Security Testing of GSM Implementations

Fabian van den Broek[1], Brinio Hond[2], and Arturo Cedillo Torres[2]

[1] Institute for Computing and Information Sciences,
Radboud University Nijmegen, The Netherlands.
f.vandenbroek@cs.ru.nl
[2] KPMG, {hond.brinio, cedillo.arturo}@kpmg.nl

Abstract. Right after its introduction, GSM security was reviewed in a mostly theoretical way, uncovering some major security issues. However, the costs and complexity of the required hardware prohibited most people from exploiting these weaknesses in practice and GSM became one of the most successful technologies ever introduced. However, there is an enormous amount of mobile enabled equipment out there in the wild, which not only have exploitable weaknesses following from the GSM specifications, but also run implementations which were never security tested. Due to the introduction of cheap hardware and available open-source software, GSM found itself under renewed scrutiny in recent years. Practical security research such as fuzzing is now a possibility.
This paper gives an overview on the current state of fuzzing research and discusses our efforts and results in fuzzing parts of the extensive GSM protocol, which is described in hundreds of large PDF documents and contains many layers and many, often archaic, options. It is, in short, a prime target for fuzzing. We focus on two parts of GSM: SMS messages and CBS broadcast messages.

1 Introduction

GSM saw its first deployment in 1991 in Finland and from there grew out to become one of the dominant communication technologies. GSM can be considered old technology, since there are numerous newer technologies in the GSM family, such as UMTS and LTE, which provide better bandwidth and possibilities for data transfer. However, that does not mean GSM is no longer a critical infrastructure. As of 2013, approximately 7 billion SIM cards are active worldwide, offering subscription services to the GSM family of networks for around 3.4 billion unique subscribers [1]. Even though these subscriptions are partly for other networks, GSM is nearly always the base subscription and almost all equipment supports GSM (or GPRS for mobile Internet equipment). Furthermore, GSM/GPRS coverage is far more extensive than the coverage of the newer protocols and GSM uses less power and is more efficient for voice calls. Also, a lot of machine-to-machine communication relies on GSM/GPRS, such as certain smart meters and traffic lights in South Africa [2,3] as well as railway systems in the European Union [4]. All this has prompted providers to speculate that newer protocols such as LTE will replace their direct predecessor (UMTS), but will still run alongside an active GSM/GPRS network [5]. So, for the foreseeable future, GSM is here to stay. When GSM was first deployed there was some security research, which mostly focused on the specifications and the reverse engineering of the secret and proprietary encryption algorithm [6]. Several weaknesses in GSM where quickly identified, though practical exploits of these weaknesses proved complicated because of all the signal processing involved. This changed around 2010 with the arrival of cheap hardware [7] and open-source software [8] which provided easy access to the GSM spectrum. This immediately led to some high profile attacks, such as the reveal of the Time-Memory Trade-Off tables for breaking GSM's standard encryption [9].

With this new hardware and software it is possible to run your own GSM cell tower to which real phones will connect, since in GSM the network does not authenticate itself to the phones. This opens the possibility to verify the implementations of GSM phones by the technique known as fuzzing. Fuzzing has been used a lot to find security holes on Internet equipment. Thanks to low level access of Ethernet cards it was easy to simply try out all kinds of possible messages, mostly those just outside of the specifications, and see what happens when these are received. Fuzzing mobile phones has mostly happened in the hackers scene of security research, with few academic publications.

The GSM system comprises many entities, such as the mobile phones and cell towers, but also many more back-end components. Our fuzzing research only focuses on mobile phones. Naturally, fuzzing the network components of a GSM network can be very interesting and have a much larger impact, but due to a lack of availability of network components to fuzz, which are not currently running inside an operational GSM network, we limited ourselves to the readily available mobile phones. In this paper we discuss our efforts and results in fuzzing two specific parts of the GSM specification: SMS messages and CBS messages.

The well-known Short Message Service (SMS) was added shortly after the initial release of GSM and the first SMS message was sent in 1992 [1]. 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 nowadays also 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. The current SMS specification is found in [10,11].

The lesser-known Public Warning System (PWS) actually started out as the Cell Broadcast Service (CBS), which was developed in parallel to the SMS service as a response of mobile developers to the competing paging services being offered in 1990. It allows providers to broadcast messages to all phones currently connected to a certain cell, i.e. all phones connected to a single transceiver on a BTS. The original business cases were to provide news, weather and traffic information to mobile users, though this never found any wide spread popularity, leading to both mobile network operators and mobile developers neglecting the implementation of the service in their equipment. However, this service has been gaining importance in the last years due to the need of governments to broadcast information in the event of an emergency to all phones in the vicinity. Several countries define and implement their own warning system that rely on the CBS to deliver emergency information. Due to the diversity of technical specifications of each warning system, ETSI with the aid of the 3GPP consortium developed a standardized system known as Public Warning System (PWS). The initial goal of the PWS was to bring a standard emergency and warning communication infrastructure, as well as specific technical requirements for mobile phones within the European Union. However, due to its standardized nature this system and its accompanying protocols can now be implemented worldwide. This allows roaming users to receive broadcast messages no matter what their location is, as long as they are in a GSM coverage area.

Structure of this paper In Section 2 we discuss the basics of the GSM air interface and provide an introduction into the SMS and CBS protocols. We then discuss fuzzing in general and the specifics of fuzzing mobile phones in Section 3. We then describe our own fuzzing research, together with the practical details and results in Section 4. It is here that we also discuss the related work, for comparison and to attempt to provide an overview of the fuzzing research into GSM up to this point. Finally, Section 5 presents the conclusions and ideas for future work.

2 GSM

The GSM broadband stack is usually described in three layers, where the third layer is again subdivided, as is shown in Figure 1. The first layer, the physical layer, creates a set of logical channels through time division on already divided frequencies. These channels can be used by higher layer functions for many different tasks, as uplink (mobile phone to cell tower), downlink (cell tower to mobile phone) or broadcast (cell tower to all connected phones) communication. These channels can either be a traffic channel, or one of a multitude of control channels. Most control data is transmitted in 184 bit frames which are split up into 4 bursts. These bursts are modulated and transmitted by radio waves.

The signaling protocol used on the second layer, the data link layer, is called LAPDm. The data link layer (and higher layers) is only defined for the signaling channels, not for the speech channels. This is because speech bursts contain no further headers or other meta information, only speech data; during a phone conversation, the traffic on the dedicated speech channels needs no meta information in order to be reconstructed correctly at the receiving end. The LAPDm protocol can provide positive acknowledgement, error protection through retransmission, and flow control.

The third layer is where the match with the OSI model stops. The third layer is subdivided into three layers, of which the last (highest) one is again subdivided into several protocols:

1. Radio Resource management (RR); this concerns the configuration of the logical and physical channels on the air-interface;

2. Mobility Management (MM); for subscriber authentication and maintaining the geographical location of subscribers (mobility management);
3. Connection management (CM); consists of several sublayers itself:
   (a) Supplementary Services (SS); managing all kinds of extra services that are not connected to the core functionality of GSM;
   (b) Short Message Service (SMS); the handling of the SMS messages;
   (c) Call Control (CC); creating and ending telephone calls;
   (d) Locations Services (LCS); location based services for both the user and the provider;

Layer 3 frames consist of a 2 byte header followed by 0 or more Information Elements (IEs). These IEs can be of several different types: T, V, TV, LV and TLV, where the letters T, L and V denote the presence of a Type, Length and Value field respectively. The type field is always present in non-mandatory IEs. Interesting from a fuzzing perspective are those IEs that contain a length field, LV and TLV, even though they are specified as having a standard length, because these are typical places where a programmer might make a mistake in handling data of non-standard length.

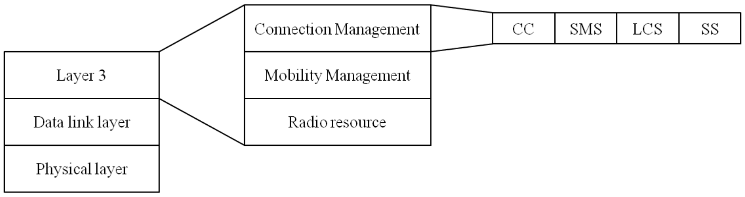
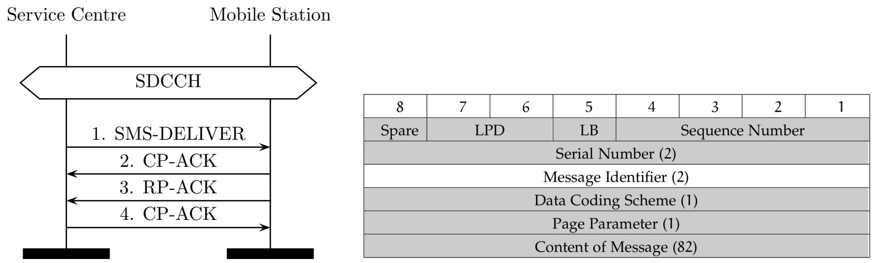


Fig. 1. The layers of GSM

We only fuzz on the third layer of the protocol stack, since this is more likely to trigger observable bugs than fuzzing on the first two layers. That is not to say that the lower layers of the protocol will likely contain less, or less nasty, bugs, they are simply harder to observe.

SMS Before messages can be sent on the SMS sublayer the cell tower needs to notify the mobile phone of an incoming message and set up the channel (Standalone Dedicated Control Channel or SDCCH) with the mobile phone. The delivery of an SMS message then requires four messages exchanged on the SMS sublayer using the SDCCH, as shown in Figure 2(a). The first message is the SMS-DELIVER message sent from the network to the phone. 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 phone 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 connection. If the CP header was correct the MS continues by verifying the RP header and checking if the phone has enough memory



8	7	6	5	4	3	2	1
Spare	LPD		LB	Sequence Number			
Serial Number (2)							
Message Identifier (2)							
Data Coding Scheme (1)							
Page Parameter (1)							
Content of Message (82)							

(a) Message sequence chart of delivering an SMS to a mobile phone

(b) CBS message fuzzing candidates

Fig. 2.

to store the message. If either of those checks fails it returns an RP-ERROR and releases the connection. If both checks succeed the MS returns an RP-ACK with a CP header. The final message is sent by the network when the RP-ACK passes the checks for the CP header.

A schematic overview of a correct SMS-DELIVER message can be found in Figure 4.3, where Figure 3(b) shows the fields we fuzzed of the SMS-DELIVER message. Of the RP-ACK message we fuzzed practically all header fields.

Public Warning System For CBS messages no specific traffic channels need to be set up for mobile phones to receive the transmission. The messages are transmitted on the broadcast channel, to which phones always listen to see if they are still in the same cell. So even if a cell is overloaded with traffic, broadcast messages can still be received by mobile phones. This very feature is what makes them interesting for emergency messages in the first place. Japan's tsunami warning system and the European emergency broadcast (EU-Alert) are examples of implementations of the PWS.

A CBS message is first announced on a broadcast channel and then transmitted in four frames. A schematic representation of a CBS message is shown in Figure 2(b) where all the fields we fuzzed are shown in grey.

3 Fuzzing

Fuzzing is the process of transmitting automatically generated, uncommon inputs to a target with the purpose of creating unexpected behavior. This unexpected behavior is typically something like program crashes or failing built-in code assertions. In contrast to human testing, fuzzing can be largely automated and can find (security) errors that likely won't be triggered during normal use or testing. Fuzzing is already a relatively old testing technique, dating from the end of the 70s and start of the 80s [12]. Fuzzing has evolved over the years into several variants:

1. plain fuzzing,
2. protocol fuzzing, and
3. state-based fuzzing.

Plain fuzzing is the original idea behind fuzzing: simply generating lots of test cases, often with random data, and feed this to the program you are testing. These test cases are usually made by mutating correct inputs and is used to test the error-handling routines. It is a highly portable way of fuzzing, but also provides very little assurance on code coverage.

In protocol fuzzing the test cases are generated based on the specifications, especially on specifications of packet formats. Here the fuzzer will try to choose specific test cases which are likely to provide the largest code coverage based on its knowledge of the specifications. Typically, fuzzers will look at each field which can contain more values than are allowed by the specifications and generate test cases with values on the corner cases, values just over the corner cases and some values way out of the range of what is allowed. This is also called “partition fuzz testing”, since the possible inputs for a field are partitioned (e.g. a half byte which represents an ID and allows for the values 1-12, would give three partitions: 0, 1 to 12 and 13 to 15). These are often not partitions in the mathematical sense, as they need not be disjoint, but the union of all partitions usually do span the entire input space. Note that although these fuzzers are named “protocol fuzzers” and are in fact mostly used to test protocols, they can also be used to test non-protocol implementations, as long as the input has a format to which it should conform.

The first two fuzz variants discussed here try to find errors by changing the content of individual messages. But there is another part that can often be fuzzed: the state machine. Most protocols have some sort of set sequence 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. State-based fuzzers not only change the content, but also the sequence of messages.

An alternative classification is based on the information available at the time of the test case generation: the “black box”, “white box” and “grey box” testing. Black box testing is an ideal scenario for plain fuzzing, since there is no knowledge of what the program is supposed to do. With white box testing, fuzzers can generate test cases based on the source code, to try and cover each branch in the program. Especially in very large code projects, where full code audits or formal methods are infeasible. The access to the source code can also help fuzzers to estimate the test coverage that is achieved with these test cases. For grey box testing some of the internal structure of the test subject is known or guessable. Typically, protocol fuzzing and state-based fuzzing can only be used in at least a grey box test situation, since they require some knowledge of the specifications.

Whichever fuzzing approach is used, fuzzing will usually follow three distinct phases:

1. generating the test cases,
2. transmitting the test cases, and
3. observing the behavior.

The fuzzing approaches discussed above concern the first phase. On traditional computer networks the second phase is trivial. Also, observing the effects after transmitting the fuzz tests is usually easier on traditional computers than on GSM phones, because there is often the possibility of running a debugger, or simply looking for familiar error messages. Fuzzing GSM implementations on baseband chips has many of the same problems as the fuzzing of embedded systems; one cannot easily observe the effect of fuzzed inputs [13].

3.1 Fuzzing GSM phones

There are many different GSM-enabled mobile phones. Mobile phones started out with just the ability to make and receive voice calls, but nowadays phones are available that have a wide range of features and possible connections. It is important to realize that the current market contains a wide variety of GSM-enabled devices, not only of different make and model, but also internally: GSM phones can consist of a single processor, typically cheaper or older GSM models, which runs the GSM protocol stack and a very limited OS for the user interface. The chip running the GSM protocol stack is referred to as the *broadband chip* [14]. More complicated phones run their OS on a separate general purpose processor, named the *application processor*. Both processors can communicate through a variety of protocols, where the application processor uses the baseband processor like a modem. Most modern phones combine the application and baseband processor in a single SoC (System on a Chip).

Fuzzing mobile phones is challenging compared to e.g. fuzzing network cards, mainly because it is hard to observe undefined behavior. Most phones are closed devices without any debugging tools, so it is impossible to, for instance, look at the memory during operation. Also most phones run closed source software. Even Android phones use closed software libraries for low level communication with the baseband chip. This makes it harder to predict where errors will occur. Phones usually have limited interaction possibilities, which makes observation a time consuming effort. Generally there are few alternatives to simply using the phone after a fuzz message and observing whether it shows any undefined behavior, which can lead to false positives. This means internal errors that do not directly lead to observable undefined behavior, may go unnoticed.

The fuzzed messages need to be introduced to the target phones. This can either be done by transmitting them as actual GSM messages to the phones, or by directly inserting them in the phones, for instance by inserting them on the wire between the broadband and application chip. The latter option is cumbersome to use on many different phones and much harder on modern phones with a single SoC, but was the only available option when open source GSM networks were not available [15]. For transmitting the messages there are several options:

1. you could use a running GSM network, either because you happen to have access to commercial GSM network equipment, or by transmitting fuzzed messages from a modified phone to a target phone over the normal network.
2. A more feasible solution is to change existing GSM equipment, such as a femtocell [16] for transmitting fuzz messages, though the success of this method will depend on the success in breaking the femtocell security.
3. Finally there are several open-source projects that allow you to set-up your own GSM network.

Using the existing network severely limits the fields you can fuzz and adapting a femtocell to do the fuzzing could prove unsuccessful, so we chose the third option. The most important of the open-source projects are OpenBTS [8] and OpenBSC [17]. Both systems run on most ordinary PCs and require extra hardware for transceiving GSM signals. OpenBTS is based on the GNU Radio project [18] and is designed to work with the USRP (Universal Software Radio Peripheral), a generic and programmable hardware radio component. The USRP can be modified through the use of daughterboards for specific applications and frequencies. Several versions of the USRP are currently available and a typical setup for a local GSM network costs around $1500,- [7]. OpenBSC runs a basestation controller and therefore interfaces with an actual basestation in order to work. OpenBSC started out as a controller for the Siemens BS11, an actual commercial cell tower of which a small batch became available on eBay, and the nanoBTS from ip.access, a corporate solution miniature cell tower. Both cell towers are hard to obtain, so recently OpenBSC will also work with a new project, OsmoBTS [19], which in turn implements a cell tower on several devices such as the custom made sysmoBTS and even, experimentally, two modified mobile phones.

The OsmoBTS project was not yet available when we started our research, which led us to choose the OpenBTS option, for full control of the GSM air link.

4 Our fuzzing

For GSM all the specifications are openly available, but implementations of the broadband stacks are not. Examining the specifications led to the conclusion that although GSM is a very complicated protocol, there are actually very few state changes in the broadband stacks. This is why we mostly resorted to protocol fuzzing, as will be described in Section 4.1 and 4.2. We attempted some state-based fuzzing on the SMS sublayer by 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. This only showed unexpected results for one phone, the Sony-Ericsson T630, which accepted confirmations of unsent SMS messages, but which did not lead to any exploitable results.

There are several open-source protocol fuzzing frameworks available [20]. However, these frameworks 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. We did end up using one fuzzer, Sulley [21], as the basis for our SMS fuzzer.

4.1 How do we fuzz?

For this research we made our own fuzzer GSMFuzz, for the generation of the fuzz messages. It is a fuzzer, with features designed specifically for GSM, but which nonetheless can be used for other protocols as well. The fuzzer is written in Python (version 2.6) and loosely based on Sulley[21] an open-source fuzzing framework. It has the following features:

- Fuzzing of bit positions within a byte;
- 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 (often used in GSM);
- Hexadecimal output of fuzz cases to a file, which can be used directly in our extended version of OpenBTS.

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).

Figure 3(b) shows the fields we fuzzed in the SMS-DELIVER message and Figure 2(b) shows the fields we fuzzed in the CBS message.

For the transmission of the fuzzed messages we used the open-source OpenBTS software together with a USRP-1 (where the internal clock was replaced with the more precise Fairwaves ClockTamer-1.2) and a collection of (a WBX and two RFX1800) daughterboards. Combining this with two Ettus LP0926 900 MHz to 2.6 GHz antennas yielded a setup of around 1500 €.

We tuned the software to only allow a specific set of SIM cards to connect, but this did not prevent several phones in the surroundings to still connect to our cell. This already shows errors in how phones handle connecting to cell towers. Since we did not want to unintentionally harm the phones of our colleagues, we made a Faraday cage around the whole setup, using chicken wire. With a maze size of 12.5mm, which is smaller than ten times the wavelength of GSM signals on 1800MHz, we managed to keep our GSM broadcasts contained.[3]

The OpenBTS software does not support emergency broadcasts[4], so for these broadcasts we installed a specific branch of an older OpenBTS version (OpenBTS 2.5.4 - SMS-CB), where this service was already implemented.

[3] There was some leakage through the power cord, but not enough to get phones outside of the cage to connect.

[4] Although it is likely that this will be included in a newer release [8].

Table 1. Overview of cell phones tested in this research and the most noticeable results. Legend: I: unremoveable icons, D: DoS message, M: memory bug, N: no notification, R: Reboot S: message handling in violation of specification.

Brand	Type	Firmware/OS	SMS fuzz	Result	CBS fuzz	Result
Apple	iPhone 4	iOS 4.3.3	yes	I,D	no	–
Blackberry	9700	BB OS 5.0.0.743	yes	I	yes	S
HTC	Legend	Android 2.2	yes	I,D	no	–
Nokia	1100	6.64	yes	I	no	–
Nokia	1600	RH-64 v6.90	no	–	yes	S
Nokia	2600	4.42	yes	I,M,R	no	–
Nokia	3310	5.57	yes	I	yes	S
Nokia	3410	5.06	yes	I	no	–
Nokia	6610	4.18	yes	I,N,R	no	–
Nokia	6610	4.74	yes	I,N,R	no	–
Nokia	7650	4.36	yes	I,R	no	–
Nokia	E70-1	3.0633.09.04	yes	I	no	–
Nokia	E71-1	110.07.127	yes	I	no	–
Samsung	SGH-A800	A80XAVK3	yes	I,N,R	no	–
Samsung	SGH-D500	D500CEED2	yes	I,M,R	no	–
Samsung	Galaxy S	Android 2.2.1	yes	I	no	–
Samsung	Galaxy Note	Android 4.1.2	no	–	yes	S
Sony Ericsson	T630	R7A011	yes	I,N	no	–

Having the ability to generate and transmit fuzzed messages, leaves the third stage: the observation. In our SMS fuzzing case, we alternated each fuzzed SMS message with a correct SMS message to see if the phone still responded by acknowledging the correct message. For our CBS fuzzing we simply used most functions of the tested phones after a batch of fuzzed messages, since there is no acknowledgement of received CBS messages.

Table 1 shows the make and models of the phones we used during the fuzzing research. During the research it turned out that many phones did not support the CBS features, so the test set for CBS fuzzing was small.

4.2 Fuzzing results

We now give an overview of some of the most interesting results we found during our fuzzing research, which are also summarized in Table 1. For a complete overview of the exact fuzzing performed and the obtained results we refer to the Master's theses on fuzzing SMS [22] and CBS [23] on which this paper is based.

SMS Fuzzing All tested phones had a funny problem in that they accepted some rarely used SMS variants, such as Fax-over-SMS, which causes strange icons to appear in screens. These SMS variants are so obscure that often the GUI of these mobile phones offered no way to actually access these messages or remove these icons.

More serious issues were that for five out of sixteen phones we found SMS messages that are received and stored by the phones without any notification to the user. This enables attacks of filling up the SMS memory remotely, but all phones notify the user of a full SMS memory. Seven out of the sixteen tested phones could be forced into a reboot, each through a different SMS message.

The Nokia 2600 showed a strange behavior where a particular SMS message would display random parts of the phone memory when opened, instead of the SMS message. Both the iPhone 4 and the HTC Legend could be forced in a DoS state were they silently received a message and afterward could no longer receive any SMS messages, without any notification to the user. Rebooting these phones or roaming to a different network would stop the DoS.

Strangely enough we found no real correlation between specific harmful messages and phone brands. So, a message triggering a reboot in a specific Nokia phone, would have no effect on all other Nokia phones. This is likely due to the large variety in phones as explained in Section 3.1

CBS Fuzzing Our CBS fuzzing research did not reveal any obvious errors such as spontaneous phone reboots. One of the main problems here is that we had no way to tell whether an ignored CBS message was not received by the broadband stack, or that the phone OS did not know how to display it.

The Galaxy Note displayed a fuzzed CBS message which according to the specifications should have been ignored! According to the GSM specification, mobile phones should only receive CBS messages containing Message Identifiers registered in their memory or SIM card. In our initial tests we used the Message Identifier value of 0 and did not register this topic number in the mobile phones. All mobile phones except for the Blackberry received the CBS message. In addition, once we changed the Message Identifier to a value different from 0, all mobile stations did not receive the CBS messages even though this time we did register the topic in the mobile phones.

So we observed that most phones have a lot of trouble to show even correct CBS messages. Since several countries are clearly pushing to get the CBS messages re-supported by phone manufacturers, CBS fuzzing tests should definitely be repeated when wider support is provided.

4.3 Related work

One of the most well-known bugs found in SMS implementations is the "Curse of Silence" found by Thomas Engel, though it is not directly clear if he used any systematic way, such as fuzzing, to find the vulnerability. With this bug certain Nokia phones stopped receiving SMS messages after receiving an email as SMS message[5] with a sender's email address longer than 32 characters [24].

The most prolific academic researcher in the fuzzing of GSM phones is Collin Mulliner [25,15,26,27]. In 2006 he fuzzed the Multimedia Messaging Service

[5] Simply this option of receiving email over SMS is a good illustration of how baroque the SMS standard is!

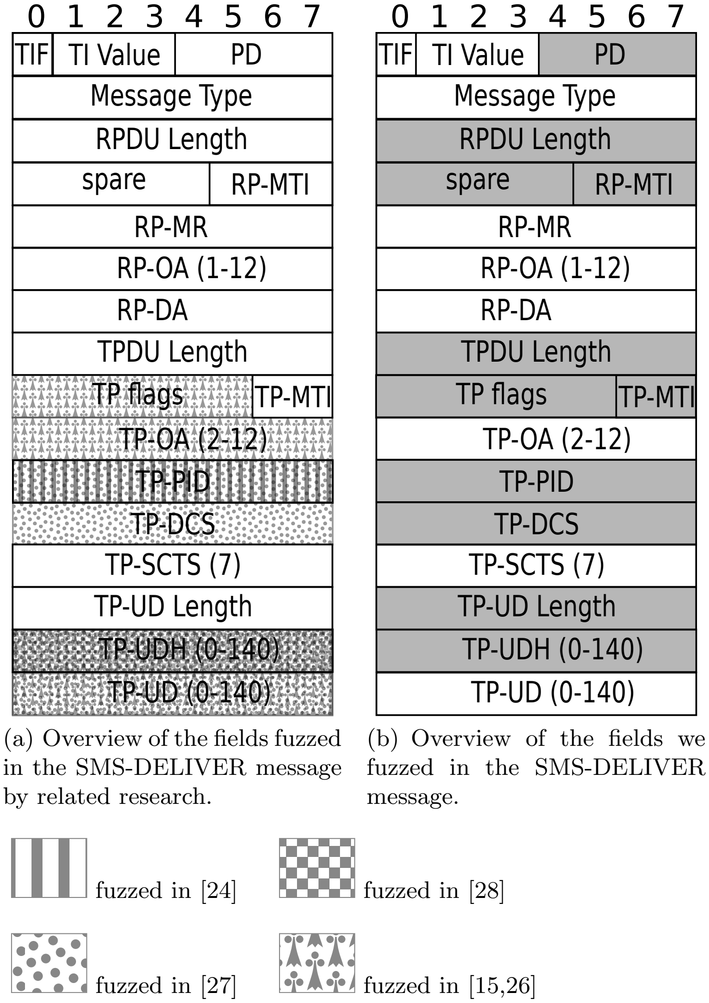
(MMS) feature of GSM [25]. MMS is an extension to SMS for the exchange of multimedia content. When an MMS message is sent the recipient receives an SMS message with a Uniform Resource Identifier (URI) to a server where the MMS content can be retrieved from using the Wireless Application Protocol (WAP). Mulliner et al. used the first method of message transmission described on Page 8, by building a virtual (malicious) MMS server using open source software and retrieving content from it on different cell phones. They found several weaknesses in various implementations, including buffer overflows in the Synchronized 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. Mulliner together with Charlie Miller fuzzed SMS messages on smart phones [15,26] using the second method of transmission. The three smart phones available for this research were an iPhone, an Android Phone and a Windows Phone. An application was developed for each of the three platforms, which makes it possible to directly generate and inject SMS messages into the phones modems. Through this application, the researchers were able to make the device believe that an SMS was just received from the GSM network. Finally, Mulliner and Golde fuzzed the SMS implementation on feature phones [27]. This time they used a rogue cell tower based on OpenBSC, so they used the third method of message transmission. Furthermore they used a J2ME3 application for monitoring on the cell phones. Despite the spectacular title of this publication ("SMS of Death") there was no hard evidence that a fuzzed message caused the death of a phone, since this test was not repeated. The researchers did find DoS attacks for six different popular feature phone brands. They formed SMS messages that can even be sent over commercial (real) networks and will cause the phones to reboot, temporarily losing network connectivity. After consultation with the phone manufacturers Mulliner and Golde did not publish the actual messages that cause the DoS.

The company Codenomicon released a white paper detailing a product that fuzzes SMS messages in order to test the whole network chain for delivery of fuzzed SMS messages [28]. this is targeted towards providers as a tool that can be connected inside the core GSM network.

It is often hard to find out exactly which fields were fuzzed in the studies discussed so far. We have attempted to provide an overview of the fields that have been fuzzed in the SMS-DELIVER message, as far as we can tell from these publications and sometimes through personal communication with the authors. This overview can be found in Figure 3(a). We chose to limit this overview to the one message we also fuzz in our research.

Naturally, fuzzing is not the only way to reveal (security) bugs in the GSM baseband stack. Weinmann et al. decompiled baseband firmware updates from two popular broadband chips and performed a manual code inspection which led to several bugs, amongst which one which led to remote code execution [29,30].

Few researchers have access to the GSM core network. One private security company specializes in fuzz testing GSM core networks and they apparently have a database of possible attacks [31], though the nature of the found vulnerabilities is not public knowledge.



(a) Overview of the fields fuzzed in the SMS-DELIVER message by related research.

(b) Overview of the fields we fuzzed in the SMS-DELIVER message.

Fig. 3. Overview of the fields fuzzed in the SMS-DELIVER message.

5 Conclusions and directions for future work

We have demonstrated that fuzzing is useful to find bugs in the implementation of GSM stacks on mobile phones. Just think on the number of different mobile phones out in the wild and the information we store on them. Setting up a fake base station and sending out malicious messages is, at least for GSM, not hard at all and the potential damages could be enormous.

The wide diversity of phones makes it harder to find a single bug affecting many different mobile phones. Nevertheless, our fuzzing research in GSM has shown several issues with mobile phones. The most important attacks here led to various types of DoS messages which can usually be solved through a reboot of the phone. Some results show clear buffer overflow errors, such as the SMS message which will show random parts of the phone's memory when read on the Nokia 2600. Although it is not immediately clear how to abuse such an error for remote code execution, it is possible that such an attack will be constructed in the future for a popular brand of mobile phones. The CBS service seems to be too poorly supported at the moment to achieve any meaningful fuzzing results, or use it as an emergency broadcast service for that matter.

The hardest part of fuzzing mobile phone implementations is observing the phones behavior, which is hard to automate. Then again, there are not a lot of other options, other than human testing, for security analysis of the closed source broadband stacks on mobile phones. For future research it would be interesting to focus on fuzzing rooted Android devices, where it is possible to run debuggers in the memory to better observe strange behavior.

For now almost all fuzzing research into GSM has focused on fuzzing mobile phones, and then mostly on fuzzing SMS messages, which still leaves many areas open to explore, such as all the other broadcast messages, but also the network side of a GSM network. It would seem logical to assume that the baseband stack on network equipment will contain as many bugs as the stacks on mobile phones, and attacks against the GSM network itself would probably have a much larger impact.

Since the 3G and 4G protocols all have mutual authentication, it is not possible to simply deploy a fake base station in order to fuzz the 3G and 4G baseband stacks. A way around this obstacle would be to use self controlled SIM cards, so you can have your cell tower authenticate to the mobile phone. However, as far as we know there is no open source 3G or 4G cell tower software available yet, so this would require a large amount of work to implement.

Most of our effort came from getting the open source GSM base station up and running. After that implementing the fuzzers was only a few weeks of work. The initial effort to set up a base station and incorporate a fuzzer was substantial, but this solution can now be used to fuzz test any GSM phone on SMS or CBS weaknesses in one and a half hour. So fuzzing is a cost-effective and feasible technique for making implementations of mobile phone stacks more robust and safe.

References

1. GSM-Association: data and analysis for the mobile industry. `https://gsmaintelligence.com/`.
2. UK smartmeter company using GSM/GPRS. `http://www.smsmetering.co.uk/products/smart-meters/gsm-gprs-meters.aspx`.

3. Hack a day website on sim card carrying traffic lights. http://hackaday.com/2011/01/28/sim-card-carrying-traffic-lights/.
4. GSM-R Industry Group. http://www.gsm-rail.com/.
5. News story on the absence of plans to stop 2g services. http://www.computerweekly.com/news/2240160984/Will-the-UK-turn-off-its-2G-networks-in-2017.
6. Marc Briceno, Ian Goldberg, and David Wagner. A pedagogical implementation of the GSM A5/1 and A5/2 "voice privacy" encryption algorithms, 1999. http://cryptome.org/gsm-a512.htm (originally on www.scard.org).
7. Website of the Ettus company, selling USRPs. http://www.ettus.com/.
8. David Burgess. Homepage of the OpenBTS project. http://openbts.sourceforge.net/.
9. Karsten Nohl. Attacking phone privacy. *Blackhat 2010*, 2010. https://srlabs.de/blog/wp-content/uploads/2010/07/Attacking.Phone_.Privacy_Karsten.Nohl_1.pdf.
10. ETSI. *Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Point-to-Point (PP) Short Message Service (SMS) support on mobile radio interface*, 2012. (3GPP TS 24.011 version 11.1.0 Release 11).
11. ETSI. *Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS);Technical realization of the Short Message Service (SMS)*, 2013. (3GPP TS 23.040 version 11.5.0 Release 11).
12. G. J. Myers. *The Art of Software Testing*. John Wiley & Sons, 1979.
13. Rikke Kuipers and Ari Takanen. Fuzzing embedded devices. *GreHack 2012*, page 38, 2012.
14. Harald Welte. Anatomy of contemporary GSM cellphone hardware. *http://laforge.gnumonks.org/papers/gsm_phone-anatomy-latest.pdf*, 2010.
15. Collin Mulliner and Charlie Miller. Injecting SMS Messages into Smart Phones for Security Analysis. In *Proceedings of the 3rd USENIX Workshop on Offensive Technologies (WOOT)*, Montreal, Canada, August 2009.
16. Fabian van den Broek and Ronny Wichers Schreur. Femtocell Security in Theory and Practice. In *18th Nordic Conference NordSec*. Springer LNCS, October 2013.
17. Harald Welte. Homepage of the OpenBSC project. http://openbsc.osmocom.org/.
18. Homepage of the GNU Radio project. http://gnuradio.org/.
19. Harald Welte. Homepage of the OsmoBTS project. http://openbsc.osmocom.org/trac/wiki/OsmoBTS.
20. Collection of fuzzing software. http://fuzzing.org/.
21. Code archive of the sulley fuzzing framework. https://github.com/OpenRCE/sulley.
22. Brinio Hond. Fuzzing the GSM protocol. Master's thesis, Radboud University Nijmegen, Kerckhoff's Master, The Netherlands, 2011.
23. Arturo Cedillo Torres. GSM cell broadcast service security analysis. Master's thesis, Technical University Eindhoven, Kerckhoff's Master, The Netherlands, 2013.
24. Tobias Engel. S60 Curse of Silence. *CCC Berlin*, 2008. http://berlin.ccc.de/~tobias/cos/.
25. Collin Mulliner and Giovanni Vigna. Vulnerability Analysis of MMS User Agents. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, Miami, FL, December 2006.
26. Collin Mulliner and Charlie Miller. Fuzzing the Phone in your Phone. *Black Hat USA*, June 2009.

27. Collin Mulliner, Nico Golde, and Jean-Pierre Seifert. SMS of Death: From Analyzing to Attacking Mobile Phones on a Large Scale. In *USENIX Security Symposium*, 2011.
28. Miia Vuontisjärvi and Tero Rontti. SMS Fuzzing. *Codenomicon whitepaper*, 2011. `http://www.codenomicon.com/resources/whitepapers/codenomicon_wp_SMS_fuzzing_02_08_2011.pdf`.
29. Ralf-Philipp Weinmann. Baseband Attacks: Remote Exploitation of Memory Corruptions in Cellular Protocol Stacks. In *WOOT*, pages 12–21, 2012.
30. Ralf-Philipp Weinmann. The baseband apocalypse. *27th Chaos Communication Congress Berlin*, 2010.
31. P1Security. website detailing a fuzzing product for telco core-networks. `http://www.p1sec.com/corp/products/p1-telecom-fuzzer-ptf/`.