# SMS of Death: from analyzing to attacking mobile phones on a large scale

*Collin Mulliner, Nico Golde and Jean-Pierre Seifert*
*Security in Telecommunications*
*Technische Universität Berlin and Deutsche Telekom Laboratories*
{collin,nico,jpseifert}@sec.t-labs.tu-berlin.de

## Abstract

Mobile communication is an essential part of our daily lives. Therefore, it needs to be secure and reliable. In this paper, we study the security of feature phones, the most common type of mobile phone in the world. We built a framework to analyze the security of SMS clients of feature phones. The framework is based on a small GSM base station, which is readily available on the market. Through our analysis we discovered vulnerabilities in the feature phone platforms of all major manufacturers. Using these vulnerabilities we designed attacks against end-users as well as mobile operators. The threat is serious since the attacks can be used to prohibit communication on a large scale and can be carried out from anywhere in the world. Through further analysis we determined that such attacks are amplified by certain configurations of the mobile network. We conclude our research by providing a set of countermeasures.

## 1 Introduction

In recent years a lot of effort has been put into analyzing and attacking smartphones [18, 20, 24, 21, 22, 23, 46, 45], neglecting the so-called feature phones. Feature phones, mobile phones that have advanced capabilities besides voice calling and text messaging, but are not considered smartphones, make up the largest percentage of mobile devices currently deployed on mobile networks around the world. In comparison, smartphones only account for about 16% of all mobile phones [43]. The lack of security research into the far more popular feature phones is explained by the fact that smartphones share much commonality with desktop computers, and, therefore are easier to analyze. Researches are able to use the same or similar tools that they are already familiar with on desktop computers. Feature phones on the other hand are highly embedded systems that are closed to developers. This results in billions (there are about 4.6 billion mobile phone subscribers [43, 16]) of potentially vulnerable mobile devices out in the field, just waiting to be taken advantage of by a knowledgeable attacker.

In this paper, we investigate the security of feature phones and the possibility for large scale attacks based on discovered vulnerabilities in these devices. We present a novel approach to the vulnerability analysis of feature phones, more specifically for their SMS client implementations. SMS is interesting because it is the feature that exists on every mobile phone. Furthermore, security issues related to SMS messaging can be exploited from almost anywhere in the world, and, thus present the ideal attack vector against such devices. To the best of our knowledge, no attempt has been made before to analyze or test feature phones for security vulnerabilities.

Analyzing feature phones is difficult for several reasons. First of all, feature phones are completely closed devices that do not allow for development of native applications and do not provide debugging tools. Moreover, analyzing the part of the phone that interacts with the mobile phone network is hard since the mobile phone network between us and the target device is essentially a black box. As a consequence, analysis becomes time consuming, unreliable, and costly.

We address these problems by building our own GSM network using equipment that can be bought on the market. *We use this network not only for sending SMS messages to the phones we analyze, but also as an advanced monitoring system. The monitoring system replaces our need for debuggers and other tools that are normally required for thorough vulnerability analysis, but do not exist for feature phones.*

Vulnerability analysis was conducted using fuzzing. We chose fuzzing as the testing technique because we did not have access to source code and reverse engineering a large number of devices is not feasible. Additionally, fuzzing proved to be very efficient since this allowed us to analyze a large amount of mobile handsets with the same set of tests.

So far, we have found numerous vulnerabilities in feature phones sold by the six market leading mobile phone manufacturers. The vulnerabilities are security critical as they can remotely crash and reboot the entire target phone. In the process the mobile phone is disconnected from the mobile network, interrupting any active calls and data connections. Such bugs and attacks have existed before on the Internet, known as Ping-of-Death [6]. We believe this represents a serious threat to mobile telephony world wide.

To complete our research we further analyzed the effect of such attacks on the mobile phone core network. This resulted in two interesting findings. First, the mobile phone network can be abused to amplify our Denial-of-Service attacks. Second, by attacking mobile phones one can attack the mobile phone network itself.

The main contributions of this paper are:

- **Vulnerability Analysis Framework for Feature Phones:** We introduce a novel method to conduct vulnerability analysis of feature phones that is based on a small GSM base transceiver station. We solve the major issue of such analysis: the monitoring for crashes and other unexpected behavior. We present multiple solutions for monitoring such devices while analyzing them. Our method furthermore shows that once a system, such as GSM, becomes partially open, the security of the entire system, including the parts that are still closed, can be analyzed and exploited.

- **Bugs Present in Most Phones:** We show that vulnerabilities exist in most mobile phones that are deployed on mobile networks around the world today. The bugs we discovered can be abused for carrying out large scale Denial-of-Service attacks.

- **Attack Impact:** We show that a small number of bugs in the most popular mobile phone brands is enough to take down a significant number of mobile phones around the world. We further show that bugs present in mobile phones can possibly be used to attack the mobile phone network infrastructure.

The rest of this paper is structured in the following way. In Section 2 we discuss related work and show how our research extends previous work in this area. In Section 3 we explain how we selected our targets for analysis and resulting attacks. In Section 4 we show in great detail how to analyze feature phones for security vulnerabilities. In Section 5 we layout methods to use the vulnerabilities discovered for large scale attacks on mobile communication. In Section 6 we present methods for detecting and preventing the attacks we designed. In Section 7 we briefly conclude.

## 2 Related Work

Related work is separated into four parts. First, smartphone vulnerability analysis. Second, mobile and feature phone bugs, which were all found purely by accident. Third, studies on attacks against mobile phone networks. Fourth, Denial-of-Service (DoS) attacks since we are going to present a large scale mobile phone DoS attack in this paper.

The authors of [24] built a framework for security analysis of Multimedia Messaging Service (MMS) implementations on Windows Mobile based smartphones. Similar research in [23] conducted vulnerability analysis of Short Message Service (SMS) implementations of smartphones. Both used traditional techniques such as debuggers and analysis of crash dumps to catch exceptions generated during fuzzing.

Our work presented in this paper is different, as we do not rely on debugging capabilities provided by the various manufacturers, which mostly do not provide such capabilities at all. Instead we use a small GSM base station to monitor and catch abnormal behavior of the phones by monitoring and analyzing radio link activity. MMS-based attacks that lead to battery exhaustion due to increasing power consumption have been studied in [39]. They utilized the fact that MMS messages use more battery resources because of GPRS and increased CPU usage. However, we did not conduct this kind of analysis since our focus was software bugs in SMS implementations.

Over the last few years a small number of bugs have been discovered by individuals. Most of them have been found by accident. To our knowledge no systematic testing has been conducted. Some examples are: the Curse-of-Silence [44] named bug for Symbian OS that prevents a phone from further receiving any SMS after receiving the *curse* SMS message. The WAP-Push vCard bug on Sony Ericsson phones [33] that caused a target phone to reboot. Some Nokia phones [34] contained a bug that could be abused to remotely crash a phone by sending it a specially crafted vCard via SMS. Some mobile phones produced by Siemens contained a bug [17] that would shutdown the phone when displaying an SMS message that contained a special character. Bugs like these fuelled our research effort since we believed that most phones contain similar bugs. A large number of similar issues in an exploit arsenal can likely be used to carry out attacks against a bigger percentage of mobile phone users around the world.

Enck et al. show in [47] that SMS messages sent over the Internet can be used to carry out a Denial-of-Service attack against mobile phone networks. The attack focused on blocking the mobile network's control channels, therefore, no more calls could be initiated. Solutions against this type of resource consumption attack are investigated in [37]. However our attacks, described in this paper, are not based on attacking the radio link (the control channel) in any way. We attack the handsets directly without targeting the control channel. A study on the capabilities of mobile phone botnets [36] shows that these could be used to carry out DoS attacks against a mobile network. The attack works by overloading the Home Location Register (HLR) by triggering large amounts of state changes by zombie phones. However, in this paper we show that one can achieve a similar kind of DoS attack against an operators network by disconnecting large amounts of mobile phones from the network. The difference to the botnet approach is that we do not need to have control over the zombie phones in the first place. We can remotely force them to reboot and disconnect and re-authenticate to the network and thus cause a higher load on the network core infrastructure.

Denial-of-Service attacks such as the one presented in this work have been studied in a wide area. Attacks ranging from the Web to DNS [38]. More interesting in our context are attacks that disable real-world systems and processes such as emergency services [29] (although just as a side effect) or even postal services [40].

Essentially the work presented in this paper is different in many aspects. We focus on feature phones because feature phones are much more popular than smartphones. Therefore, attacks against feature phones have a larger global impact. In this work we present a security testing framework for analyzing SMS implementations of any kind of mobile phone. We used this framework to analyze feature phones of the most popular manufacturers in the world, as shown in Section 3. We also performed this type of analysis because it has not been done in the past, even though these devices are widely deployed.

## 3 Target Selection

To achieve maximum impact with an attack, it makes sense to target the most popular devices. We determined that feature phones are the dominant type of mobile phones. They account for 83% of the U.S. mobile market [10], smartphones in comparison just make for 16% of all mobile phones world wide [43]. We acknowledge that today smartphone sales are rising very fast, but feature phones still dominate when it comes to deployed devices in the field.

Most of the definitions of the term *feature phone* are a bit fuzzy. A loose definition of the term is: every mobile phone that is neither a dumb phone nor a smartphone is considered a feature phone. Dumb phones are phones with minimal functionality, often they only support voice calls and sending SMS messages, just basic functionality. Feature phones have less functionality than smartphones but still more than dumb phones. Feature phones have proprietary operating systems (firmware) and have additional features (thus the term feature) such as playing music, surfing the web, and running simple applications (mostly J2ME [41]). Despite this lack of functionality (compared to smartphones) they are quite popular because they are cheap and offer long battery life.

Technically interesting is the fact that feature phones are based on a single processor that implements the baseband, the applications, and user interface. Smartphones usually have a dedicated processor for the baseband. The consequence of this is that a simple bug on a feature phone may bring down the complete system.

Mobile phones are produced by many different manufacturers that all have their own OS, therefore, targeting a single one of them will not result in global effect. Since we can not simply target all mobile phone platforms we have to select the few ones that have enough market share to be of global relevance.

To determine the major mobile phone manufacturers we analyzed various market reports: World wide [42] and European [31] market share. Market shares in the United States [28] and in Germany [27]. In the Appendix of this paper we include a table containing the raw numbers we gathered from the various market reports.

Through this analysis we got a clear picture about the top manufacturers. These are `Nokia`, `Samsung`, `LG`, `Sony Ericsson`, and `Motorola`. We further chose to add `Micromax` [4] to the list of interesting mobile phone manufacturers because we read [9] that they are the third most popular brand of mobile phones in India.

## 4 Security Analysis of Feature Phones

Analyzing feature phones for security vulnerabilities is hard for several reasons. There is no access to source code of the OS and applications. There are no existing native-SDKs, therefore, there is no way to run native code on the device and further no access to a debugger. JTAG-based debugging is also no option since not all devices have JTAG enabled. Furthermore, deeper knowledge of the hardware and software is required in order to use JTAG debugging in a meaningful way.

Because of these reasons we choose to conduct fuzz-based testing. The testing is carried out on our own GSM network. In order to monitor for misbehavior, crashes, and to find the related bugs, we designed our own monitoring system. Throughout this section we will first describe the setup of our GSM network. Followed by the way we send SMS messages in this setup. Then we will describe our novel monitoring setup. The final part of the section will discuss test cases and the resulting bugs that were discovered throughout this work.

## 4.1 Network Setup

Since we want to send large amounts of SMS messages we decided to build our own GSM network rather than sending SMS messages over a real network. On the one hand this has the advantage of not costing any money and on the other hand we do not risk to interfere with the telecommunication networks. We want to avoid crashing the operator's network equipment by either content or quantity of SMS messages. Having our own network assures reproducible results because we have control of the entire system and are able to quickly find parameters that cause unexpected results. Analysis over a real operator network would only leave us with the possibility of guessing in many cases. In addition, the delivery of SMS messages is much faster on our small network compared to a production setup of a mobile operator.

On the hardware side we decided to use an ip.access nanoBTS [32], which is a small, fairly cheap (about 3500 Euro) GSM Base Transceiver Station (BTS) that provides an A-bis over IP interface. The A-bis interface is used to communicate between the BTS and the Base Station Controller (BSC). The BSC part of our setup is driven by OpenBSC [30]. OpenBSC is a Free Software implementation of the A-bis protocol that implements a minimal version of the BSC, Mobile Switching Center (MSC), Home Location Register (HLR), Authentication Center (AuC) and Short Message Service Center (SMSC) components of a GSM network. Figure 1 shows a picture of our setup.

Figure 1: Our setup: A laptop that runs OpenBSC and the fuzzing tools, the nanoBTS, and some of the phones we analyzed.

As GSM operates on a licensed frequency spectrum we had to carry out our experiments in an Faraday cage.

Utilizing this setup we are able to send SMS messages to a mobile phone. OpenBSC allows us to either send a text message from its telnet interface to a subscriber of our choice or it processes an SMS message that it received Over-the-Air in a store and forward fashion. As we later see the existing interface is not feasible for fuzzing since we need the ability to closely control all parameters in the encoded SMS format as well as a way to inject binary payloads.

Using a mobile phone to inject SMS messages into the network is not an option as this would be very slow as we show later. Instead we built a software framework based on a modified version of OpenBSC that allows us to:

- Inject pre-encoded SMS into the phone network
- Extensive logging of fuzzing related feedback from the phone
- Logging of non-feedback events, i.e. a crash resulting in losing connection to the network
- Automatic detection of SMS that caused a certain event
- Process malformed SMS with OpenBSC
- Smart fuzzing of various SMS features
- Ability to fuzz multiple phones at once
- Sending SMS at higher rate than on a real network

The format of an SMS [15] differs depending on whether the message is `Mobile Originated (MO)` or `Mobile Terminated (MT)`. This is mapped to the two formats `SMS_SUBMIT` (MO) and `SMS_DELIVER` (MT). In a typical GSM network, shown in Figure 4, an SMS message that is sent from a mobile device is transferred Over-the-Air to the BTS of an operator in SMS_SUBMIT format. Every BTS is handled by a Base Station Controller (BSC) that is interacting with a Mobile Switching Center (MSC), which acts as the central entity handling traffic within the network. The MSC relays the SMS message to the responsible Short Message Service Center (SMSC), which is usually a combination of software and hardware that forwards and relays messages to the destination phone or other SMSCs (in case of inter-operator messages or an operator with multiple SMSCs). In our setup OpenBSC acts as BSC, MSC, and SMSC. During the final transmission to the destination the SMS will get converted to SMS_DELIVER, this is taken care of by OpenBSC. Both formats are similar and no field that is subject to our fuzzing is lost. SMS_SUBMIT only contains the destination number and since SMS works in a store-and-forward fashion, the destination address is replaced with the sender number on the final transmission to the destination. SMS_DELIVER does not include the destination number but instead relies on an existing channel to the phone (after the phone has been paged). For this reason we utilize the SMS_SUBMIT format when injecting messages.

### 4.2 Sending SMS Messages

OpenBSC itself does not provide an interface to submit pre-encoded SMS messages to the network, but only an interface to submit text SMS messages that are then converted into the corresponding encoding. We added a new interface to OpenBSC that allows us to submit SMS messages directly in SMS_SUBMIT format. These messages are inserted into a database that is used by OpenBSC as part of the SMSC functionality. In our version not only the parsed SMS values are stored, but also the complete encoded message for easy reproducibility. Modifying the existing text message interface to be capable of handling binary encoded SMS messages proved to be infeasible. Messages submitted over this interface are instantly transmitted to the subscriber if he is attached to the network. This means opening a channel, initiating a data connection, sending the message and tearing down the connection. This works, but is very slow and takes about seven seconds per message. This is also the reason why we did not want to use a mobile phone to send our fuzz-messages in the first place. Our method of injecting messages is much faster. Prior to testing we use our new interface to inject thousands of messages into the SMSC database. Next, we send these messages. Ideally, this only opens a channel once and sends all SMS messages (pending delivery) to the recipient and then closes the connection. This greatly improves the speed at which we can fuzz since the actual message transfer only takes about one second.

In essence we removed the sending mobile phone and replace it with a direct interface to the network. This way it was not necessary to modify the target mobile phone in any way.

### 4.3 Monitoring for Crashes

In fuzz-based testing, monitoring is one of the essential parts. Without good monitoring one will not catch any bugs.

OpenBSC itself already has an error handler that takes care of errors reported from the phone, which we modified to fit our fuzzing case. The default error handler does not differentiate between errors and is not taking the cause of an error into account. It simply stops the SMS sending process in case of an error. The only exception is a `Memory Exceeded` error, which causes OpenBSC to dispatch a signal handler to wait for an SMMA signal (released short message memory) indicating that there is enough space again.

The mobile phone as well as the MSC are usually divided into separated layers for transferring and processing a message. As shown in Figure 2 they consist of a Short Message Transport Layer (SM-TL), Short Message Relay Layer (SM-RL) and the Connection Sublayer (CM-Sub). The SM-TL [13] receives and relays messages that it receives from the application layer in TPDU form (Transport Protocol Data Unit). This is the original encoding form that we describe later in this paper. The message is passed to the SM-RL to transport the TPDU to the mobile station. At this point the TPDU is encapsulated as an RPDU. As soon as a connection is established between the mobile station and the network the RPDU is transferred Over-the-Air encapsulated in a CP-DATA unit that is part of Short Message Control Protocol (SM-CP). Both sides communicate via their CM-Subs with each other. The CM-Sub on the phone side will unpack the CPDU and forward the encapsulated TPDU to the Transport Layer using an RP-DATA unit. At this point the mobile phone stack has already performed sanity checks on the content of the SMS and parsed it. The resulting reply, passed to CM-Sub, will include an acknowledgement of the SMS message and it will then be passed to the higher layers. From there it will end up in the user interface or an error message is encapsulated and sent back to the network. For our monitoring we need to log these replies carefully to observe the status of the phone.

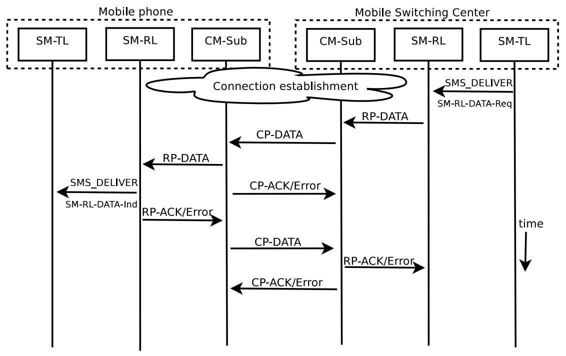

Figure 2: Mobile terminated SMS

From the wide variety of error messages a phone can reply to a received SMS message (defined in [14]), we observed during our fuzzing experiments that all of the tested phones either reply with a `Protocol Error` or `Invalid Mandatory Information` message in the case of a malformed message. These two responses besides the memory error have been the only errors that we observed in practice. We added code to flag such an SMS message as invalid in the database and continue delivering the next SMS that has not been flagged as invalid. OpenBSC would otherwise continue trying to retransmit the malformed SMS message and thus block further delivery for the specific recipient.

SMS messages are usually sent over a SDCCH (Standalone Dedicated Control Channel) or a SACCH (Slow Associated Control Channel). The details of such a channel are not important for the scope of this paper. However the use of such a logical channel is an important measurement to detect mobile phone crashes. Such a channel will be established between the BTS and the phone on the start of an SMS delivery by paging the phone on a broadcast channel. As we explained earlier, we only open the channel once and send a batch of messages using this one channel. The channel related signaling between the BSC and the BTS happens over the A-bis interface over highly standardized protocols. We added modifications to the A-bis *Radio Signaling Link* code of OpenBSC that allows us to check if a channel tear down happens in a usual error condition, log when this happens and which phone was previously assigned to this channel.

So while we lack possibilities to conduct traditional debugging methods on the device itself we can use the open part - OpenBSC - to do some debugging on the other end of the point-to-point connection.

The difference to traditional debugging techniques is that we are mostly limited towards noticing an error condition and monitoring the impact of such an error. We are not able to peek at register values and other software related details of the phone firmware. However, it is enough to be able to reliably detect and reproduce the error. Using this method it also possible to find code execution flaws. However exploiting them and getting to know the details about the specific behavior requires the effort of reverse engineering the firmware for a specific model. We try to avoid such a large scale test of phones but these bugs are a good base for further investigations such as reverse engineering of firmware.

In the next step we have written a script that parses the log file, evaluates it and takes actions in order to determine which SMS message caused a problem.

When delivering an SMS message to a recipient phone under the assumption that it is associated with the cell in practice three things can happen. Either the message is accepted and acknowledged, it is rejected with a reason indicating the error, or an unexpected error occurs. Such an unexpected error can be that the phone just disconnected because it crashed or due to other reasons the received message is never acknowledged. In the latter case, OpenBSC stores the SMS message in the database, increases a delivery attempt counter and tries to retransmit the SMS message when the phone associates with the cell again. For our fuzzing results this means that this method detects bugs in which the SMS message either results in a phone crash after it accepted the message or already during receiving it in which it will never be acknowledged and OpenBSC continuously tries to deliver the SMS message.

Detecting the SMS message that caused such an error condition then is fairly simple. Our script checks the error condition and if it occurred because of the loss of a channel it first looks up the database to find SMS messages that have a delivery count that is bigger or equal to one and the message is not marked as sent (meaning it was not acknowledged). In this case we can with a high probability say that the found SMS message caused the problem. If there is no message the script checks which messages have been sent in a certain time interval around the time of the log event. During our testing we decided that a one minute time interval works well enough to have a fairly small subset of candidate SMS messages that could have caused a problem. Figure 3 shows the logical view of our monitoring setup.

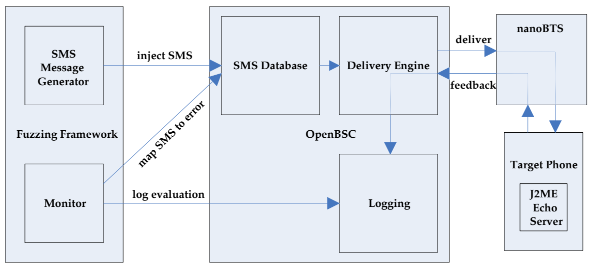

Figure 3: Logical view of our setup.

## 4.4 Additional Monitoring Techniques

In addition to the aforementioned OpenBSC setup we have developed more methods for monitoring for abnormal behavior.

**Bluetooth:** Bluetooth can be used to check if a device crashes or hangs. Our monitor script connects to the device using a Bluetooth virtual serial connection (RFCOMM) by connecting to the RFCOMM channel for the phone's dial-up service. The script calls `recv(2)` and blocks since the client normally is supposed to send data to the phone. When the phone crashes or hangs, the physical Bluetooth connection is interrupted and recv(2) returns, thus signaling us that something went wrong.

**J2ME:** Almost every modern feature phone supports J2ME [41] and this is providing us with the only way to do measurements on the phone since they do not run native applications. Applications running on the mobile phone can register a handler in an SMS registry similar to binding an application to a TCP/UDP port. SMS can make use of a User Data Header [13] (UDH) that indicates that a certain SMS message is addressed to a specific SMS-port. When the phone receives a message this header field will be parsed and the message is forwarded to the application registered for this port. Our J2ME application that is installed to the fuzzed phone registers to a specific port and receives SMS messages on it. For each chunk of fuzzed SMS messages we inject a valid message that is addressed to this port. The application then replies with an SMS message back to a special number that is not assigned to a phone. Figure 3 shows this as the J2ME echo server. The message is just saved to the SMS database. This allows us to easily lookup the count of SMS messages for this special number in the database and check if it increased or not. If not, it is very likely that some odd behavior was triggered. This kind of monitoring is useful to identify bugs that block the phone from processing received messages such as those described in [44].

## 4.5 SMS_SUBMIT Encoding

The SMS_SUBMIT format as defined in [13] consists of a number of bit and byte fields, the destination address, and the message payload. Below we briefly describe the parts the are important for our analysis. We included a diagram of the structure of an SMS_SUBMIT message in the Appendix.

*TP-Protocol-Identifier* (1 octet) describes the type of messaging service being used. This references to a higher layer protocol or *telematic interworking* being used. While this is included in the specifications, we believe that these interworkings are mostly legacy support and not in use these days. This makes it an interesting target to study unusual behavior.

*TP-Data-Coding-Scheme* (1 octet) as described in [12] indicates the message class and the alphabet that is used to encode the *TP-User-Data* (the message payload). This can be either the default 7 bit, 8 bit or 16 bit alphabet and a reserved value.

The *TP-User-Data* field together with the *TP-Protocol-Identifier* and the *TP-Data-Coding-Scheme* are the main targets for fuzzing. The receiving phone parses and displays the message based on this information.

However these fields are not enough to cover the complete range of possible SMS features. If the *TP-User-Data-Header-Indicator* bit (one of the earlier mentioned bit fields) is set this indicates that *TP-User-Data* includes a UDH.

The UDH is used to provide additional control information like headers in IP packets. It can hold multiple so called Information Elements [15] (IEI), for example elements for port addressing, message concatenation, text formatting and many more. IEIs are represented in a simple type-length-value format. We included an example UDH with multiple IEIs in the Appendix.

## 4.6 Fuzzing Test-cases

We have implemented a subset of the SMS specification as a Python library to create SMS PDUs (Protocol Data Unit) and used this to develop a variety of fuzzers. This includes fuzzers for vCard, vCalendar, Extended Messaging Service, multipart, SIM-Data-Download, WAP push service indication, flash SMS, MMS indication, UDH, simple text messages and various others fuzzing only single fields that are part of a specific SMS feature. Some of these features can also be combined. For example most of the features can either consist of single SMS message or be part of a multipart sequence by adding the corresponding multipart UDH.

For the scope of this paper we focused on fuzzing multipart, MMS indication (WAP push), simple text, flash SMS, and simple text messages with protocol ID/data coding scheme combinations. These test cases cover a wide variety of different SMS features.

**Multipart:** SMS originally was designed to send up to 140 bytes of user data. Due to 7-bit encoding it is possible to send up to 160 bytes. However various SMS features rely on the possibility to send more data, e.g. binary encoded data. Multipart SMS allow this by splitting payload across a number of SMS messages. This is achieved by using a multipart UDH chunk (IEI: 0, length: 3). This UDH chunk comprises three one byte values. The first byte encodes a reference number that should be random and the same in all message parts that belong to the same multipart sequence. Based on this value the phone is later able to reassemble the message. The second byte indicates the number of parts in the sequence and the last byte specifies the current chunk ID. By fuzzing these three values we were mainly looking for abnormal behavior related to combinations of the current chunk ID and the number of chunks in a sequence. For example missing chunk and chunk IDs higher than the number of total chunks.

**MMS indication:** When a subscriber receives an MMS (Multimedia Messaging Service) message an MMS notification indication message [48] is sent to him. This MMS indication is in fact a binary encoded WAP-push message sent via SMS. The notification contains multiple variable length fields for subject, transaction ID and sender name. There are no length fields for these values. They are simple zero terminated hex strings. An MMS indication message can also consist of multipart sequences. Therefore, our fuzzing target were the variable length field values included in the message seeking for classic issues like buffer overflow vulnerabilities.

**Simple text:** Implementations of decoders for simple 7 bit encoded SMS often work with a GSM alphabet represented for example with an array. The decoder first needs to unpack the 7 bit encoded values and convert them to bytes. After this step it can lookup the character values in the GSM alphabet table. Our fuzzers mixed valid 7 bit sequences with invalid encodings that would result in no corresponding array index. This could trigger all kinds of implementation bugs but most noteworthy out of bounds access resulting in null pointer exceptions and the like.

**TP-Protocol-Identifier/TP-Data-Coding-Scheme:** The combination of both of these fields defines how the message is displayed and treated on the phone. Both of these fields are one byte values and also cover several rather unpopular features and reserved values. With fuzzing combinations of these values with random lengths of user data payload we were aiming for odd behavior and bugs in code paths that are seldom used by normal SMS traffic.

**Flash SMS:** Flash messages are directly displayed on the phone without any user interaction and the user can optionally save the message to the phone memory. Our observations made it clear that often the code that renders the flash SMS message on the display is not the same as the one that displays a normal message from the menu. Therefore, it can be prone to the same implementation flaws as simple text messages. Additionally, flash SMS can consist of multipart chunks and there are several combinations of TP-Protocol-Identifier and TP-Data-Coding-Scheme that cause the phone to display the SMS as flash message. Our flash SMS fuzzers aim to cover a combination of all of the above possible implementation weaknesses.

## 4.7 Fuzzing Trial

After each fuzzing-test-run we evaluate the log generated by our monitoring script. All of the bugs described later in this paper were triggered by one or very few SMS messages and reproducing problems from log entries was rarely problematic. However, during our fuzzing studies we stumbled across various forms of strange behavior. Problems we faced included non-standard conforming message replies and various kinds of weird behavior. Some phones were not properly reporting memory exhaustion. Others did not notice free memory until a reboot. Some did not display a received SMS message on the user interface which made it hard to tell if the phone accepted a message or silently discarded it on the phone. Almost every phone we fuzzed needed a hard reset at some point because it became simply unusable for unknown reason, the mass of messages or a specific SMS needed to be deleted from the SIM card using another phone. One of the biggest issues we came across was that very few manufacturers' hard reset actually restored the phone to an initial factory state. From what we know this is done as a feature for customers in order to ensure no personal data is lost. The behavior also differed between phones of the same manufacturer. When testing a bug on the Samsung B5310 it was always sufficient to remove the offending SMS message from the phone's SIM card while the Samsung S5230 needed an additional hard reset. Understanding such issues proved to be extremely time-consuming. However, it is worth noting that purging a phone of all personal information can prove to be nearly impossible for a user. This can become an issue whenever a user plans to sell a used handset to a third party.

## 4.8 Results

During our fuzz-testing we discovered quite a few bugs that lead to security vulnerabilities. The bugs mostly lead to phones crashing and rebooting, which disconnected the phones from the mobile network and interrupted ongoing voice calls and data connections. Our testing even resulted in two *bricked phones* that could no longer be reset and brought back into working order. We did not investigate the bricking in-depth because this would have gotten quite costly. Furthermore, some of the phones crash during the process of receiving the SMS message, and, therefore, fail to acknowledge the message thus causing re-transmission of the SMS message by the network.

Below we present some of the bugs we discovered on each platform. In most cases we fuzzed only one phone from each platform and later only verified the bugs on other phones we had access to. This is expected because most manufacturers base their entire product line on a single software platform. Only customizing options such as the user interface depending on the hardware of a specific device.

We reported all bugs to the manufacturers including full PDUs in order to verify and reproduce them. The feedback we received indicates that the bugs are present in most of their products based on their feature phone platforms. So far we have not received any information about fixes or updates.

**Nokia S40:** On our test devices `6300, 6233, 6131 NFC, 3110c` we found a bug in the flash SMS implementation. The phones run different versions of the S40 operating system, the oldest of which was over 3 years older than the newest. The manufacturer confirmed that this bug is present in almost all of their S40 phones. By sending a certain flash SMS the phone crashes and triggers the "Nokia white-screen-of-death". This also results in the phone disconnecting and re-connecting to the mobile phone network. Most notably, the SMS actually never reaches the mobile phone. The phone will crash before it can fully process and acknowledge the message. On the one hand this has the side effect that the GSM network performs a Denial-of-Service attack for free as it continuously tries to transmit the message to the phone. On the other hand this has a side effect on the phone since there seems to be a watchdog in place that is monitoring such crashes. This watchdog shuts down the phone after 3 to 5 crashes depending on the delay between the crashes.

**Sony Ericsson:** Our test devices `W800i, W810i, W890i, Aino` running OSE have a problem similar to the Nokia phones. When combining certain payload lengths together with a specific protocol identifier value it is possible to knock the phone off the network. In this case there is no watchdog, but one SMS message is enough to force a reboot of the phone. As in the case of the Nokia bug, this SMS message will never be acknowledged by the phone. To get an idea on how wide spread the problem is, we investigated the age of the devices and found that the oldest phone (W800i) is from 2005 while the newest phone (Aino) is from late 2009.

**LG:** Our `LG GM360` seems to do insufficient bounds checking when parsing an MMS indication message. This allows us to construct an MMS indication SMS message containing long strings that span over three or more sms. This crashes the phone and thus forces an unexpected reboot when receiving the message or as well when trying to open the SMS message on the phone.

**Motorola:** As aforementioned, SMS supports *telematic interworking* with other network types. By sending one SMS message that specifies an Internet electronic mail interworking combined with certain characters in the payload it is possible to knock the phone off the mobile network. Upon receiving the message the phone shows a flashing white screen similar to the one shown by the Nokia phones. The phone does not completely reboot; instead it simply restarts the user interface and reconnects to the network. This process takes a few seconds and depending on the payload it is possible to achieve this twice in a row with one message. We verified this on the `Razr, Rokr,` and the `SVLR L7` – older, but extremely popular devices. The devices span 3+ years, providing us with confidence that the bug is present in their entire platform.

**Samsung:** Multipart UDH chunks are commonly used for payloads that span over multiple SMS messages. The header chunk for multipart messages is simple.

Our Samsung phones `S5230` and `B5310` do not properly validate such multipart sequences. This allows us to craft messages that show up as a very large SMS message on the phone. When opening such a message the phone tries to reassemble the message and crashes. Depending on the exact model one to four SMS messages are needed to trigger the bug.

**Micromax:** The `Micromax X114` is prone to a similar issue like the Samsung phones but behaves slightly differently. When sending one SMS that contains a multipart UDH with a higher chunk ID than the overall number of chunks and a reference ID that has not been used yet, the phone receives the SMS message without instantly crashing. However a few seconds after the receipt the display turns black for some seconds before the phone disconnects and reconnects to the network.

## 4.9 Validation and Extended Testing

After the initial fuzz-testing we needed to validate our results over a real operator network since we tested in a closed environment – our own GSM network. We need to evaluate if the bugs can be triggered in the real world or if operator restrictions prevent this. For the validation we put an active SIM card (of the four German operators) into our test phones and connected them to a real mobile phone network. We sent the SMS PDUs that triggered the bugs using the AT command interface of another mobile phone. These tests validated all the bugs described in the previous section.

During our fuzzing tests we deactivated the security PIN on the SIM cards we used in the target phones so that we did not have to enter the PIN on every reboot. We also tested the phones with an enabled SIM PIN. Our goal was to determine if such reboots also reset the baseband and the SIM card. If the SIM card is blocked after reboot the phone is not reconnected to the GSM network, and, thus, the user is cut off permanently. We determined that this is true for our LG, Samsung, and Nokia devices.

## 4.10 Bug Characterization

We group the discovered bugs depending on the software layer they trigger.

The first group are bugs that *require user interaction* such as the bug we discovered in the Samsung mobile phones. In this case the user has to view the message in order to trigger the bug.

The second group are bugs that crash *without user interaction*. These bugs occur as soon as the phone has completed receiving the entire message and starts processing it. In this group we put the bugs we found on the Motorola, LG, and Micromax devices.

The third and last group are bugs that trigger at a lower layer of the software stack. With lower layer we mean during the process of receiving the SMS message from the network. A crash *during the transfer process* means that the process is not completed and the network believes the message is not successfully delivered to the phone. We categorize the bugs discovered in our Nokia S40 and the Sony Ericsson devices in this third group.

## 5 Implementing the Attack

The attacks presented in this work utilize SMS messages to trigger software bugs and crash mobile handsets, interrupting mobile communications. These bugs cover the mobile phone platforms of all major handset manufacturers and a wide variety of different models and firmware versions. The resulting bug arsenal can potentially be abused to carry out a large scale attack.

### 5.1 Building a Hit-List

To launch an attack phone numbers of mobile phones need to be acquired since simply sending SMS messages to every possible number is problematic. Furthermore, sending SMS messages to a large number of unconnected phone numbers *dark address space* could trigger some kind of fraud prevention system, such as observed on the Internet to detect worms [7]. In addition, for the described attack only phone numbers that are connected to a mobile phone are of interest. Depending on the kind of attack, a different set of phone numbers is required. In one case an attack might be targeted towards a specific mobile operator, therefore, only phone numbers that are connected to the specific operator are of interest.

**Regulatory Databases:** In many countries around the world mobile network operators have their own area codes. Some examples are Germany[1], Italy[2], the United Kingdom[3], and Australia[4]. Such area codes can be readily acquired to help building a hit-list. Likewise one can use the *North American Numbering Plan* (NANP) to determine which area exchange codes are used by mobile operators.

**Web Scraping:** Web Scraping is a technique to collect data from the World Wide Web through automated querying of search engines using scripted tools. Finding German mobile phone numbers can be easily done through queries like `"+49151*" site:.de`. Moreover, online phonebooks [2] also include mobile phone numbers. These sites often allow wild card searches, and, thus can be abused to harvest mobile phone numbers.

**HLR Queries:** Some Bulk SMS operators [5] offer a service to query the Home Location Register (HLR) for a mobile phone number. These queries are very cheap (we found one for only 0.006 Euro) and answers the question if a mobile phone number exists and where it is connected. Together with the information from the regulatory databases one can easily generate a list of a few thousand mobile phone numbers that belong to a specific mobile network operator.

### 5.2 Sending SMS Messages

SMS messages can be sent by a mobile phone that provides either an API that allows it to send arbitrary binary messages or through its AT command interface. We used the AT interface for most of our testing and validation. To carry out any kind of large scale attack a way for delivering large quantities of SMS messages for low price is needed. Multiple options exist to achieve this:

**Bulk SMS Operators:** Bulk SMS operators such as [1, 5, 3] offer mass SMS sending over the Internet providing various methods ranging from HTTP to FTP and the specialized SMPP (Short Messaging Peer Protocol). Bulk SMS operators are so-called External Short Message Entity (EMSE) that are often connected via Internet to the mobile operators but sometimes have their own SS7 connection to the Public Switched Telephone Network (PSTN). Figure 4 shows the various connections of an EMSE. All Bulk SMS operators operate in the same way. For a given amount of money they deliver SMS messages to the specified destination(s). No questions asked. Most of the APIs support sending a single message to a list of recipients. Prices range from 0.1 to 0.01 Euro depending on the volume and destination of the messages. The APIs among the bulk SMS operators differ. Usually they allow to set a number of SMS fields from which they assemble the actual payload. Not all of them are offering the same predefined fields. For example [3] was the only one that allows us to set a TP-Protocol-Identifier field. However, we verified that the provided APIs are sufficient to carry out the presented attacks and to generate attack payloads that are identical to those sent from one of our phones.

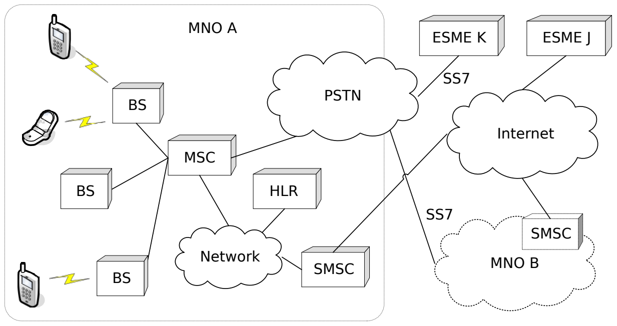

Figure 4: SMS relevant structure of a mobile network operator (MNO) network and the links to the PSTN, ESMEs, and other MNOs.

**Mobile Phone Botnets:** A botnet consisting of hijacked mobile or smartphones [35] could also be used for such attacks since every mobile phone is capable of sending SMS messages. A mobile botnet has the distinct advantage of free message delivery and high anonymity for the attacker. using a mobile phone botnet one could circumvent restrictions Bulk SMS operator might have in different countries.

**SS7 Access:** With direct access to the Signaling System 7 (SS7) of the Public Switched Telephone Network (PSTN) an attacker can very easily send SMS messages in large quantities, for example to send SMS spam [25]. Figure 4 shows the basic network connections of a mobile network operator. SMS sending via SS7 also has the advantage of not being easily traceable, thus an attacker can stay hidden for a longer period of time. Additionally, SMS messages sent via SS7 are not restricted by the Bulk SMS Operators (APIs) in terms of content or header information that they contain.

## 5.3 Reducing the Number of Messages

There is one issue left with our attack. That is how can one determine the type of mobile phone that is connected to a specific phone number. If money does not play a role in carrying out the attack this issue is easily resolved. The attacker just sends multiple SMS messages, each one containing the payload for a specific type of phone, to each phone number. One of the messages will trigger the bug if the phone is vulnerable at all. This works well but is not optimal. To reduce the number of messages an attacker has to send we developed a technique that allows the attacker to determine what kind of phone is connected to a specific phone number. Actually we can only determine if a specific malicious message has an effect on the phone that is connected to a specific number.

Our method abuses a specific feature present in the SMS standard. This feature is called recipient notification, it is indicated through the TP-Status-Report-Request flag in an SMS message. If the flag is set the SMSC notifies the sender of the message when the recipient has received the message. Most Bulk SMS operators support this feature through their APIs. Our method works by measuring the delay between sending the message and receiving the reception notification.

The technique works as follows: First, we send the message containing the payload for *crash(1)*. Second, when we receive the receipt for that message we send the payload for *crash(2)*. Third, we measure the time difference between the two notifications. If the difference is equal we continue with the next payload. If the difference between both notifications is significant we determine that the first message crashed the phone. The phone needed to reboot and register on the network before being able to accept the next message. If there is no notification we determine that the phone did not receive the message because it crashed before completely accepting the message. Fourth, we continue until all crash payloads are sent. If none of them trigger, the phone number is removed from the hit-list. The method can be optimized through ordering the crash payloads according to the popularity of mobile phones in the targeted country.

With this method an attacker can optimize a hit-list during an ongoing attack by matching bug-to-phone-number. This optimized hit-list could as well be used for highly targeted attacks. For example against the network operator as described in Section 5.5, which explains our attack scenarios.

## 5.4 Network Assisted Attack Amplification

Some of the bugs we discovered prevent the phone from acknowledging the SMS message to the network. Figure 2 shows the states that happen during a message transfer from the network to the phone. In the case of some of our bugs (Nokia S40 and Sony Ericsson; Bug Characterization Section 4.10) the message RP-ACK is not sent by the phone. This leads the network to believe that the message was not received, therefore, the SMSC will try to resend the SMS message to the phone. This redelivery attempt is a perfect attack amplifier somewhat similar to smurf attacks [26] on IP networks.

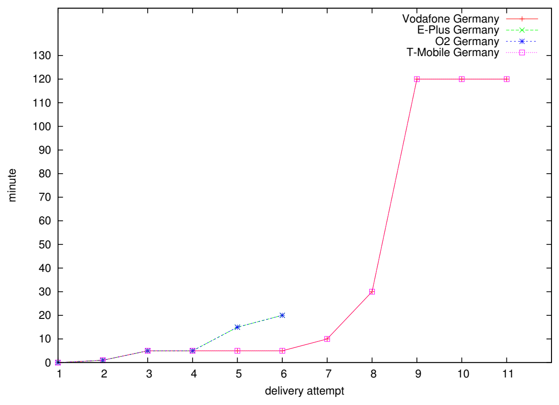

Figure 5: Timing of SMS message delivery attempts.

In our tests, sending malicious SMS messages over real operator networks, we discovered that operators have different re-transmit timings, shown in Figure 5. Furthermore, they also seem to have different transmit queues. We measured the delivery timings of some German mobile network operators in order to determine how one could abuse the delivery attempts for improving our Denial-of-Service attacks. We conducted the test by attacking one of our Sony Ericsson devices and monitoring the phone using the Bluetooth method described in Section 4.4.

The tests were carried out on the networks of Vodafone, T-Mobile, O2 (Telefonica), and E-Plus. The initial delivery attempt is at minute 0. It shows that all operators do a first re-transmit after 1 minute, and a few more re-transmits every 5 minutes. In addition to what Figure 5 shows, Vodafone does an additional re-delivery 24 hours after the last delivery shown in the graph. O2 also attempts an additional re-delivery 20 hours after the last delivery shown in the graph.

Through the same test we determined that SMS messages are not queued, but have an individual re-transmit timer. That means an attacker can send multiple malicious SMS messages to a victim's phone with a short delay between each message and thus can increase the effect of the network assisted attack by sending multiple messages.

## 5.5 Attack Scenarios and Impact

There are multiple possible attack scenarios such as organized crime going after the end-user, the mobile operator, and the manufacturer to demand money. Attacks could also be carried out for fun by script kiddies and the like. Below we discuss some possible scenarios. We acknowledge that some scenarios such as the attack against individuals are more likely then an attack against a manufacturer.

**Individuals:** Individuals could be pressured to pay a few Euros in order to keep their phone operational. This has happened with the Ikke.A [35] worm that requested the user to pay 5 Euros in order to get back the control over their iPhone. In our case the victim could be forced to send a text message to a premium rate number in order to be taken off the hit-list.

Another attack against an individual or a group could aim to prevent them from communicating. This can be efficiently carried out if the target uses a SIM card with security PIN enabled, as we describe in Section 4.9.

**Operators:** Operators could be threatened to have all their customers attacked. Such an attack would mainly kill the operator's reputation as being reliable. The operator might also lose money due to people being unable to call and send text messages. In order to have a global impact such an attack has to be carried out on a very large scale for a longer time. As a result, customers could possibly terminate their contract with the operator. Such extortion scams were and still are popular on the Internet [8].

Furthermore, the operator's mobile network can be attacked directly or as a side effect of an large attack against its users. This could work when thousands of attacked phones drop off the network and try to re-connect at the same time. This can cause an overload of the backend infrastructure such as the HLR. This kind of attack seems likely since mobile networks are not optimized for these specific kinds of requests. A similar attack based on unusual requests was shown in [36]. It is not normal that thousands of phones try to connect and authenticate at the same time over and over again. To optimize this DoS attack, the attacker needs to make sure to target phones connected to different BTSs and MSCs (Figure 4) of the targeted operator in order to circumvent bottlenecks such as the air interface at the BTS. A clogged air interface would throttle the attack.

**Manufacturers:** Likewise manufacturers could be threatened to have their brand name destroyed or weakened by attacking random people owning their specific brand of mobile phones. The attack could cost them twice. Once for the bad reputation and second for replacement devices. Even if the phones are not broken victims of such an attack will still try to claim their device broken to get a replacement.

**Public Distress:** A carefully placed attack during a time of public distress could lead to large scale problems and possibly a panic. One example occurred in Estonia [19] in 2007 when a group of people carried out a Denial-of-Service attack against the countries Internet infrastructure. Additionally, cutting off certain user groups such as fireman or police officers during an emergency situation would have a critical impact. Not every country has special infrastructure for emergency personal, and, therefore, rely on mobile phones to communicate. This is even true in countries like Germany where every police officer carries a mobile phone since their two-way-radios are often not usable.

## 6 Countermeasures

In this section we present countermeasures to detect and prevent the kind of attacks we developed. First, we present a mechanism to detect our and similar attacks through monitoring for a specific misbehavior. Second, we discuss filtering of SMS messages. Filtering can be done on either the phones themselves or on the network. We discuss the advantages and disadvantages of each of them. Third, we briefly discuss amplification attacks.

## 6.1 Detection

To prevent our attacks, operators first need to be able to detect them. Detection is not very easy since the operator does not get to look inside the phone during runtime. Therefore, the only possible way to monitor the phone is through the network. We propose the following:

**Monitor Phone Connectivity Status:** Monitor if a phone disconnects from the network right after receiving an SMS message.

**Log last N SMS Messages:** Log the last *N* SMS messages sent to a particular phone in order to analyze possible malicious messages after a crash was detected. Use the message as input for SMS filters/firewall.

**Use IMEI to Detect Phone Type:** The brand and type of a mobile phone can be derived from the IMEI (International Manufacturer Equipment Identity). This is useful to correlated malicious SMS messages to a specific brand and type of phone.

Using this technique it is possible to catch malicious SMS messages that cause phones to reboot and lose network connectivity. *This should especially help to catch unknown payloads that cause crashes.* Such a monitor is also capable of detecting if a large attack is in progress by correlating multiple SMS-receive-disconnect events in a certain time-frame.

## 6.2 SMS Filtering

SMS filtering can be implemented either directly on the phone or within the operator's network. Both possibilities have inherent benefits and drawbacks that are presented in this section.

It is important to reconsider the process of SMS delivery. First, an SMS message is sent from the sender phone to the senders SMSC. Next, the senders SMSC queries for the SMSC of the recipient and delivers the message to the responsible SMSC. Finally, the relevant SMSC locates the recipient's phone and delivers the SMS message via the BTS Over-the-Air.

**Client-side SMS Filtering** would need to be done right after the modem of the phone received and demodulated all the frames carrying the SMS message and before pushing it up the application stack. The filter would need to parse the SMS message and check for known bad messages similar to signature-based antivirus software or a packet filter firewalls. The problem with this solution is the update of the signatures. Of course, the parser in the SMS filter must be bug free otherwise the attack just moves from the phone software to the filter software. Also, devices that are already in the field would not profit from such a filter since only new phones will have this. Also, newer phones will likely not contain bugs that are known at the time they are manufactured. Therefore, we believe network-side filters make more sense.

**Network-side SMS Filtering** takes place on the SMSC of the mobile network operator. Therefore, it can inspect all incoming and outgoing SMS messages. There are multiple advantages of network-side filtering. First, the filter software runs on the network, therefore, it covers all mobile phones connected to that network. Second, changing the filter rules can be done in one central place. Third, malicious SMS messages are not sent out to the destination mobile phones, therefore, reducing network load during an attack.

Network-side filters also have drawbacks. First, if a phone is roaming within another operator's network, the SMS message does not travel through the network of the home operator. Thus the filters are not touched. This is the only advantage of phone-side SMS filtering. In this case the user becomes attackable as soon as he leaves his home network. For traveling business people in Europe, this is quite normal. The GSMA already has a solution for this issue called SMS homerouting. *SMS Homerouting* as specified in [11] defines that SMS messages are always routed through the receiver's home-network. Meaning that all SMS messages travel through SMSCs of his service provider at home. SMS messages, therefore, can be filtered by the receiver's service provider. The second issue with network-side filtering is privacy. In order to do SMS filtering the operator must be allowed to inspect SMS messages. This could be an issue in some countries where mobile telephony falls under special regulations.

## 6.3 Preventing Network Amplification

Attack amplification through re-transmissions of SMS messages should be avoided since this greatly helps an attacker. We suggest that operators limit the number of re-transmissions. Some operators re-send the messages 10 times, this seems unnecessary.

## 7 Conclusions

In this paper we have shown how to conduct vulnerability analysis of feature phones. Feature phones are not open in any way, the hardware and software are both closed and thus do not support any classical debugging methods. Throughout our work we have created analysis tools based on a small GSM base station. We use the base station to send SMS payloads to our test phones and to monitor their behavior. Through this testing we were able to identify vulnerabilities in mobile phones built by six major manufacturers. The discovered vulnerabilities can be abused for Denial-of-Service attacks. Our attacks are significant because of the popularity of the affected models – an attacker could potentially interrupt mobile communication on a large scale. Our further analysis of the mobile phone network infrastructure revealed that networks configured in a certain way can be used to amplify our attack. In addition, our attack can be used to not only attack the mobile handsets, but through their misbehavior can be used to carry out an attack against the core of the mobile phone network.

To detect and prevent these kind of attacks we suggest a set of countermeasures. We conceived a method to detect our and similar attacks by monitoring for a specific behavior.

## Acknowledgements

The authors would like to thank Charlie Miller, Andreas Krennmair, Dmitry Nedospasov, Borgaonkar Ravishankar, and Simon Schoar for their help reviewing the paper and for helping us to acquire phones for testing.

## References

[1] Clickatell Bulk SMS Gateway. `http://www.clickatell.com`.

[2] Das Örtliche. `http://dasoertliche.de`.

[3] Hay Systems Ltd. `http://www.hslsms.com`.

[4] Micromax mobile. `http://www.micromaxinfo.com`.

[5] Routo Messaging. `http://www.routomessaging.com`.

[6] Ping of Death. `http://insecure.org/sploits/ping-o-death.html`, October 1996.

[7] Honeynet Project. `http://project.honeynet.org`, 2005.

[8] DDoS extortion-themed scam circulating. `http://www.zdnet.com/blog/security/ddos-extortion-themed-scam-circulating/7180`, August 2010.

[9] Micromax becomes the third largest handset manufacturer in India. `http://www.topnews.in/micromax-becomes-third-largest-handset-manufacturer-india-2260105`, April 2010.

[10] When It Comes to Apps, Feature Phones Are the New Black. `http://gigaom.com/2010/03/27/when-it-comes-to-apps-feature-phones-are-the-new-black/`, May 2010.

[11] 3GPP/ETSI. TR 23.840 Study into routeing of MT-SMs via the HPLMN.

[12] 3GPP/ETSI. 3GPP TS 03.38 Alphabets and language-specific information. `http://www.3gpp.org/ftp/Specs/html-info/0338.htm`, 1998.

[13] 3GPP/ETSI. 3GPP TS 03.40 Technical realization of the Short Message Service. `http://www.3gpp.org/ftp/specs/html-info/0340.htm`, 1998.

[14] 3GPP/ETSI. 3GPP TS 04.11 Point-to-Point (PP) Short Message Service (SMS) Support on Mobile Radio Interface. `http://www.3gpp.org/ftp/specs/html-info/0411.htm`, 1998.

[15] 3GPP/ETSI. 3GPP TS 23.040 - Technical realization of the Short Message Service (SMS). `http://www.3gpp.org/ftp/Specs/html-info/23040.htm`, September 2004.

[16] ABI RESEARCH. Worldwide Mobile Subscriptions Number More than Five Billion. `http://www.abiresearch.com/press/3531-Worldwide+Mobile+Subscriptions+Number+More+than+Five+Billion`, October 2010.

[17] B. JURRY XFOCUS TEAM. Siemens Mobile SMS Exceptional Character Vulnerability. `http://www.xfocus.org/advisories/200201/2.html`, January 2002.

[18] B. MÜLLER. From 0 to 0-Day On Symbian. `https://www.sec-consult.com/files/SEC_Consult_Vulnerability_Lab_Pwning_Symbian_V1.03_PUBLIC.pdf`, 2009.

[19] BBC NEWS. Estonia hit by 'Moscow cyber war'. `http://news.bbc.co.uk/2/hi/europe/6665145.stm`, 2007.

[20] C. GUO, H. J. WANG, W. ZHU. Smartphone attacks and defenses. In *Third ACM Workshop on Hot Topics on Networks* (2004).

[21] C. MILLER. Exploiting the iPhone. `http://securityevaluators.com/content/case-studies/iphone/`, August 2007.

[22] C. MILLER, M. DANIEL, J. HONOROFF. Exploiting Android. `http://securityevaluators.com/content/case-studies/android/index.jsp`, October 2008.

[23] C. MULLINER, C. 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).

[24] C. MULLINER, G. VIGNA. Vulnerability Analysis of MMS User Agents. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)* (Miami, FL, December 2006).

[25] CELLULAR-NEWS. A "rising Tide" of SS7 Based Mobile Network Fraud. `http://www.cellu`

lar-news.com/story/46377.php, November 2010.

[26] CERT. Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks. http://www.cert.org/advisories/CA-1998-01.html, January 1998.

[27] COMSCORE. German Mobile Market Share. http://www.comscore.com/index.php/Press_Events/Press_Releases/2010/1/comScore_Reports_November_2009_German_Mobile_Market_Share, November 2010.

[28] COMSCORE. U.S. Mobile Subscriber Market Share. http://comscore.com/Press_Events/Press_Releases/2010/7/comScore_Reports_May_2010_U.S._Mobile_Subscriber_Market_Share, May 2010.

[29] D. MOORE, V. PAXSON, S. SAVAGE, C. SHANNON, S. STANIFORD, N. WEAVER. Inside the Slammer Worm. *IEEE Security and Privacy 1* (2003), 33–39.

[30] H. WELTE. OpenBSC. http://openbsc.osmocom.org/trac/, 2008.

[31] IDC. Western European Mobile Phone Market Grows. http://www.idc.com/getdoc.jsp?containerId=prUK22402810, June 2010.

[32] IP.ACCESS LTD. nanoBTS 1800. http://www.ipaccess.com/picocells/nanoBTS_picocells.php.

[33] MOBILE SECURITY LAB. SonyEricsson WAP Push Denial of Service. http://www.mseclab.com/?page_id=123, January 2009.

[34] O. WHITEHOUSE. Nokia Phones Vulnerable to DoS Attacks. http://www.infoworld.com/article/03/02/26/HNnokiados_1.html, February 2003.

[35] P. A. PORRAS, H. SAIDI, V. YEGNESWARAN. An Analysis of the iKee.B iPhone Botnet. In *Proceedings of the 2nd International ICST Conference on Security and Privacy on Mobile Information and Communications Systems (Mobisec)* (May 2010).

[36] P. TRAYNOR, M. LIN, M. ONGTANG, V. RAO, T. JAEGER, T. LA PORTA, P. MCDANIEL. On Cellular Botnets: Measuring the Impact of Malicious Devices on a Cellular Network Core. In *ACM Conference on Computer and Communications Security (CCS)* (November 2009).

[37] P. TRAYNOR, W. ENCK, P. MCDANIEL, T. LA PORTA. Mitigating attacks on open functionality in sms-capable cellular networks. In *In ACM MobiCom* (2006), pp. 182–193.

[38] R. FARROW. DNS Root Servers: Protecting the Internet.

[39] R. RACIC, D. MA, H. CHEN. Exploiting MMS vulnerabilities to stealthily exhaust mobile phone's battery. In *Proceedings of the Second IEEE Communications Society / CreateNet International Conference on Security and Privacy in Communication Network (SecureComm)* (Baltimore, MD, Auguest 28 - September 1, 2006).

[40] S. BYERS, A. D. RUBIN, D. KORMANN. Defending against an Internet-based attack on the physical world. *ACM Trans. Internet Technol. 4*, 3 (2004), 239–254.

[41] SUN MICROSYSTEMS. Java Micro Edition. http://www.oracle.com/technetwork/java/javame/index.html.

[42] T. AHONEN. Mobile Phone Market Shares for year of 2009. http://communities-dominate.blogs.com/brands/2010/02/phone-market-shares-for-year-of-2009-and-last-quarter-2009.html, February 2010.

[43] T. AHONEN. *Tomi Ahonen Almanac 2010 Mobile Telecoms Industry Review*. February 2010.

[44] T. ENGEL. Remote SMS/MMS Denial of Service - Curse Of Silence. http://berlin.ccc.de/~tobias/cursesms.txt, December 2008.

[45] THE INTREPIDUS GROUP. WebOS: Examples of SMS delivered injection flaws. http://intrepidusgroup.com/insight/2010/04/webos-examples-of-sms-delivered-injection-flaws/, April 2010.

[46] V. IOZZO, P. WEINMANN. iPhone Safari vulnerability allowed to steal the SMS database. http://news.cnet.com/8301-27080_3-20001126-245.html, March 2010.

[47] W. ENCK, P. TRAYNOR, P. MCDANIEL, T. LA PORTA. Exploiting Open Functionality in SMS-Capable Cellular Networks. In *Conference on Computer and Communications Security* (2005).

[48] WAP FORUM. WAP-209-WSP Wireless Application Protocol MMS Encapsulation Protocol. http://www.wapforum.com, 2002.

## Notes

[1]http://en.wikipedia.org/wiki/Telephone_numbers_in_Germany

[2]http://en.wikipedia.org/wiki/Telephone_numbers_in_Italy

[3]http://en.wikipedia.org/wiki/Telephone_numbers_in_the_United_Kingdom

[4]http://en.wikipedia.org/wiki/Telephone_numbers_in_Australia

## APPENDIX

Figure 6 shows the layout of an SMS message in the SMS_SUBMIT format. Figure 7 shows the generic layout of a User Data Header (UDH) with a number of Information Elements.

| Field | Size |
|---|---|
| TP-Message-Type-Indicator | 2 bit |
| TP-Reject-Duplicates | 1 bit |
| TP-Validity-Period-Format | 2 bit |
| TP-Status-Report-Request | 1 bit |
| TP-User-Data-Header-Indicator | 1 bit |
| TP-Reply-Path | 1 bit |
| TP-Message-Reference | integer |
| TP-Destination-Address | 2-12 byte |
| TP-Protocol-Identifier | 1 byte |
| TP-Data-Coding-Scheme | 1 byte |
| TP-Validity-Period | 1 byte/7 byte |
| TP-User-Data-Length | integer |
| TP-User-Data | depends on DCS/UDL |

Figure 6: Format of the SMS_SUBMIT PDU.

| Field | Size |
|---|---|
| UDHL | 1 byte |
| $IEI_1$ | 1 byte |
| $IEIDL_1$ | 1 byte |
| $IEID_1$ | n bytes |
| ... | |
| $IEI_n$ | 1 byte |
| $IEIDL_n$ | 1 byte |
| $IEID_n$ | n bytes |

Figure 7: The User Data Header

Table 8 shows an overview of the popularity of mobile phone manufacturers in Germany, the United States, in Europe, and around the world.

| Manufacturer | Market Share |
|---|---|
| Nokia | 35.4% |
| Sony Ericsson | 22.0% |
| Samsung | 15.0% |
| Motorola | 8.6% |
| Siemens | 5.4% |

(a) Germany, November 2009

| Manufacturer | Market Share |
|---|---|
| Samsung | 22.4% |
| LG | 21.5% |
| Motorola | 21.2% |
| RIM | 8.7% |
| Nokia | 8.1% |

(b) U.S.A., May 2010

| Manufacturer | Market Share |
|---|---|
| Nokia | 32.8% |
| Samsung | 12.5% |
| LG | 4.1% |
| Sony Ericsson | 3.7% |
| Apple | 3.0% |
| RIM | 2.4% |
| Others | 3.0% |

(c) Europe, June 2010

| Manufacturer | Market Share |
|---|---|
| Nokia | 38.0% |
| Samsung | 20.0% |
| LG | 10.0% |
| Sony Ericsson | 5.0% |
| Motorola | 5.0% |
| ZTE | 4.5% |
| Kyocera | 4.0% |
| RIM | 3.5% |
| Sharp | 2.6% |
| Apple | 2.2% |
| Others | 5.0% |

(d) World, for the year 2009

Figure 8: Mobile phone Manufacturer Market share