is being allocated upon a channel request over the RACH and positive response from the BTS over the AGCH. An SDCCH can be used for call setup (which includes allocating a TCH), location updates or sending of SMS messages. It is also used for authentication and setting up of encryption. As soon as the SDCCH has outlived its purpose, or some error has occurred, it can be released again so the logical channel can be reused.

  A Slow Associated Control Channel (SACCH) is an internal part of every TCH and SDCCH. Messages on this logical channel are sent instead of a regular TCH or SDCCH message at a set interval (for TCHs this is once every 26 messages). The two main purposes of the SACCH are improving channel performance (for example by issuing synchronisation commands) and the ‘heartbeat’ signal for that TCH/SDCCH. Reception of the ‘heartbeat’ SACCH message means that the associated channel is still alive, therefore in every SACCH frame a message has to be sent, even if no control data has to be transmitted.

The last DCCH is the Fast Associated Control Channel (FACCH). Like the SACCH it is not an actual separate logical channel, but is instead part of every TCH. Unlike the SACCH the FACCH does not have a set timing within the TCH, but can instead 'steal' part of a data burst for control messaging. The FACCH is used for urgent control messages like handovers or hangups (disconnects), which can happen at unpredictable times during a conversation.

A.1.2 Data Link Layer

As discussed in Section 2.2.2 the Data Link Layer offers two services to Layer 3:

- acknowledgements and retransmits when applicable
- segmentation when applicable.

The Data Link Layer can transmit messages in either acknowledged or unacknowledged mode. Messages sent in acknowledged mode have to be acknowledged by the receiver, which can be either the MS or the BTS. If the sender does not receive such an ACK within a certain time frame the message is retransmitted until an ACK for that message does arrive in time. This guarantees that all messages arrive at the destination. Messages sent in unacknowledged mode do not have such guarantees and are sent on a best-effort basis.

Whether acknowledged or unacknowledged mode is used depends on logical channel and message; acknowledged mode can only be used on DCCHs, while unacknowledged mode can be used on all channels except for the RACH. Messages on the RACH use a different scheme altogether, because multiple MSs can transmit on this channel at the same time. For further reading on Data Link Layer RACH messages see Clause 4.2.6 of [19].

In case acknowledged mode is used the Data Link Layer can also perform segmentation. This is used when a Layer 3 message that has to be transmitted in acknowledged mode is too long to fit in a single Data Link Layer message. In such a case the message is split up in several parts and the parts are transmitted separately. The Data Link Layer on the receiving end sends an ACK for each segment. When all segments have arrived the receiving Data Link Layer restores the original Layer 3 message and passes it on to the receiving Layer 3. In unacknowledged mode the Data Link Layer does not offer any segmentation capabilities.

A.1.3 Layer 3

In this subsection further information regarding sublayers for Radio Resource and Mobility Management of the GSM Layer 3 is given. The CM sublayer was already quite extensively described in Section 2.2.3.

RR sublayer

The Radio Resource (RR) sublayer is the lowest sublayer of the Layer 3 stack and described in [34]. It interfaces with the Data Link Layer below it, and the Mobility Management sublayer above it. Messages on the RR sublayer are sent over BCHs, CCCHs and also DCCHs.

The most important aspect of the RR sublayer is the creation and tearing down of TCHs and DCCHs. The higher sublayers can request the RR sublayer to use this functionality to create an RR connection, which allows point-to-point communication between the network and the MS. This connection is subsequently used by the higher sublayers to exchange messages.

Part of the procedure of establishing and maintaining these RR connections are handovers. Whenever an MS roams (i.e. moves) to a different cell the RR sublayer takes care of the correct opening of channels with the new BTS and then closing of the old channels. This happens without the user or the upper sublayers noticing it.

Besides channel management and handover support the RR sublayer also takes care of transmitting control data over the various other control channels and keeps track of which kind of ciphering is used on which channel.

MM sublayer

The Mobility Management (MM) sublayer is used to keep track of the location of connected MSs, connect users that are new to the network and allow CM sublayer services to send service control messages. MM runs on DCCHs only.

Before a newly arrived MS can be connected to the PLMN it has to authenticate itself first via its IMSI and a challenge-response scheme based on the secret symmetric key the SIM shares with the provider. This message exchange takes place on the MM sublayer. Only when the MS successfully authenticated itself it can be connected to the PLMN[14].

Another (optional) feature that operates on the MM sublayer is the issuing of a TMSI to an MS in order to provide identity confidentiality. The mapping of TMSIs to IMSIs is stored inside the NSS. The TMSI is only valid in the area it was issued. The network decides whether TMSIs are used and when they are updated.

A.2 Layer 3 message format

All messages sent on Layer 3 of the GSM protocol have a standard message layout. This layout is the topic of this section. First some basics of the message formats and representations used in the GSM protocol are described. Then the header of Layer 3 messages is described, followed by a section on the (optional) body of the messages. The terms used in this section are the same as those used in the full specification of the Layer 3 message layout found in [20].

A.2.1 GSM Layer 3 message basics

GSM Layer 3 messages are nothing more than bit strings of finite length. The bits are usually ordered in groups of eight, called an octet in the ETSI documents. The bits in an octet are numbered from right to left as 1 to 8 with bit 8 being the most-significant bit. Consecutive octets are simply concatenated for GSM, but in the figures used throughout this paper they are depicted from top to bottom. Another convention used in this paper is to write numbers simply

[14]In the authentication protocol the session key with which all traffic data will be encrypted is created as well.

Figure A.1 Semi-octet representation of the number "123". The half-octet value `1111` is used to mark the end of the number in case of an odd amount of digits.

8	7	6	5	4	3	2	1
0	0	1	0	0	0	0	1
1	1	1	1	0	0	1	1

as number (e.g. 22), hexadecimal representation of a number starting with 0x (e.g. 0x16), binary notation in a monospaced font (e.g. `00010110`) and a literal string between double quotes (e.g. "house").

The way data within an octet is represented can differ and depends on the use of the octet. The most commonly used data representation schemes are:

- Bit fields. A run of at least one bit in the octet can have a specific meaning, allowing for up to eight bit fields to be specified in a single octet.
- Spare parts. Parts of an octet can consist of spare bits. A spare bit should be coded as a default value (usually 0), but should not result in an error when coded as a different value. Spare parts do not have a meaning and are only used as fill bits.
- Integer. The octet can represent an integer either in (unsigned) binary or 2-complement binary.
- Semi-octet. The octet is split in two parts of four bits (half-octet) and each half-octet represents an integer. For both parts the most-significant bit is the left-most bit. This representation is typically used for addresses (or more commonly called phone numbers). Important to note is that the digits of the address are stored in the octet from right to left, as shown in Figure A.1.

Every Layer 3 message consists of at least two octets which form the header of the message. The header is optionally followed by a body of data octets. The format of the header is described in the next section, the format of the body in the subsequent section.

A.2.2 Layer 3 message header

The first two (or three) octets of each Layer 3 message form a header. This header consists of three parts:

1. Protocol Discriminator (PD)
2. Transaction Identifier (TI) or Skip Indicator
3. Message Type (MT).

In Figure A.2 the first octet of a Layer 3 message is shown. Bits 1 to 4 of this octet are the Protocol Discriminator (PD). The PD determines which Layer 3 sublayer should handle the message. In Table A.1 the meaning of all possible PDs is given.

Figure A.2 First octet of a GSM Layer 3 message.

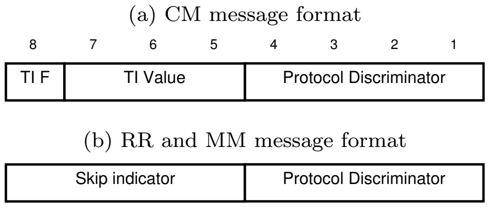


Table A.1 Meaning of all sixteen possible Protocol Discriminators. Note that not all of them are used for GSM; Evolved Packet System (EPS) and General Packet Radio Service (GPRS) specific messages are defined too.

bits	Sublayer
`0000`	GCC
`0001`	BCC
`0010`	Evolved Packet System (EPS) Session Management messages
`0011`	CC & call related SSs
`0100`	General Packet Radio Service (GPRS) Transparent Transport Protocol
`0101`	MM messages
`0110`	RR management messages
`0111`	EPS MM messages
`1000`	GPRS MM messages
`1001`	SMS messages
`1010`	GPRS Session Management messages
`1011`	Non call related SSs
`1100`	LCSs
`1101`	Undefined in the GSM standard
`1110`	Reserved for future extension of the PD to one octet length
`1111`	Reserved for test procedures

The other half of the first octet (bits 5 to 8) is occupied by the TI or Skip Indicator[15]. CM messages use the format of Figure A.2(a) with the TI, while RR and MM messages use the format of Figure A.2(b) with a Skip Indicator.

The Skip Indicator is a spare part and should be coded as `0000`. The TI is used to identify to which transaction this message belongs. The TI consists of two parts, the TI flag and TI value. The TI flag is a single bit at bit position 8 that encodes who initiated the transaction. Messages sent by the initiator have the TI flag set to `0`, while messages sent by the other party have the TI flag set to `1`. The TI value is coded in the remaining three bits (5 to 7) and is a value between 0 and 6. By varying the TI value and TI flag seven uplink and seven downlink transactions can be identified in parallel.

In case these seven transactions in either direction are not enough an extension mechanism is specified. When the extended Transaction Identifier is used the TI value in octet one of the Layer 3 message is coded as `111`. This means that bits 1 to 7 of the second octet encode the TI value of this transaction, while

[15]There are actually two other possible meanings for these four bits specified in [20], but these are not used in GSM.

Figure A.3 First two octets of a Layer 3 message header using an extended Transaction Identifier.

8	7	6	5	4	3	2	1
TI F	1	1	1	PD			
EXT	TI Extended						

Figure A.4 Format of the Message Type octet in a Layer 3 message header for the sublayers from Table A.1.

(a) RR format

8	7	6	5	4	3	2	1
0	Message Type						

(b) GCC, BCC and LCS format

0	N(SD)	Message Type					

(c) MM, CC and SS format

N(SD)		Message Type					

(d) SMS format

Message Type							

the TI flag in the first octet retains its meaning. This is shown in Figure A.3. The EXT bit can be used to further extend the extended TI by setting it to `0`, but this is reserved for future use only. It should now always be coded as `1`.

The octet after the TI is the Message Type (MT) octet. Usually this is the second octet in the Layer 3 message, but when the extended TI is used it is the third. The most important part of the MT octet is (confusingly) called the Message Type (MT) field. Together with the PD and the TI flag the MT field uniquely identifies the exact type of the message, for example whether a CC message is to setup a new call or an RR message is to release a channel. The exact meaning of the MT field therefore depends on the PD and also on the TI flag of the message.

There are slightly different formats to represent the MT octet, which format is used depends solely on the PD. The four possible formats are shown in Figure A.4. In RR messages bit 8 is set to the default value `0`, the rest is used for the MT field. In LCS, MM, CC and SS messages a send sequence number (N(SD)) is included. This field is incremented (ignoring overflow) for every next message and is used to detect duplicate messages in the case of roaming. In LCS messages this field takes up a single bit with the other bit set to `0`, in the other three cases the N(SD) is two bits. Finally for SMS messages the whole MT octet is used as MT field.

A.2.3 Layer 3 message body

Most Layer 3 messages do not only contain a header, but also transmit data to the receiving party. This is done using the message body. Data in the body is represented in Information Elements (IEs), which is in essence a different name

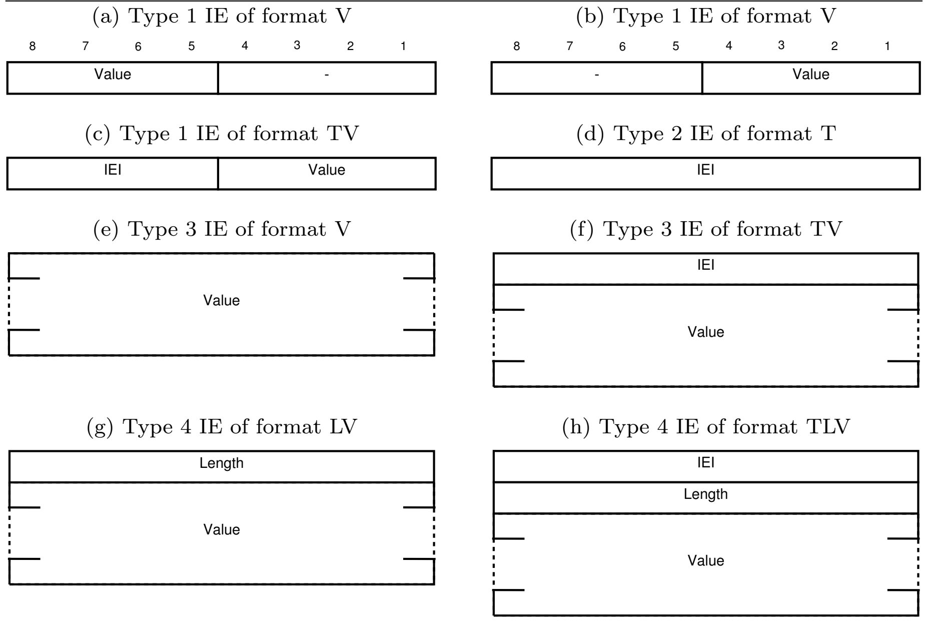

(a) Type 1 IE of format V
8 7 6 5 4 3 2 1
Value
-
(b) Type 1 IE of format V
8 7 6 5 4 3 2 1
-
Value
(c) Type 1 IE of format TV
IEI
Value
(d) Type 2 IE of format T
IEI
(e) Type 3 IE of format V
Value
(f) Type 3 IE of format TV
IEI
Value
(g) Type 4 IE of format LV
Length
Value
(h) Type 4 IE of format TLV
IEI
Length
Value

Figure A.5 The eight different formats of Layer 3 Information Elements.

for Type-Length-Value (TLV) triples. In the GSM standards the type is called Information Element Identifier (IEI), the length is called Length Indicator (LI) and the value is simply called value. The LI specifies the length of the value in octets. Not all of these three fields have to be present, as long as at least one of them is. This is done to save precious space in Layer 3 messages.

For each unique message type (thus PD, TI flag and MT field combination) the GSM standards define which IEs are mandatory and which IEs are optional. Mandatory IEs always precede optional IEs. Because the mandatory fields are always present and in a static order in the message, the IEI for those IEs is omitted to save space. Likewise if a specified IE (mandatory or optional) always has a certain length (including length zero) the LI field can be omitted. Therefore seven possible combinations of IE TLV elements exist, with a total of eight different IE formats defined called Type 1 to 4. These are depicted in Figure A.5 and described below.

Type 1 IEs have a value that takes up only four bits. It can be used either with or without IEI. When the IEI is omitted the value can be the left or right half-octet (Figure A.5(a) and A.5(b) respectively), depending on the circumstances. In case an IEI is used it is always the left half-octet while the value is the right half-octet (Figure A.5(c)).

Type 2 IEs only have an IEI of one octet and do not have an LI or value (Figure A.5(d)). They are nothing more than an optional flag.

Type 3 IEs have a value of a static length of at least one octet. Because the length is static the LI field is omitted. Type 3 IEs can be used without an IEI or with an IEI (Figure A.5(e) and A.5(f) respectively).

Type 4 IEs have a value of variable length of zero or more octets. Therefore the LI element is always present. Just like the Type 1 and Type 3 IEs the Type

4 IE can have an IEI present or not (Figure A.5(g) and A.5(h)).

Appendix B

Protocol fuzz testing in depth

Protocol fuzz testing is a testing technique based on 'regular', i.e. non-protocol, fuzz testing (which will be called original fuzz testing in this paper to prevent ambiguity). Original fuzz testing is a testing technique to test (generic) software reliability and robustness. Reliability and robustness issues can directly impact the security of the software, because they can threaten confidentiality, integrity (e.g. using a buffer overflow exploit) or availability (e.g. DoS by causing the software to crash). Original fuzz testing is done test cases executed by the program. This is in contrast to human testing methods like code inspection, walk-through and review, where a small group of people carefully checks the software code for programming errors. Testing using test cases is recommended to be performed after human testing [35].

Original fuzz testing is already relatively old. Studies of different original fuzzing strategies date back to the end of the 70s and start of 80s [35, 36]. Protocol fuzz testing is much newer and is in essence an advanced form of original fuzzing. Messages in a protocol often have a very specific format with optional fields and variable lengths (for example HTTP headers or IP packets) and a certain encoding (e.g. ASCII or Unicode). Compared to a Graphical User Interface (GUI) or character string as input method in most generic software this allows for many more fuzzing possibilities. Furthermore in a protocol environment consecutive input to the program is often much more directly related to previous input or output (for example through a nonce), or to a static value not revealed in the communication (cryptographic keys). Original fuzz testing usually does not have these characteristics.

This appendix first describes original fuzz testing in more detail. It also describes a few original fuzzing techniques, because some of the concepts are also applicable for protocol fuzzing. The second section describes several existing protocol fuzzing tools and their characteristics. In the last section it is described in more detail why we did not reuse an existing protocol fuzzing tool, but instead chose to build our own tool for this research.

B.1 Original fuzz testing

Original fuzz testing comes in a large variety of flavours, from quick and dirty to slow and extensive. Each strategy has different characteristics and make it suitable for different kinds of verification, depending on time, budget and desired quality.

The quality of a fuzz test can be expressed by its code coverage. Code coverage is a measurement of the amount of software code that is actually tested by a fuzz testing set. The higher the code coverage, the higher the number of (potentially erroneous) statements executed and thus the more likely it is that the application will misbehave.

Original fuzz testing techniques can be divided into two main categories: black box and white box, depending on the knowledge of the source code of the software. In this section both of these categories are described more extensively.

B.1.1 Black box fuzz testing

Black box fuzzing is the easiest form of fuzz testing, but not necessarily the most effective. In the black box approach the specifics of the software are ignored, which allows you to use simple and efficient methods to (pseudo-)randomly generate input for the software.

Two basic approaches of black box fuzzing exists: random fuzzing and exhaustive fuzzing. These two basic strategies are discussed below.

Random fuzz testing

Random fuzzing is the most basic approach of black box fuzzing and is used quite often. It only uses a couple of very basic parameters, like input length, input length variation and possible values for the bytes to test. This makes random testing very easy to implement and highly portable between different applications.

The simplicity and cheapness of random fuzz testing are also its curse. Because the test input is randomly generated, it is unlikely that all execution paths in the implementation are tested. For example, if the application only supports one single value for a certain byte and the implementation checks this byte properly, then only 1 out of $2^8 = 256$ test values get past this check. If more of these checks are included for other bytes or larger data structures, the test set that tests past this point in the software quickly shrinks, and so does the code coverage as a result.

Exhaustive fuzz testing

The second basic black box fuzzing strategy is exhaustive fuzzing. Exhaustive fuzzing is simply trying every single input value on the software, which consequently means you have full code coverage and thus are guaranteed to find all robustness issues (and as a side-effect also correctness issues).

However, even though you are sure to find all problems, exhaustive fuzzing has a big problem: it is in almost all cases infeasible. More often than not the input domain will be infinitely large (for example all possible bit sequences) or at least too large to test exhaustively within a reasonable time frame (for example all sequences of printable ASCII characters of length up to 15). Therefore practical applications of exhaustive fuzzing are hard to find.

B.1.2 White box fuzz testing

White box fuzz testing is more sophisticated than black box fuzz testing, because the software code is taken into account to generate the test cases, which can result in tests with higher code coverage. Downside is that it is much more labour intensive to create white box fuzz tests and the resulting tests are unlikely to be portable to other applications.

One of the most well-known white box testing approaches is partition fuzz testing [37, 38]. The idea of partition fuzzing is that you partition potential inputs in domains according to some criteria and test at least one input from each domain. Other white box approaches include using constraint solvers to efficiently and automatically mutate correct input to create (slightly) incorrect input with large code coverage [39]. In this section the focus is on partition fuzzing strategies, because this approach is also used in this research.

Partition fuzz testing

Partition fuzz testing divides the software input in domains and executes at least one test case from each domain. It is important to note that often the partitions made for partition testing are not mathematical partitions, because they are not necessarily disjoint (on the other hand, the union of all partitions usually does span the entire input space).

Partition fuzzing comes in a large variety of flavours, from very generic to very specific. The specificity of partition fuzz testing depends on the criteria used to create the partitions of the input domain. Several standard criteria have been established for often employed varieties of partition fuzzing and each of these varieties has been named. Some examples, ordered from low code coverage to high:

1. Random fuzzing. While not really considered a white box fuzz testing strategy, it can be seen as an extreme form of partition fuzzing where all possible inputs are collected in a single domain from which multiple inputs are drawn.

2. Statement coverage. A variety for which each program statement has a domain, which contains exactly those program inputs that execute that statement. The resulting domains are (extremely) overlapping, but executing the test means that all statements are tested at least once.

3. Branch coverage. With this form the program inputs are divided in domains depending on execution path through the program, such that the all different branches of the program are covered by at least one domain [40]. The code coverage is equal to that of statement coverage, but the number of partitions is generally much lower. This strategy results in overlapping domains.

4. Path coverage. A flavour where the program inputs are divided in sets that share the exact same entrance and exit statement (i.e. paths). The

Block B.1 Branch coverage might not find the error in this function, while path coverage will.

```
Input: $a$ as integer
Returns: integer
$x \leftarrow 3$
if $a > 20$ then
    $x \leftarrow x - 2$
end if
if $a > 1 \wedge a < 22$ then
    $x \leftarrow x - 1$
end if
return $6/x$
```

difference with branch coverage is subtle and will be explained with an example below. Path coverage results in partitions that are disjoint [37].

5. Exhaustive testing. Another black box approach that can be seen as extreme partition testing where every single program input value has its own domain and thus all input values are tested. Trivially this also means that all partitions are disjoint.

The difference between branch and path coverage will be explained with an example. The pseudo-code in Block B.1 is a small function that does something completely arbitrary, but will crash if and only if $a = 21$ because that results in division by zero. Using branch coverage there will be four partitions, two for each if-statement depending on whether the condition evaluates to true or false. The partitions are (in terms of a):

1. $a \leq 20$ (first condition false)
2. $a > 20$ (first condition true)
3. $a \leq 1 \vee a \geq 22$ (second condition false)
4. $a > 1 \wedge a < 22$ (second condition true).

If one random case from each partition is tested, the probability that the error case is picked is approximately 1 : 20.

With path coverage we see a very different picture. Path coverage also has four partitions, but this time it is for all possibilities through both if-statements combined. In terms of a that gives:

1. $a \leq 1$ (both conditions false)
2. $a \geq 22$ (only first condition true)
3. $a > 1 \wedge a \leq 20$ (only second condition true)
4. $a = 21$ (both conditions true).

This time if we pick a random case from each partition we are sure to find the division by zero problem with case 4.

B.2 Existing protocol fuzzing tools

Over the years several protocol fuzzing tools and frameworks have been released. Some of the more well-known tools will be described below, but many more exist[16].

B.2.1 SPIKE

SPIKE is one of the older and most heavily used smart fuzzing frameworks [29]. It is a free network protocol fuzzer for smart black box fuzzing written in C. SPIKE was the first to use a block-based approach for data representation. With this approach the protocol messages can be build from blocks of primitive data types and for each block it can be specified if and how it should be fuzzed. In addition it enables the creation of dependencies among blocks, for example to implement length fields and checksums. Aitel [41] argued that block-based data representation allows for easier and quicker representation of complex protocol data than other approaches. A disadvantage of SPIKE is that it is state-less and protocol state fuzzing has to be manually implemented by the user.

Besides normal SPIKE there is also SPIKEfile, which works similarly but is made for file fuzzing.

B.2.2 Sulley

Sulley is an open source network protocol fuzzing framework for smart black box fuzzing written in the Python programming language and released under a GPL license [30]. Sulley does not only assist in automated test generation, but also takes care of the data transmission and can even do target monitoring using virtual machines. Especially the latter eases the burden on the fuzz tester, since you can leave Sulley running without paying attention to it and still get reports of all the inputs that caused an error. The data representation scheme of Sulley is a tidied up version of that of SPIKE, and hence also block-based. Additionally Sulley supports protocol state fuzzing by default.

Peach

Peach is an open source smart black box fuzzing platform released under the MIT License [42]. The current version (version 2) is written in Python and allows for both generation-based and mutation-based fuzzing. Both message formats (block-based like SPIKE and Sulley) and the state machine are represented using eXtensible Markup Language (XML), allowing for high customisability but also a steep learning curve [43]. Just like Sulley it also supports process monitoring of the target process with virtual machines.

B.2.3 mangleme

mangleme is a smart black box fuzzer specifically designed to automatically generate broken HyperText Markup Language (HTML) in order to test web browsers and is released under the LGPL license [31]. It has successfully found

[16] `http://www.fuzzing.org` contains a more extensive, yet also not exhaustive, list of fuzzers.

dozens of bugs in all major web browsers. Unlike the previous three it is not very customisable and really only for HTML.

B.3 Fuzzing tools and GSM

The tools discussed in the previous section are not made to be used with cell phones. Especially the target monitoring aspects generally work on network interfaces and virtual machines, while we have separate devices connected over a (custom) radio interface. This makes automatic target monitoring with one of these tools impossible. However, one of these tools could be used to generate fuzzed messages based on the protocol specifications or by mutating correct input. The block-based approach from SPIKE, Sulley or Peach seems the most useful for this.

B.3.1 Picking a tool

Picking a tool for fuzz test generation turned out to be harder than expected. Since Sulley is essentially SPIKE but in Python and with additional features, SPIKE was the first framework to be dropped. Peach followed closely because the documentation was confusing and the learning curve was indeed very steep. Only Sulley was left in the running.

But Sulley was not ideal either. Sulley does not support fuzzing on binary data very well (for example fuzzing only certain bit positions within a byte), while this is required to be able to fuzz GSM messages. In addition Sulley does not innately support writing output to a file, but this was fairly easy to solve, because each output block could already be rendered to a string. A third problem is that state fuzzing is done at a later stage and not compatible with using the render function for the blocks to print it to an output file. In order to fuzz a message sequence with Sulley one has to manually concatenate multiple outputs.

To get around these issues it seemed easier to built a trimmed-down version of Sulley from scratch, with all required features supported by default, than to extend Sulley itself. The resulting fuzz test generator would be much smaller, since the whole monitoring part could be left out entirely. More about the fuzzer can be found in Appendix C.

Appendix C

Fuzz test generator

This appendix describes the fuzzer GSMfuzz used for this research. It is a mutation-based smart fuzzer, with features designed specifically for GSM, but nonetheless can be used for other protocols as well. The fuzzer is written in the Python programming language (version 2.6) and loosely based on Sulley. It therefore uses the same block-based approach as SPIKE and Sulley. It has the following features:

- Block-based with syntax similar to Sulley
- Fuzzing of bit positions within a byte
- Possibility to exclude certain byte values from the result
- Partition fuzz testing of special fields (type, length), resulting in few cases with maximum impact
- Innate support for the eight different GSM Layer 3 IEs
- Length fields can count octets, septets or half-octets
- Output in hexadecimal to a file, which can be used directly in our extended version of OpenBTS (OpenBTSraw).

GSMfuzz itself is just over 900 lines of code (excluding white space). Besides the source code of the program itself we created 34 files with input to mutate valid messages. The input files are 3601 lines in total (excluding white space and comments).

Both GSMfuzz and OpenBTSraw will be published in the near future under the GNU GPLv2 license and GNU AGPLv3 license respectively. For now source code for both projects can be gotten upon request. The input files for GSMfuzz can be gotten upon resuest as well.

Appendix D

SMS fuzzing

The Short Message Service (SMS) service is used to transfer messages from user to user or from network to user (an example of the latter is the "Welcome!" message when entering a different country). As described in Chapter 4 the SMS sublayer offers many options for protocol fuzzing the header, but few options for fuzzing the state machine. In this appendix all options are explored and it is pointed out which were tried in this research.

D.1 SMS protocol stack

The SMS sublayer is in turn a small protocol stack (see Chapter 2). Four different layers can be identified within the protocol, which are depicted in Figure D.1 and described in Section D.3:

1. Connection Management (CM) specified in [22]
2. Short Message Relay Layer (SM-RL) specified in [22]
3. Short Message Transfer Layer (SM-TL) specified in [23]
4. Short Message Application Layer (SM-AL)

The SMS stack is layered because SMS messages are delivered on a best-effort basis. If the network is very busy (e.g. on New Year's Eve) or the recipient is unreachable (e.g. the MS is switched off) delivery of the message can be delayed and the message can even be discarded in case the recipient remains unreachable for a longer period of time. In addition the layered model made it possible to standardise SMS not only for GSM but also for 3G and 4G protocols.

Whenever a mobile originating SMS message is sent, it is first delivered to the SMS-Service Centre (SMS-SC), a part of the NSS in a PLMN. This SMS-SC stores the received message and attempts to forward it to the recipient. If forwarding fails for any reason the SMS-SC keeps the message for a certain time and periodically polls whether the message can be delivered. When the receiving MS is connected to a network and there is bandwidth available the SMS-SC forwards the message via an intermediate NSS entity to the MS (see Figure D.2). After successful delivery the copy of the message in the SMS-SC

Figure D.1 The SMS protocol stack. On the very right are the names of the messages on that specific layer.

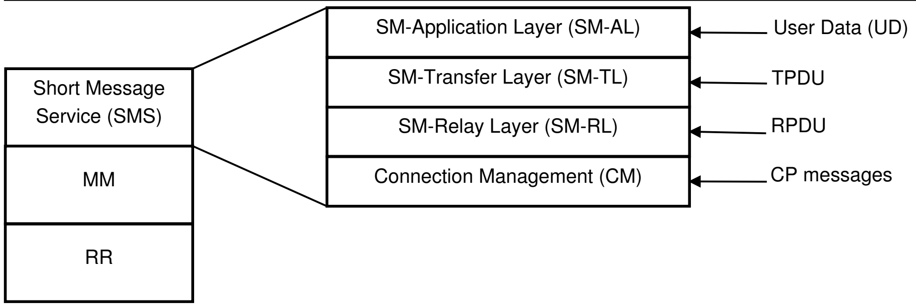
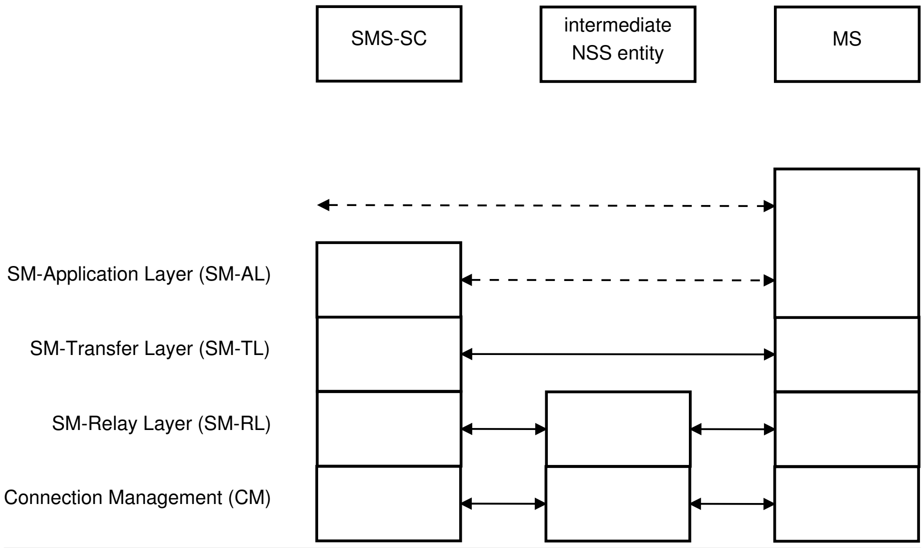

Short Message Service (SMS)
MM
RR
SM-Application Layer (SM-AL)
SM-Transfer Layer (SM-TL)
SM-Relay Layer (SM-RL)
Connection Management (CM)
User Data (UD)
TPDU
RPDU
CP messages

Figure D.2 Interaction between the SMS-Service Centre and the MS on the different layers of the SMS protocol (simplified) [23].


SMS-SC
intermediate NSS entity
MS
SM-Application Layer (SM-AL)
SM-Transfer Layer (SM-TL)
SM-Relay Layer (SM-RL)
Connection Management (CM)

is deleted. However if the validity period of a message stored in the SMS-SC expires before the target MS is available the message is discarded[17].

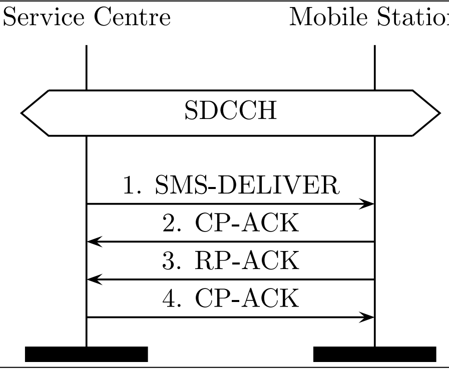
In the upcoming two parts the process of delivering an SMS message to the MS are further described. First the sequence of messages is described and then the layers and the exact format of the messages on each layer are explained.

D.2 SMS message delivery process

Delivery of SMS messages happens in a few steps. The first step in delivering an SMS message to an MS is setting up an MM connection with that MS. This is done by paging the MS over the PCH and allocating an SDCCH upon receiving the response over the RACH. The MS then initiates all the steps for setting up

[17]By default the sender is not notified of this. However on many modern cell phones the sender can request a status report from the network upon (un)successful delivery of the SMS message.

Figure D.3 Message sequence chart of delivering an SMS to an MS.



an MM connection on the SDCCH and when successful delivery can start.

After the MM connection is established four messages are exchanged as shown in Figure D.3. The first message is the SMS-DELIVER message sent from the SMS-SC to the MS. This message contains the actual content (user data) with an optional User Data Header (UDH) and mandatory Transfer Protocol (TP), Relay Protocol (RP) and Connection Protocol (CP) headers. The MS first parses the CP header and verifies it. If it is valid the MS returns a CP-ACK message, otherwise it returns a CP-ERROR message and releases the MM connection to free up the allocated channel.

If the CP header was correct the MS continues by verifying the RP header and checking if the phone has enough memory to store the message. If either of those checks fails it returns an RP-ERROR and releases the MM connection. If both checks succeed the MS returns an RP-ACK with a CP header.

The final message is sent by the SMS-SC when the RP-ACK passes the checks for the CP header at the network side.

D.3 SMS message structure

In this section the different layers of the SMS stack are discussed from bottom to top, as well as the structure of messages on each layer.

D.3.1 SMS Connection Management sublayer and CP messages

The CM sublayer in SMS differs from the CM sublayer as part of Layer 3. The Layer 3 CM sublayer is used to denote any of the six services that can be run on top of the MM sublayer, while the SMS CM layer is to ensure proper delivery of Relay Protocol Data Units (RPDUs), the messages of the layer immediately above it.

For correct delivery of RPDUs there are three different CP message types defined for the SMS Protocol Discriminator (which was 9 as shown in Table A.1). The CP message type is contained in the Layer 3 Message Type (MT) field and encoded as follows:

- CP-DATA, used to deliver an RPDU, encoded with MT 0x01

Figure D.4 Format of Layer 3 SMS CP-DATA messages.

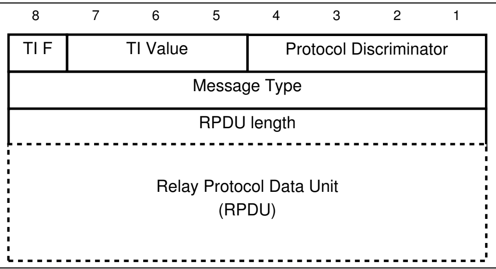
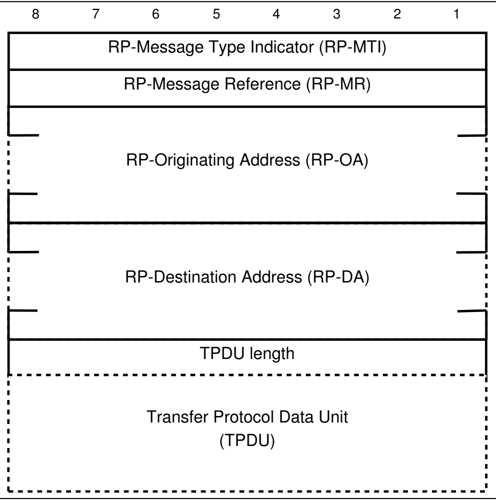


Figure D.5 Format of SMS RP-DATA messages. The entire message is the Relay Protocol Data Unit (RPDU).


8 7 6 5 4 3 2 1
RP-Message Type Indicator (RP-MTI)
RP-Message Reference (RP-MR)
RP-Originating Address (RP-OA)
RP-Destination Address (RP-DA)
TPDU length
Transfer Protocol Data Unit (TPDU)

- CP-ACK, used to acknowledge the received CP-DATA, encoded with MT 0x04
- CP-ERROR, used to report an error with a received CP-DATA message, encoded with MT 0x10.

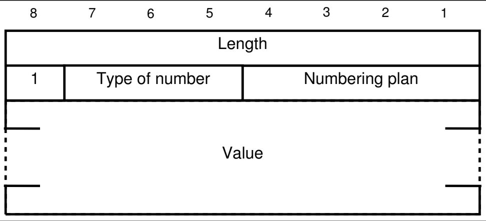
A CP-DATA message contains as body a single Type 4 LV Information Element (IE): the RPDU (see Figure D.4). A CP-ACK only consists of the Layer 3 header and does not have a body. CP-ERROR messages contain a single Type 3 V IE of a single octet as body, encoding the reason for the error. An example error cause is 81 when the received TI was invalid.

D.3.2 Short Message Relay Layer and RPDUs

The Short Message Relay Layer (SM-RL) is used by both the SMS-SC and MS to report problems with delivery, for example when the destination address is unknown or when the memory of the receiving MS is full. To this end RP messages, i.e. Relay Protocol Data Units (RPDUs), are used which contain several fields:

1. RP-Message-Type-Indicator (RP-MTI), a Type 3 format V IE of one

Figure D.6 Format of the Originator-Address and Destination-Address Information Elements.



octet. This field specifies which type of RP message this is and is mandatory in all RPDUs. Four different values are defined:

- RP-DATA, used to transfer a Transfer Protocol Data Unit (TPDU) (see Figure D.5). Encoded as 0x00 from MS to SMS-SC and 0x01 in the other direction
- RP-SMMA, used to notify the network that the MS again has memory available to receive SMS messages. Encoded as 0x06
- RP-ACK, used to acknowledge receiving of an RP-DATA or RP-SMMA message. Encoded as 0x02 from MS to SMS-SC and 0x03 in the other direction
- RP-ERROR, used to report an error with a received RP-DATA or RP-SMMA message. Encoded as 0x04 from MS to SMS-SC and 0x05 in the other direction.

2. RP-Message-Reference (RP-MR), an IE in Type 3 V format of one octet. This single octet is a reference number used to link an RP-ACK or RP-ERROR reply to the preceding RP-DATA or RP-SMMA message. It is a mandatory IE in all the possible RPDU message types.

3. RP-Originator-Address (RP-OA), the address of the originating SMS-SC in Type 4 LV format, only used in RP-DATA messages. In case of an uplink RP-DATA message the length is set to 0, signifying this field is unused. In the downlink direction the the length is the total length of the value in octets, where the first octet of the value is the address encoding scheme (0xA1 (national) or 0x91 (international) by default), followed by the actual address in semi-octet representation, see Figure D.6. The address encoding scheme can be extended in the future, but the EXT bit should now always be coded as `1`. The maximum value of the length octet is 11, resulting in a maximum address length of twenty digits.

4. RP-Destination-Address (RP-DA), the address of the destination SMS-SC in Type 4 LV format, only used in RP-DATA messages. This field uses the exact same encoding as RP-OA, but is used in the opposite direction. So in downlink messages the length is set to 0, while in uplink messages it specifies the SMS-SC address.

5. RP-Cause (RP-Cause), which is a Type 4 LV format IE with a value part of one or two octets. Included if and only if the RP-MTI is RP-ERROR. The RP-Cause conveys the cause of the error in a single octet, sometimes with an additional octet specifying diagnostics information.

6. RP-User-Data (RP-UD), which is the TPDU in Type 4 LV (when mandatory) or TLV (when optional) format. This part is mandatory in RP-DATA messages and optional in RP-ACK and RP-ERROR messages. The maximum length of the value part of the RP-UD is 231 (RP-DATA) or 232 (RP-ACK and RP-ERROR) octets.

D.3.3 Short Message Transfer Layer and TPDUs

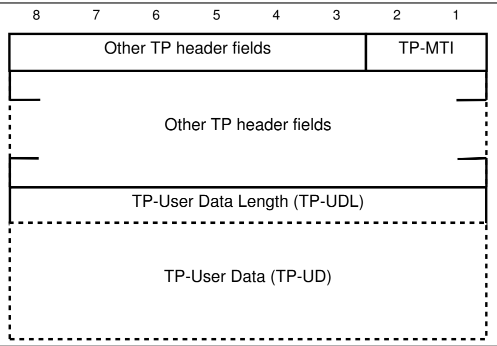
The layer above SM-RL is the Short Message Transfer Layer (SM-TL). Messages on this layer are called Transfer Protocol Data Units (TPDUs) and again contain several fields. The most important of these fields is present in all TPDUs and is called the TP-Message-Type-Indicator (TP-MTI), which is in bits 1 and 2 of the first octet of the TPDU[18]. This field, together with the direction of the message, determines the message type of the TPDU. There are six different message types and the message type determines which other fields are present in the TPDU. The six possible TPDU message types are discussed below:

- SMS-DELIVER, used by the SMS-SC to deliver an SMS to the MS. Encoded as TP-MTI 0x00.
- SMS-DELIVER-REPORT, a report from the MS conveying the positive or negative acknowledgement to an SMS-DELIVER or SMS-STATUS-REPORT message. Encoded as TP-MTI 0x00.
- SMS-SUBMIT, used by the MS to deliver an SMS to the SMS-SC. Encoded as TP-MTI 0x01.
- SMS-SUBMIT-REPORT, a report from the SMS-SC to the MS conveying the positive or negative acknowledgement to an SMS-SUBMIT or SMS-COMMAND message. Encoded as TP-MTI 0x01.
- SMS-STATUS-REPORT, a report by the SMS-SC to the MS conveying the status of the SMS-SC. Encoded as 0x02. Difficult to fuzz because these messages refer to data in a preceding mobile originating transaction.
- SMS-COMMAND, conveying a command from the MS to the SMS-SC. Example commands are to request a status report or delete a previously submitted message. Encoded as 0x02.

As can be seen from the above list only three TP-MTI values are valid, because the meaning of each value depends on the direction of the message. As a result only three different message types can be fuzzed, because the other three are not interpreted as such due to the direction. From those three the SMS-SUBMIT-REPORT and SMS-STATUS-REPORT messages are sent as response to a mobile originating transaction, making them inefficient to fuzz (unless there is a problem with the state machine). SMS-DELIVER messages are sent in a mobile terminating transaction and can thus be fuzzed efficiently.

[18]In the official ETSI documents regarding SMS the bits are numbered from 0 to 7, but in this paper the numbering convention 1 to 8 of the GSM documents, which was introduced in Appendix A, is used.

Figure D.7 Format of SMS TPDUs.



SMS-DELIVER TPDU Most of the remaining fields in TPDUs are only used in specific types of TPDUs. Therefore only the fields useful for fuzzing (i.e. those that are included in SMS-DELIVER messages) are discussed below. An extensive overview of all the TPDU fields and their meaning can be found in Clause 9 of [23].

1. Because the TP-MTI is only two bits long, the remaining bits of the octet are used for various flags. Bit 3 is used for the TP-More-Messages-to-Send (TP-MMS) to indicate if there are more messages to deliver from the SMS-SC to the MS. Bit 4 can be used in certain cases as TP-Loop-Prevention (TP-LP) to prevent infinite looping in case of forwarding or automatic message generation and bit 5 is a spare bit. Bit 6 can be used as TP-Status-Report-Indication (TP-SRI) to request a status report from the receiving entity, while the TP-User-Data-Header-Indicator (TP-UDHI) in bit 7 indicates whether the content of this message, called the TP-User-Data (TP-UD), starts with a header or only contains the actual message. Finally bit 8 in the first octet of the TPDU is used as TP-Reply-Path (TP-RP) which specifies whether or not replies must be routed back via the SMS-SC specified in the RP-OA.

2. The TP-Originator-Address (TP-OA) is an address in an unusual format. Just like the RP-OA it starts with an octet for the length, then an octet for the address encoding scheme, and finally the actual address in semi-octet representation. However, unlike the RP-OA the LI in the TP-OA is not the number of octets of the value (including the address encoding scheme octet), but the number of semi-octets in the actual address (i.e. excluding the address encoding scheme octet). The address specified in this field is the Mobile Subscriber Integrated Services Digital Network number (MSISDN) of the MS that sent the message, i.e. the phone number of the sender.

3. The TP-Protocol-Identifier (TP-PID) is a field that can specify a higher layer (SM-AL) protocol that should get this message. Example protocols are default text, Internet e-mail or fax, although this feature seems to be rarely used.

4. The TP-Data-Coding-Scheme (TP-DCS) which denotes the coding scheme used within the TP-UD of this message. Three main schemes exist, where

each scheme has several flavours that determine how the MS should handle the message exactly (for more on these flavours see [44]). The TP-DCS can also specify that the content is compressed. The main schemes are:

- 7 bit GSM default alphabet, where each character is represented in seven bits
- 8 bit data, used only for transmission of data
- Universal Coded Character Set-2 (UCS-2) which is a 16 bit alphabet allowing for many more different characters.

Because the maximum length of the content is 140 octets, these coding schemes result in a maximum of 160 characters, 140 bytes of data and 70 characters respectively.

5. The TP-Service-Centre-Time-Stamp (TP-SCTS) is a field that consists of seven octets. It represents the data and time that the SMS-SC received the message. The first six octets are the year (last two digits only), month, day, hour, minute and second respectively in semi-octet format. The seventh octet encodes the time zone, also in semi-octet format, in quarters of an hour from GMT. Bit 4 of the time zone octet is used as the sign (`0` for positive, `1` for negative) of the distance from GMT.

6. The TP-User-Data-Length (TP-UDL) specifies the length of the TP-UD of this message. If the TP-DCS is set to the GSM 7 bit default alphabet the length is counted in septets (i.e. characters), in the other two coding schemes it is counted in octets.

7. The last field in SMS-DELIVER messages is the TP-User-Data (TP-UD). This is the actual content that is being sent.

D.3.4 Short Message Application Layer

The Short Message Application Layer (SM-AL) is the highest layer in the SMS stack. Actually the SM-AL is a collective noun for many different protocols, of which one is selected for each SMS message by the TP-PID field in the header of the TPDU. SMS messages can originate from an MS or SMS-SC, which is shown as the two dashed arrows on the SM-AL in Figure D.2 of the SMS stack.

The message transmitted on the SM-AL is the TP-UD. It can contain one or more headers (when the TP-UDHI is set) and have different coding schemes (via the TP-DCS). SMS can be used for many different messages, from simple text to pictures, sounds and even animations. For all but the first headers have to be used to define what type of content it is. Another feature of SMS is to use segmentation to send content longer than the limit of 140 octets. The sender splits up the content over multiple SMS messages and the receiver glues them together again. Message D.1 is a complete example of an SMS-DELIVER message.

Message D.1 Example of a valid SMS-DELIVER message from the MS with Mobile Subscriber Integrated Services Digital Network number “0612” conveying the text “Hello”

Octet	Hex	Bits	Meaning
1	09	`----1001`	PD: SMS
1	09	`-000----`	TI value: 0
1	09	`0-------`	TI flag: direction SMS-SC to MS
2	01	`00000001`	MT: CP-DATA
3	1C	`00011100`	Length of the following RPDU is 28 octets
4	01	`00000---`	Spare bits
4	01	`-----001`	RP-MTI: RP-DATA from SMS-SC to MS
5	34	`00110100`	RP-MR: 52
6	03	`00000011`	Length of the RP-OA is 3 octets
7	A1	`1-------`	RP-OA is not extended
7	A1	`-010----`	RP-OA is a (default) national number
7	A1	`----0001`	RP-OA uses the (default) ISDN numbering plan
8	21	`00100001`	RP-OA: “12”
9	F3	`11110011`	RP-OA: “3”
10	00	`00000000`	Length of the RP-DA is 0 octets
11	14	`00010100`	Length of the following TPDU is 20 octets
12	00	`------00`	TP-MTI: SMS-DELIVER
12	00	`-----0--`	TP-MMS not set, no more messages waiting in the SMS-SC
12	00	`----0---`	TP-LP not set, this message is not spawned or forwarded
12	00	`---0----`	Spare bit
12	00	`--0-----`	TP-SRI not set, no status report requested upon delivery
12	00	`-0------`	TP-UDHI not set, no UDH included in the TP-UD
12	00	`0-------`	TP-RP not set, no reply path specified
13	04	`00001010`	Length of the TP-OA is 4 semi-octets
14	A1	`1-------`	TP-OA is not extended
14	A1	`-010----`	TP-OA is a (default) national number
14	A1	`----0001`	TP-OA uses the (default) ISDN numbering plan
15	60	`01100000`	TP-OA: “06”
16	21	`00100001`	TP-OA:“12”
17	00	`00000000`	TP-PID: 0, no SM-AL protocol specified
18	00	`00000000`	TP-DCS: 0, GSM 7 bit default alphabet
19	11	`00010001`	TP-SCTS: year ’11
20	30	`00110000`	TP-SCTS: month 03
21	41	`01000001`	TP-SCTS: day 14
22	01	`00000001`	TP-SCTS: hour 10
23	45	`01000101`	TP-SCTS: minute 54
24	63	`01100011`	TP-SCTS: second 36
25	80	`10000000`	TP-SCTS: timezone GMT+2
26	05	`00000101`	TP-UDL: length of the TP-UD is 5 septets
27	C8	`11001000`	TP-UD: “H”
28	32	`00110010`	TP-UD: “e”
29	9B	`10011011`	TP-UD: “l”
30	FD	`11111101`	TP-UD: “l”
31	0E	`00000110`	TP-UD: “o”

Appendix E

Call Control fuzzing

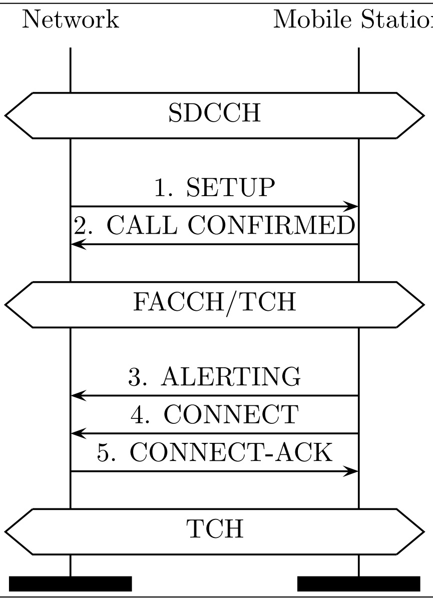
Call Control (CC) is the service in GSM that is used for all messages regarding phone calls, from setting them up to closing them down. Nearly three dozen different MTs are defined for CC and call related Supplementary Services. Unfortunately only few of these messages were implemented in OpenBTS at the time of writing, so not all of them could be fuzzed. In fact only the setup of calls was tested and from this set of messages only the content and not the state machine. We also only tested using early assignment (see below), but the other two channel assignment types are interesting for future work. In this appendix the message sequence of call setup is described, as well as the structure of CC-SETUP messages.

E.1 Call setup message sequence

Three different message sequences are defined for setting up a phone call: late assignment, early assignment and very early assignment. The difference between the sequences is which messages are exchanged over which logical channel. In OpenBTS only early assignment and very early assignment are implemented, with early assignment being the default. In Figure E.1 the (simplified) message exchange for early assignment is given. In the actual message exchange several RR and MM messages are also exchanged to facilitate the channel changes, but these are excluded from this figure.

Whenever a user is getting called on a network using early assignment the network first pages the cell phone of the recipient over the PCH, which responds on the RACH. The response from the network is sent to the phone on the AGCH and both entities tune in to the allocated SDCCH. The network then starts by transmitting the SETUP message. The phone acknowledges this message with a CALL CONFIRMED message, causing the network to allocate a TCH over which the actual conversation will take place. However this channel is first used for control messages (thus as FACCH) starting with an ALERTING message from the phone when it started notifying the user of the incoming call. Once the user accepts the call the phone sends a CONNECT message to the network, acknowledged by the CONNECT ACK. From that point on the TCH is an actual TCH for the voice data.

Figure E.1 Message sequence chart of a Mobile Terminating call setup using early assignment.



E.2 Call setup message structures

During a mobile terminating call setup two messages are sent from the network to the cell phone: the SETUP and CONNECT ACK messages. The CONNECT ACK is the simplest of the two, consisting only of an Layer 3 header with a Protocol Discriminator (PD) of 3, Transaction Identifier (TI) the same as the SETUP message and Message Type (MT) of 0x0F. The SETUP message is a lot more complicated.

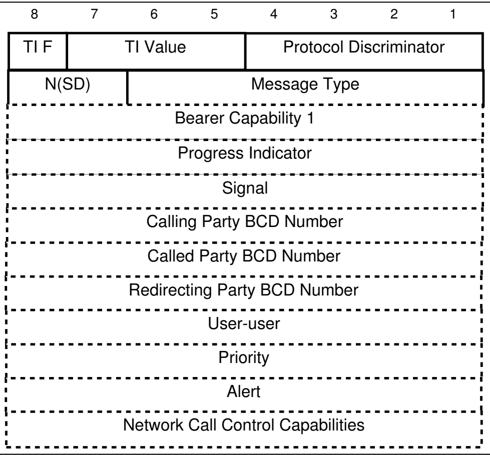
A CC SETUP message consists, like all other Layer 3 messages, of a mandatory header. The PD must be set to 3, the TI must lie between 0 and 6 and the MT must be set to 0x07. Besides this header a SETUP message has an optional body of Information Elements (IEs). Because all elements in the body are optional all have to start with an Information Element Identifier (IEI) and most also include a Length Indicator (LI). IEs that are out of order should be ignored, as should any duplicated IE after the first (unless more than one is allowed).

Due to time constraints only some of the elements in a SETUP message were tested. Below only those elements are described, for further reading on all elements see Clause 9 of [21]. An overview of the SETUP message as it was used in this research can be found in Figure E.2.

1. The Bearer Capability 1 element specifies capabilities supported by the network. It is a type 4 IEs in TLV format with the IEI set to 0x04. The Bearer Capability element is full of different mandatory and optional bit fields, too many to discuss here. For a full specification of all the bit fields see clause 10.5.4.5 of [21].

2. The Progress Indicator IE is a type 4 TLV format element with IEI set to

Figure E.2 The elements of CC SETUP messages that were fuzzed in this research.



0x1E. It can be used to describe an event that happened during a call.

3. The the Signal element is a Type 3 TV format element of length 2 and is used by the network to specify certain tones. The IEI of this element is 0x34.

4. The Calling Party BCD Number is the phone number of the caller. It is encoded just like the RP-OA element for SMS with an LI, numbering plan and the value in semi-octet notation. The IEI is 0x5C.

5. The Called Party BCD Number is the phone number of the callee, starting with IEI 0x5E. It is encoded the same as the Calling Party BCD Number.

6. The Redirecting Party BCD Number is also encoded as the Calling Party BCD Number but uses IEI 0x74. This element is included when the call was redirected from the original callee to this MS.

7. The User-user element is a type 4 TLV format element with IEI 0x7E. This element can be used to transmit user information from the caller to the callee.

8. The Priority IE is a type 1 TV format message with IEI 0x8 that can be used to set the priority of the call.

9. The Alert IE can encode which alerting tone the phone should use to alert the user of the incoming call. This feature does not have to be supported by the phone. The IEI of this element is 0x19.

10. The last fuzzed element is the Network Call Control Capabilities element with IEI 0x2F. This type 4 TLV format message actually only includes a single bit value that specifies whether the network supports multicalls.

Appendix F

Fake base station

The fake base station used in this research was built by Ronny Wichers Schreurs and Fabian van den Broek of the Radboud University Nijmegen. It consists of the following hardware components from Ettus Research LLC[19] and Fairwaves[20] (with price estimation excluding taxes in brackets) and is shown in Figure F.1:

- Ettus USRP-1 (€550)
- Ettus WBX daughterboard (€450)
- Fairwaves ClockTamer-1.2 (€210)
- Ettus LP0926 900 MHz to 2.6 GHz antenna (€30)

The software in use was OpenBTS version 2.6.0 [5], modified to be able to transmit arbitrary SMS and CC messages. We used the 1800 MHz baseband on Absolute Radio Frequency Channel Number (ARFCN) 881, which is free for (unlicensed) use with low power equipment in the Netherlands until at least 2013. For the basestation identification we used the country code of the Netherlands (204) and the unused network code 98 to not interfere with genuine networks. Two cheap prepaid SIM cards were used to connect the phones to the private network.

The USRP-1 was hooked up to an IBM ThinkPad Lenovo R60e via a USB-2 connection. The notebook has the following specifications:

- Intel Core Duo T2300 CPU at 1.66GHz
- 512 MB DDR-2 RAM
- Hitachi 80 GB ATA hard drive
- Ubuntu 10.10 32 bit

A second notebook was used for bookkeeping of the results using Microsoft Excel 2007.

[19]`http://www.ettus.com`
[20]`http://shop.fairwaves.ru`

Figure F.1 The USRP-1 with on top of it the antenna.

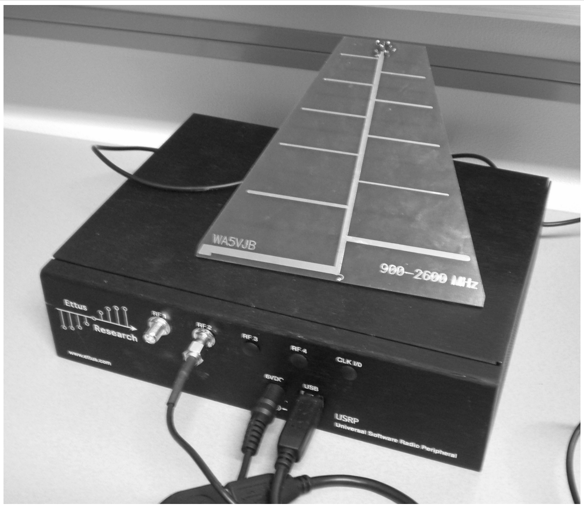
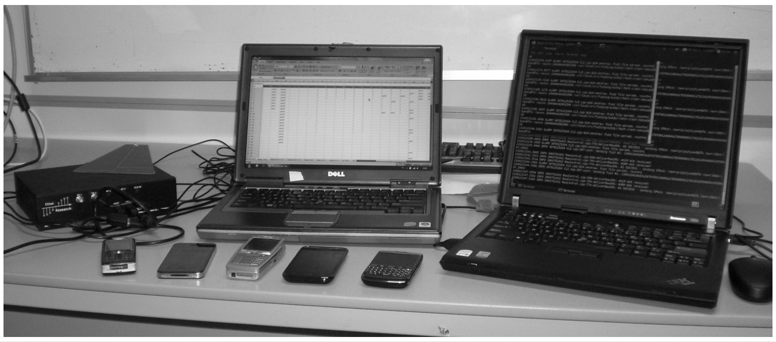


Figure F.2 The setup with on the left the USRP-1, on the right the notebook running OpenBTS and in the centre the notebook used for bookkeeping. In the front are five cell phones.



Appendix G

Details of the fuzzing results

In this appendix the results of the practical part of this research are described in more detail. For all the different cell phones the field(s), introduced in Appendix D, that cause problems are identified and specific values that cause strange behaviour are given. Note that not all of the identified issues happened consistently, which also means that some other inconsistent issues might not have shown itself.

This appendix is organised by cell phone brand and type. But first the annoyance of the icons on specific types of SMS messages is described, since that affects all tested cell phones.

G.1 Icons for special SMS messages

The TP-DCS field can be used to encode a notification to the user that a voice, fax or email message is waiting to be retrieved from the network. This message is easy to send from a fake basestation by encoding the TP-DCS as 0xD8, 0xD9 and 0xDA respectively. These values conform to the specifications and the resulting message is a syntactically and semantically correct message. As was discussed in Chapter 5 getting rid of the icons on the cell phone is rather difficult and the icons themselves can be confusing and annoying.

G.2 HTC

G.2.1 Legend

On the tested HTC Legend a bug exists which makes it possible to prevent the phone from receiving any more SMS messages until the phone is turned off or switched to a different network. This occurs when an SMS message with the RPDU length is set to a low value (we only tested 0 and 1, but possibly other values lower than the actual length are affected as well). Turning the phone off and on, switching to a different network or getting out of range of the BTS or network[21] stop the DoS.

G.3 iPhone

G.3.1 4

The tested iPhone 4 has a bug similar to the HTC Legend where a silent SMS message causes a DoS on receiving SMS messages. In addition, after having received this malicious message it is impossible to switch to a different network. This behaviour is triggered by using the TI extension mechanism in an SMS message. The DoS can be stopped by switching the phone off and on again or getting out of range of the BTS or network.

G.4 Nokia

G.4.1 1100

Nothing extraordinary was found for the Nokia 1100.

G.4.2 2600

The Nokia 2600 has an issue that causes strange but inconsistent behaviour. This happens when playing with the TP-UDHI field. When setting this field to `1` the first octet in the TP-UD is interpreted as the length of the (non-existing) UDH. When this octet is sufficiently large (we found that with 3 octets of data 10 was the minimum value) one of three things happened when sending the message to the Nokia 2600:

1. The message would disappear, i.e. would get acknowledged by the phone to the network but nothing happened on the phone. This happened about 75% of the time.

2. The message would be stored and show part of the phone memory when opened (as described in Section 5.2.3), happening about 20% of the time.

3. The phone would immediately reboot without acknowledging to the network that it received the message, happening the remaining 5% of the time. This reboot would require the user to enter the SIM PIN again before the phone would return to standby mode.

G.4.3 3310 and 3410

We did not identify any problems on the Nokia 3310 and 3410.

[21]We did not have the means to verify whether getting out of range of the BTS is sufficient or that one has to move out of range of the entire network.

G.4.4 6610

On the Nokia 6610 it is possible to silently fill up the phone memory. When playing with the DCS5 thingy. TODO

In addition the phone reboots when sending a valid SMS-STATUS-REPORT or SMS-SUBMIT-REPORT with some of the unknown fields guessed (e.g. the RP-MR of the last SMS submitted by the MS). However, this only happened to the first report of both types, for any subsequent report the phone would return an RP-ERROR message. This indicates that the phone first checks the state before it parses the message and only crashes when actually parsing the message. Finding out what exactly caused this behaviour was out of scope, but would be interesting for future work.

G.4.5 Nokia 7650

The Nokia 7650 would consistently restart when the TPDU length field is set to 0. It did not matter what came after the TPDU length field, the TPDU could be valid or omitted entirely, the phone would reboot either way.

G.4.6 E70-1 and E71-1

No issues were identified on the Nokia E70-1 and E71-1.

G.5 Samsung

G.5.1 SGH-A800

Two different issues on the Samsung SGH-A800 were identified in this research. The first issue allows an attacker to fill up the phone memory by setting the TP-DCS field to 0xD7, 0xDF or 0xEF. Bit 1 and 2 in this message mean that there is an "other message" waiting at the network, while bit 4 sets the flag for this to inactive (`0`) or active (`1`). Bit 3 is spare and should be coded as `0`, but is set to `1` instead. Bits 5 to 8 of this field designate that the TP-UD in this message should be stored on the MS and is encoded in the GSM default 7 bit alphabet (0xD) or in the UCS-2 format (0xE) .

The second issue occurs when the TPDU length field is set to a value that is much larger than the actual length (38 instead of 19 was the minimum we found). The result depends on whether the phone is charging or not. When the phone is not charging it reboots and goes back to standby. However, when the phone is charging it will not reboot, but instead completely shut down. As a result the phone can not receive any calls or SMS messages until the user turns it on and enters the PIN again.

G.5.2 SGH-D500

The Samsung SGH-D500 was the phone with the most robustness issues. Similar to the SGH-A800 setting the TPDU length field to a sufficiently large value causes a reboot. When this happens the vibrate alert of the phone will go off and continue to do so until the phone is completely rebooted and back in standby mode. Unlike the SGH-A800 the phone will always reboot when receiving these messages, even when charging.

Fuzzing on two other fields in SMS messages can also cause a reboot of the SGH-D500. When the RP-DA length is set to a sufficiently large value the phone reboots. Also when the TP-UD is much longer than allowed (tested with 480 7bit characters) the same thing happens[22].

A third issue found with SMS is that it always accepts SMS-SUBMIT-REPORT messages. When received they are stored as a regular SMS message with (presumably) some part of the phone memory in them. The content of the message is really stored as such on the SIM, unlike on the Nokia 2600 where the content would be different every time the message is opened.

The last issue with SMS on the SGH-D500 is that one can silently fill up the SIM card memory. When setting the TP-LP bit to `1` the phone will not notify the user of an incoming message, unless there is already a new message waiting. In that case it will actually notify the user. In either case a message is stored on the SIM that can not be viewed or deleted with this phone, in fact the inbox does not show it at all. However something is stored on the SIM card. When the SIM card is full an empty message is displayed on the phone when another one of these malicious messages is received. The only way to free up memory on the SIM again is to clear it using a different phone.

With CC fuzzing two more values that would cause a reboot were found. When setting the length of the "Redirecting party BCD number" to 0xFF the phone would reboot. With any other values nothing strange happened.

Also when making the CC SETUP message too long (tested with 350 octets in total of duplicate "Calling party BCD number" values) the phone would reboot. Interesting enough this also happened with SMS messages that were too long, but not with arbitrary messages that are too long. So it is not a buffer overflow in the input buffer, but rather two separate problems.

G.5.3 Galaxy S

No issues were found on the Galaxy S.

G.6 Sony Ericsson

G.6.1 T630

The only tested Sony Ericsson phone had several problems as well. Similar to the Samsung SGH-D500 an SMS message with its TP-UD too long or a CC SETUP message that is too long cause it to reboot.

In addition when playing around with the TP-DCS field two different issues were found, both when the first half of the TP-DCS octet is set to 0xF, meaning that the remaining half octet encodes the data coding scheme and message class. When the second half octet is set to 0xA, 0x6 or 0xE the message could not be deleted using the "delete all new messages" function, but could be deleted one by one or via "delete all messages". However, when the second half of this octet is set to 0x3, 0x7, 0xB or 0xF the result was quite different. A message is

[22]In this case OpenBTS stops working as well with an assertion failure.

received but no notification is given to the user. As a result it is possible to fill up the entire phone memory without the user noticing.

The last interesting finding on the T630 is that it accepts all STATUS and SUBMIT report messages. This shows a problem with the state machine, because such a report should only be accepted once after an SMS is submitted. The phone might also be susceptible to invalid fields in these messages, but this was not attempted in this research. This could be interesting for future research.

List of Abbreviations

GCC	Group Call Control	13, 15, 24, 43, 44
GPRS	General Packet Radio Service	43
GPS	Global Positioning System	14
GSM	Global System for Mobile Communication	iii, 1–7, 9–14, 21–27, 30, 33, 36–38, 40, 41, 43, 45, 52, 53, 55, 60, 62, 63, 65, 73
GUI	Graphical User Interface	47
HTML	HyperText Markup Language	51, 52
HTTP	HyperText Transfer Protocol	17–19
IE	Information Element	44–46, 53, 58, 59, 66, 67
IEI	Information Element Identifier	45, 46, 66, 67
IMSI	International Mobile Subscriber Identity	10, 39, 41
IP	Internet Protocol	17
ISDN	Integrated Services Digital Network	10, 61, 63, 78
LCS	Location Service	13, 14, 23, 24, 43, 44
LI	Length Indicator	45, 61, 66, 67
ME	Mobile Equipment	10
MM	Mobility Management	13, 22, 23, 40, 41, 43, 44, 56, 57, 65
MMS	Multimedia Messaging Service	3, 4
MS	Mobile Station	4, 9–11, 13–15, 21–24, 36–41, 55–63, 67, 73
MSISDN	Mobile Subscriber Integrated Services Digital Network number	10, 14, 61, 63
MT	Message Type	22, 42, 44, 45, 57, 58, 63, 65, 66
N(SD)	send sequence number	44
NSS	Network Switching Subsystem	9–11, 41, 55
OS	Operating System	3, 4, 21, 23–27, 29, 33
OSI	Open Systems Interconnection	11, 12, 22
PCH	Paging Channel	39, 56, 65

PD	Protocol Discriminator	42–45, 57, 63, 66
PIN	Personal Identification Number	30, 72, 73
PLMN	Public Land Mobile Network	9–11, 13, 15, 37, 41, 55
PSTN	Public Switched Telephone Network	11
RACH	Random Access Channel	39, 40, 56, 65
RP	Relay Protocol	57–60, 79
RP-Cause	RP-Cause	59
RP-DA	RP-Destination-Address	59, 63, 74
RP-MR	RP-Message-Reference	59, 63, 73
RP-MTI	RP-Message-Type-Indicator	58, 59, 63
RP-OA	RP-Originator-Address	59, 61, 63, 67
RP-UD	RP-User-Data	60
RPDU	Relay Protocol Data Unit	57–59, 63, 71
RR	Radio Resource	13, 22, 23, 40, 41, 43, 44, 65
SACCH	Slow Associated Control Channel	39, 40
SCH	Synchronisation Channel	38
SDCCH	Standalone Dedicated Control Channel	39, 56, 57, 65
SDR	Software Defined Radio	iii, 1, 2, 36
SIM	Subscriber Identity Module	10, 28–32, 41, 69, 72, 74
SM-AL	Short Message Application Layer	5, 55, 61–63
SM-RL	Short Message Relay Layer	55, 58, 60
SM-TL	Short Message Transfer Layer	55, 60
SMIL	Synchronised Multimedia Integration Language	3
SMLC	Serving Mobile Location Centre	14
SMS	Short Message Service	iii, 2–6, 13, 14, 21, 23, 25–33, 35, 36, 39, 43, 44, 55–63, 67, 69, 71–75, 79
SMS-SC	SMS-Service Centre	55–63
SS	Supplementary Service	13, 14, 24, 43, 44, 65
TCH	Traffic Channel	38–41, 65
TDMA	Time Division Multiple Access	38
TI	Transaction Identifier	42–45, 58, 63, 66, 72
TLV	Type-Length-Value	45
TMSI	Temporary Mobile Subscriber Identity	39, 41
TP	Transfer Protocol	57, 59–62, 79, 80
TP-DCS	TP-Data-Coding-Scheme	61–63, 71, 73, 74

TP-LP	TP-Loop-Prevention	61, 63, 74
TP-MMS	TP-More-Messages-to-Send	61, 63
TP-MTI	TP-Message-Type-Indicator	60, 61, 63
TP-OA	TP-Originator-Address	61, 63
TP-PID	TP-Protocol-Identifier	61–63
TP-RP	TP-Reply-Path	61, 63
TP-SCTS	TP-Service-Centre-Time-Stamp	62, 63
TP-SRI	TP-Status-Report-Indication	61, 63
TP-UD	TP-User-Data	61–63, 72–74
TP-UDHI	TP-User-Data-Header-Indicator	61–63, 72
TP-UDL	TP-User-Data-Length	62, 63
TPDU	Transfer Protocol Data Unit	59–63, 73
UCS-2	Universal Coded Character Set-2	62, 73
UDH	User Data Header	57, 63, 72
URI	Uniform Resource Identifier	3
USRP	Universal Software Radio Peripheral	4, 5
USRP-1	Universal Software Radio Peripheral-1	iii, 3, 4, 36, 69, 70
VM	Virtual Machine	26
WAP	Wireless Application Protocol	3
XML	eXtensible Markup Language	51

Bibliography

[1] European Telecommunications Standards Institute, "ETSI History." URL: http://www.etsi.org/WebSite/AboutETSI/AboutEtsi.aspx, 2011. Last accessed 14 July 2011.

[2] Internet World Stats, "Internet World Stats." URL: http://www.internetworldstats.com/stats.htm, March 2011. Last accessed 14 July 2011.

[3] GSM Association, "GSMA History." URL: http://www.gsmworld.com/about-us/history.htm, 2009. Last accessed 14 July 2011.

[4] GSM Association, "GSMA Market Statistics." URL: http://www.gsmworld.com/newsroom/market-data/market_data_summary.htm, 2009. Last accessed 14 July 2011.

[5] "OpenBTS." URL: http://openbts.sourceforge.net/, 2008. Last accessed 14 July 2011.

[6] "OpenBSC." URL: http://openbsc.osmocom.org/trac/wiki/OpenBSC, 2008. Last accessed 14 July 2011.

[7] "OsmocomBB." URL: http://bb.osmocom.org/trac/, 2009. Last accessed 14 July 2011.

[8] K. Nohl, "Attacking Phone Privacy." Presented at Blackhat, Abu Dhabi. White paper URL: https://media.blackhat.com/bh-ad-10/Nohl/BlackHat-AD-2010-Nohl-Attacking-Phone-Privacy-wp.pdf, 2010. Last accessed 14 July 2011.

[9] C. Mulliner and G. Vigna, "Vulnerability Analysis of MMS User Agents," in *22nd Annual Computer Security Applications Conference (ACSAC)*, pp. 77–86, ACM, 2006.

[10] H. Welte, "Using OpenBSC for Fuzzing of GSM Handsets." Presented at 26C3. URL: http://events.ccc.de/congress/2009/Fahrplan/events/3535.en.html, 2009. Last accessed 14 July 2011.

[11] C. Mulliner and C. Miller, "Fuzzing the Phone in your Phone." Presented at Black Hat, USA. URL: http://www.blackhat.com/presentations/bh-usa-09/MILLER/BHUSA09-Miller-FuzzingPhone-PAPER.pdf, 2009. Last accessed 14 July 2011.

[12] C. Mulliner and N. Golde, “SMS-o-Death.” Presented at 27C3. URL: `http://events.ccc.de/congress/2010/Fahrplan/events/4060.en.html`, 2010. Last accessed 14 July 2011.

[13] ETSI, “Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Network architecture,” 3GPP TS 23.002 Version 9.5.0 Release 9, European Telecommunications Standards Institute, 2010.

[14] F. van den Broek, “Catching and Understanding GSM-Signals,” Master’s thesis, Radboud University Nijmegen, 2010.

[15] ETSI, “Digital cellular telecommunications system (Phase 2+); Security-related network functions,” 3GPP TS 43.020 Version 9.1.0 Release 9, European Telecommunications Standards Institute, 2010.

[16] ATIS, “Integrated Services Digital Network – Basic Acces Interface for Use on Metallic Loops for Application on the Network Side of the NT (Layer 1 Specification),” ATIS 0600601.1999(R2009), Alliance for Telecommunications Industry Solutions, 2009.

[17] ISO/IEC, “Information technology – Open Systems Interconnection – Basic Refereence Model: The Basic Model,” ISO/IEC International Standard 7498-1:1994, International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC), 1994.

[18] ETSI, “Digital cellular telecommunications system (Phase 2+); Layer 1; General Requirements,” 3GPP TS 44.004 Version 9.0.0 Release 9, European Telecommunications Standards Institute, 2010.

[19] ETSI, “Digital cellular telecommunications system (Phase 2+); Data Link (DL) Layer General Aspects,” 3GPP TS 44.005 Version 9.0.0 Release 9, European Telecommunications Standards Institute, 2010.

[20] ETSI, “Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications Systems (UMTS); LTE; Mobile radio interface signalling layer 3; General Aspects,” 3GPP TS 24.007 Version 9.0.0 Release 9, European Telecommunications Standards Institute, 2010.

[21] ETSI, “Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications Systems (UMTS); LTE; Mobile radio interface Layer 3 specification; Core network protocols; Stage 3,” 3GPP TS 24.008 Version 9.5.0 Release 9, European Telecommunications Standards Institute, 2010.

[22] ETSI, “Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications Systems (UMTS); LTE; Point-to-Point (PP) Short Message Service (SMS) support on mobile radio interface,” 3GPP TS 24.011 Version 9.0.1 Release 9, European Telecommunications Standards Institute, 2010.

[23] ETSI, “Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Technical realization of the Short Message Service (SMS),” 3GPP TS 23.040 Version 9.3.0 Release 9, European Telecommunications Standards Institute, 2010.

[24] ETSI, "Digital cellular telecommunications system (Phase 2+); Functional stage 2 description of Location Services (LCS) in GERAN," 3GPP TS 43.059 Version 9.0.0 Release 9, European Telecommunications Standards Institute, 2010.

[25] ETSI, "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications Systems (UMTS); LTE; General on supplementary services," 3GPP TS 22.004 Version 9.0.0 Release 9, European Telecommunications Standards Institute, 2010.

[26] ETSI, "Digital cellular telecommunications system (Phase 2+); Group Call Control (GCC) protocol," 3GPP TS 44.068 Version 9.0.0 Release 9, European Telecommunications Standards Institute, 2010.

[27] ETSI, "Digital cellular telecommunications system (Phase 2+); Broadcast Call Control (BCC) protocol," 3GPP TS 44.069 Version 9.0.0 Release 9, European Telecommunications Standards Institute, 2010.

[28] S. Gorbunov and A. Rosenbloom, "AutoFuzz: Automated Network Protocol Fuzzing Framework," *IJCSNS International Journal of Computer Science and Network Security*, vol. 10, no. 8, pp. 239–245, 2010.

[29] Immunity, Inc., "SPIKE fuzzing tool." URL: `http://www.immunityinc.com/resources-freesoftware.shtml`, 2001. Last accessed 14 July 2011.

[30] A2009, "Sulley: Fuzzing Framework." URL: `http://code.google.com/p/sulley/`, 2007. Last accessed 14 July 2011.

[31] M. Zalewski, "mangleme." URL: `http://freshmeat.net/projects/mangleme/`, 2004. Last accessed 14 July 2011.

[32] T. Engel, "S60 Curse of Silence." URL: `http://berlin.ccc.de/~tobias/cursesms.txt`, 2008. Last accessed 14 July 2011.

[33] ETSI, "Digital cellular telecommunications system (Phase 2+); Physical layer on the radio path; General description," 3GPP TS 45.001 Version 9.3.0 Release 9, European Telecommunications Standards Institute, 2010.

[34] ETSI, "Digital cellular telecommunications system (Phase 2+); Mobile radio interface layer 3 specification; Radio Resource Control (RRC) protocol," 3GPP TS 44.018 Version 9.7.0 Release 9, European Telecommunications Standards Institute, 2010.

[35] G. J. Myers, *The Art of Software Testing.* John Wiley & Sons, 1979.

[36] J. W. Duran and S. C. Ntafos, "An Evaluation of Random Testing," *IEEE Transactions on Software Engineering*, vol. 10, no. 4, 1984.

[37] E. J. Weyuker and B. Jeng, "Analyzing Partition Testing Strategies," *IEEE Transactions on Software Engineering*, vol. 17, no. 7, 1991.

[38] W. J. Gutjahr, "Partition Testing vs. Random Testing: The Influence of Uncertainty," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, 1999.

[39] P. Godefroid, “Automated Whitebox Fuzz Testing,” in *Proceedings of the Network and Distributed Systems Security Symposium*, ACM, 2008.

[40] W. E. Howden, “Functional Program Testing,” *IEEE Transactions on Software Engineering*, vol. 6, no. 2, 1980.

[41] D. Aitel, “The Advantages of Block-Based Protocol Analysis for Security Testing,” White paper, Immunity, Inc., 2002.

[42] M. Eddington, “Peach Fuzzing Platform.” URL: `http://peachfuzzer.com/`, 2004. Last accessed 14 July 2011.

[43] M. Sutton, A. Greene, and P. Amini, *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley, 2007.

[44] ETSI, “Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Alphabets and language-specific information,” 3GPP TS 23.038 Version 9.1.1 Release 9, European Telecommunications Standards Institute, 2010.