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
D-10587, Berlin, Germany
{collin,nico,jpseifert}@sec.t-labs.tu-berlin.de

***Abstract*—tbw**

Keywords: Mobile Phones, SMS, Vulnerability Analysis, Denial-of-Service, Large Scale Attack.

I. Introduction

In the recent years a lot of effort has been put in to analyzing and attacking smart phones [1], [2], [3], [4], [5], [6], [7], [8], totally neglecting the so-called feature phones. Feature phones, mobile phones that have advanced capabilities besides voice calling and text messaging but are not considered smart phones, make up for the largest percentage of mobile devices currently deployed on mobile networks around the world. In comparison, smart phones only account for about 16% of all mobile phones [9]. This gap of security analysis to popularity most likely roots from the fact that smart phones are closer to desktop computers, and, therefore, are easier to analyze. Feature phones, on the other hand, are highly embedded systems that are closed for developers. This results in having billions (there are 4.6 billion mobile phone subscribers) of possible vulnerable mobile devices out in the field, just waiting to be taken advantage of by a knowledgeable attacker who launches large scale attacks against mobile users and operators around the world.

In this paper, we investigate the security of feature phones and the possibility for large scale attacks based on vulnerabilities discovered in those.

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. Further, security issues related to SMS messaging can be exploited from almost anywhere in the world, and, thus present the ideal attack vector against those 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, without any possibilities for developing native applications or debugging. Further, analyzing the part of the phone that interacts with the mobile phone network is hard since we have a large black box between us and the target device. The mobile phone network. As a consequence, the testing becomes time consuming, unreliable, and costly.

We address these problems by building our own GSM network using equipment that can be bought on the public market. We use this network not only for sending SMS messages to the phones we analyze, but further use it to develop a sophisticated monitoring system. The monitoring system replaces our need for debuggers and other tools that are normally required for thorough vulnerability analysis.

Vulnerability analysis was conducted using fuzzing. We chose fuzzing as the testing technique because we did not have access to source code. Further, using fuzzing we can design tests once and use them to analyze an arbitrary number of mobile phones and thus be very efficient.

So far, we have found numerous vulnerabilities in feature phones soled by the six market leading mobile phones manufacturers. The vulnerabilities are security critical since the allow to remotely crash and reboot the entire phone. In the event disconnecting the phone from the mobile network and thus interrupting calls and data connections. Such bugs and attacks have existed before on the Internet, known as Ping-of-Death [10]. We believe this represents a serious threat against mobile telephony world wide.

To complete our research we further analyzed the effect of such attacks no the mobile phone core network. We came to two interesting findings. First, the mobile phone network can be abused to amplify our Denial-of-Service attacks. Second, through attacking mobile phones one can attack the mobile phone network itself.

The main contributions of this paper are:

- **Vulnerability Analysis of Feature Phones:** We introduce a novel method to conduct vulnerability analysis of feature phones based on fuzzing. Our method is based on a small GSM basestation that is available on the public market. We solve the major issue of such analysis: the monitoring for crashes and other misbehavior. We present multiple solutions for monitoring such devices while fuzzing them. Further, our method proves 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 used 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 all mobile phones around the world. We further show that bugs present in mobile phones can possible be use to attack the mobile phone network infrastructure.

The rest of this paper is structured in the following way. In Section II we discuss related work and show how our research is different from previous work in this area. In Section III explain how we selected our targets for analysis and attacks. In Section IV we show in great detail how to analyze feature phones for security vulnerabilities. In Section V we layout methods to use the vulnerabilities discovered by us for large scale attacks on mobile communication. In Section VI we present methods for detecting and preventing the attacks we designed. In Section VII we briefly conclude.

II. Related Work

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

In [3] the authors build a framework for security analysis of Multimedia Messaging Service (MMS) client implementations on smart phones. The authors conducted fuzzed-based testing of a Windows Mobile device. They found and exploited several vulnerabilities of which some lead to arbitrary code execution. In [11] the authors build a framework for conducting vulnerability analysis for Short Message Service (SMS) client implementations on smart phones. Particular the iPhone, Android, and Windows Mobile. They found several vulnerabilities. These could be used to disconnect these types of smart phones from the mobile phone network and could execute arbitrary code on the iPhone. Because of the popularity of especially the iPhone and Android-based devices many researchers poke these operating systems and publish vulnerabilities for them. Other bugs such as the Curse-of-Silence [12] named bug for Symbian OS (it prevents a phone from further receiving any SMS after being attacked) where found by accident.

In the area of mobile phones bugs and vulnerabilities all finds occurred by accident since no real academic testing has been conducted before. We believe that some internal testing is being conducted by some manufactures since there exits commercial software for such purpose [13]. Over the last few years a small number of bugs haven been discovered by individuals. The WAP-Push vCard against Sony Ericsson phones [14] that lead an attacked phone to reboot. Nokia phones [15] contained a bug in the vCard parser that could be abused to remotely crash a phone by sending it special manipulated vCard through SMS. Some mobile phones produced by Siemens contained a bug [16] that would shutdown the phone when displaying a SMS message that contained a special character.

Traynor at el. show in [17] that SMS messages sent from the Internet can be used to carry out a Denial-of-Service attack against mobile networks. The attack focused on blocking the mobile network's control channels, therefore, no more calls could be initiated. A study on the capabilities of a mobile phone botnet [18] shows that it could be used to carry out an Denial-of-Service attack against an operator network. Their attack works by overloading the Home Location Register (HLR) through massively triggering state changes through the zombie phones.

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 [19]. More interesting in our context are attacks that disable real-world systems and processes such as emergency services [20] or even postal services [21].

The work presented in this paper is different in many aspects. First, we focus on feature phones rather than smart phones. We do this because feature phones are much more popular than smart phones, therefore, attacks against feature phones have a larger impact. In our 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 III. We do this since this has not been done so far. Second, we show how one can use bugs and vulnerabilities in popular mobile phones to carry out large scale attacks against mobile phone users and mobile network operators world wide. The kind of attack shown by us can be used by the script kiddy next door and organized crime in order to make money.

III. Target Selection

To achieve maximum impact with an attack it makes sense to go after the most popular devices. This is what happens with trojans and botnets, the botmasters go after Microsoft Windows because it is the dominant operating system.

We determined that feature phones are the dominate type of mobile phones. They account for 83% of the U.S. mobile market [22], smart phones in comparison just make for 16% of all mobile phones world wide [9].

Most of the definitions of the term *feature phone* are a bit fuzzy. A lose definition of the term is: every mobile phone that is neither a dumb phone nor a smart phone is considering a feature phone. Dump 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 smart phones but still more than dumb phones. Feature phones have proprietary operating systems (firmware), have additional features (thus the term feature) such as playing music, surfing the web, and running simple applications (mostly J2ME [23]). Despite this lack of functionality (compared to smart phones) 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. Smart phones usually consist of two processors. 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 [24] and European [25] market share. Market shares in the United States [26] and in Germany [27]. The essence of these market reports are shown in Table I. Table I(a) shows Germany and Table I(b) shows the popularity for the United Stats, interesting is that those tables are almost the opposite of each other. Tables I(c) shows Europe and Table I(d) shows the world wide popularity.

Through this analysis we got a clear picture about the top manufacturers. These seem to be: `Nokia`, `Samsung`, `LG`, `Sony Ericsson`, and `Motorola`. We further chose to add `Micromax` [28] to the list of interesting mobile phone manufacturers because we read [29] that they are the third most popular brand of mobile phones in India. Thus, a nice example of a targeted attack against a country.

Table I
MOBILE PHONE MANUFACTURER MARKET SHARE

(a) Germany, November 2009

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

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

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

(c) Europe, June 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%

(d) World, for the year 2009

Manufacturer	Market Share
Nokia	38%
Samsung	20%
LG	10%
Sony Ericsson	5%
Motorola	5%
ZTE	4.5%
Kyocera	4%
RIM	3.5%
Sharp	2.6%
Apple	2.2%
Others	5%

IV. SECURITY ANALYSIS OF FEATURE PHONES

Analyzing feature phones for security vulnerabilities is hard for several reasons. No access to source code of the OS and applications. No existing native-SDKs, therefore, no chance to run native code on the device and further no access to a debugger.

Because of these reasons we choose to conduct fuzzed-based testing. The test would run 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. Than we will describe our novel monitoring setup. The final part of the Section will discuss test cases and the resulting bugs we discovered through our work.

A. 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 don't risk to interfere with the telco network. We want to avoid crashing telco network equipment by either content or quantity of SMS messages. Using our own network further guarantees us to create reproducible results since we control the entire system and are not left with guessing in the case something does not work as expected. In addition the delivery of SMS messages is much faster on our small network compared to a production setup of a

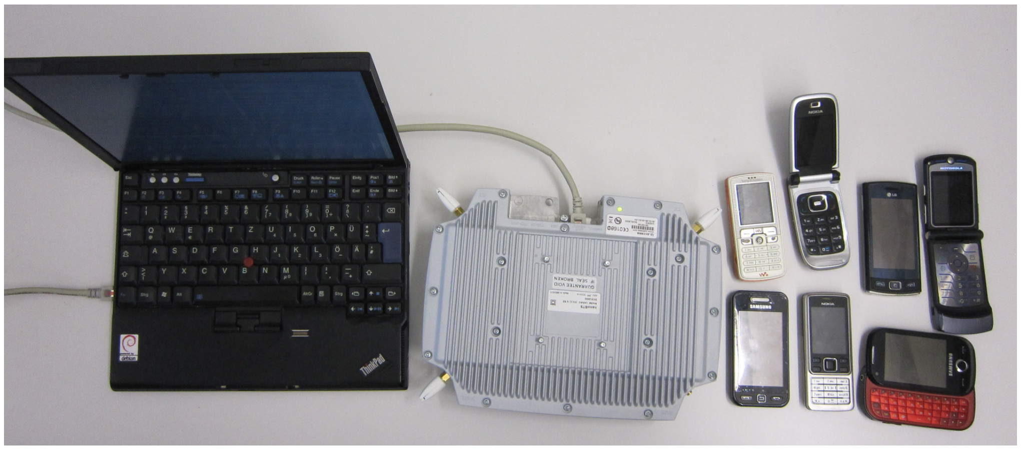

Figure 1. Our setup: The laptop that runs OpenBSC and the fuzzing tools, the nanoBTS, and some of the phones we analyzed.

mobile operator. Further, network operators can not spy on us while we conducting our testing.

On the hardware side we decided to use an ip.access nanoBTS [30] which is a small, fairly cheap (about 5000 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 [31]. OpenBSC is a Free Software BSC implementation of the A-bis protocol which 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.

As GSM operates on a licensed frequency spectrum we had to carry out our experiments in an Faraday cage. Our Faraday cage is a 3x3 meter room with a filtered power supply and fiber optic network connection. In side the room we have working space for three people and our GSM network equipment. A malicious party would of course not bother with this kind of setup.

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 store and forward fashion. As we later see the telnet 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 no 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

SMS [32] PDUs exist in two formats, SMS_SUBMIT and SMS_DELIVER. In [11] fuzzing was based on the SMS_DELIVER format and SMS were directly injected on the smart phones. We utilize our GSM network for fuzzing, and, therefore, our fuzzed SMS are based on SMS_SUBMIT. In a typical GSM network, shown in Figure 6, an SMS message that is sent from a mobile device is transferred Over-the-Air to the Base Transceiver Station (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 in 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 and its job is to forward and relay the message 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. We wont get into detail on the mechanism of finding and routing messages to responsible destinations as this is not important in the fuzzing setup since there is only one SMSC. During the final transmission to the destination the SMS will get converted to SMS_DELIVER, this is taken care of by OpenBSC.

B. Sending SMS Messages

OpenBSC itself does not provide an interface to submit pre-encoded SMS messages to the network but only a telnet interface to submit text SMS messages that are than converted into the corresponding encoding. We added a new telnet interface command to OpenBSC that allows us to submit SMS messages directly in SMS_SUBMIT format. This fills an internal SMS data structure with the data that is used to store the SMS message in an SQLite database that is used by OpenBSC as part of the SMSC functionality. Modifying the existing text message interface to be capable of handling binary encoded SMS messages would have been no option. 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. About seven seconds per message. This is also the reason why we didn't want to use a mobile phone to send our fuzzed-messages in the first place. Our way of injecting messages is much faster and works in the following way. First, we use our new telnet interface to inject messages in the SUBMIT format. This way we can add many hundreds or thousand messages to the SMSC database. Second, we send these messages. This ideally only opens the channel once, sends out all SMS message to the recipient and than close 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 by a direct interface to the network. This way we don't require any modifications on the phones. By default OpenBSC only stores the parsed values in the database and the decoded text. As in our case we don't only send simple text messages we also added storing the complete encoded SMS message into the database. This also allows us to easily replay SMS messages that may have caused problems by simply selecting the column from the database.

C. *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 which takes care of errors reported from the phone which we modified to fit our fuzzing case. The default error handler doesn't differ between errors and causes the SMS sending process to stop in case of an error. The only difference 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 space again.

The mobile phone as well as the MSC is usually divided into separated layers for transferring and processing a message. As visible 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 [33] receives and relays message that it receives from the application layer in TPDU form (Transport Protocol Data Unit), this is the original encoding form which will be described at a later point 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-SLs with each other. The CM-SL 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 done sanity checks on the content of the SMS and parsed it. The resulting reply, passed to CM-Sub, will include either an acknowledgement of the SMS message and it will than 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.

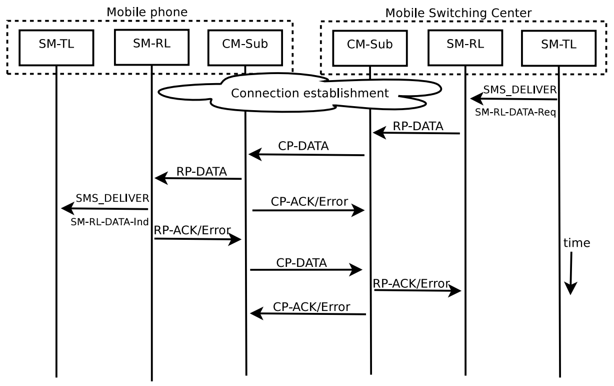

Figure 2. Mobile terminated SMS

From the wide variety of error messages a phone can reply to a received SMS messages, defined in [34], we observed during our fuzzing experiments that all of the tested phones either reply with a `Protocol Error` or `Insufficient Mandatory Information` message in the case of malformed messages. 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 hasn't 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 (Slow Dedicated Control Channel) or a SACCH (Slow Associated Control Channel), the details of such a channels 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

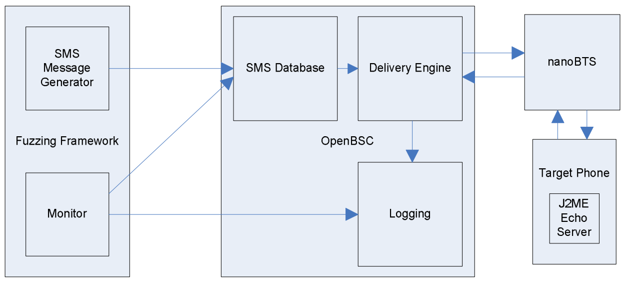

Figure 3. Logical view of our setup.

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 signalling between the BSC and the BTS happens over the A-bis interface over highly standardized protocols. We added modifications to the A-bis Radio Signalling Link code of OpenBSC that allows us to check if a channel tear down happens in an 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 limited towards the technical reason and impact of such an error condition in the meaning that we are not able to peak at register values and other software related details. 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've 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 under the assumption that he 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. Third, 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 than is rather easy. Our script checks the error condition and if it is 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 just checks which messages have been sent in a certain time interval around the occurrence of the log entry. During our testing we decided that a one minute time interval works well enough to have a small amount of prospective SMS messages that could've caused a problem. Figure 3 shows the logical view of our monitoring setup.

D. *Additional Monitoring Techniques*

Additionally to the above OpenBSC setup we have developed more methods for monitoring for abnormal behavior.

USB Cable: By hooking-up the phone via it's USB data cable we can monitor if the connection is broken (USB device disconnects) during testing. This would be a hint about abnormal behavior.

Bluetooth: Similar to connecting the USB cable 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). It connects 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 signalling us that something went wrong.

J2ME: Almost every modern feature phone supports J2ME [23] and this is the only way for us to do measurements on the phone since they don't 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 UDH [33] (User Data Header) 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 an valid message that is addressed to this port. The application than 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

Field	Size
TP-Message-Type-Indicator	2 byte
TP-Reject-Duplicates	1 byte
TP-Validity-Period-Format	1 byte
TP-Reply-Path	1 bit
TP-User-Data-Header-Indicator	1 bit
TP-Status-Report-Request	1 bit
TP-Message-Reference	1 byte
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	1 byte
TP-User-Data	depends on DCS/UDL

Figure 4. Format of the SMS_SUBMIT PDU.

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 happened here [12].

E. SMS Encoding

Before we go into details of our test cases we provide a brief introduction to the SMS_SUBMIT encoding as defined in [32] and shown in Figure 4.

TP-Message-Type-Indicator when set properly is used to indicate that the message is an SMS_SUBMIT message. Only the fields *TP-Protocol-Identifier, TP-Data-Coding-Scheme, TP-User-Data-Length, TP-User-Data* are interesting for fuzzing. The remaining fields just describe how the SMS message should be delivered or how SMSCs and mobile phones should generally handle them.

TP-Protocol-Identifier is used to describe the type of the messaging service being used. This refers to a higher layer protocol or inter-working being used.

TP-Data-Coding-Scheme as described in [35] is used to indicate a certain message class, which alphabet is used to encode *TP-User-Data*. Whether the payload is compressed and to indicate if new messages are waiting for the customer (used in certain countries to indicate waiting voice-mails for example). Message class 1 is usually the default for normal text messages and indicates that should be stored on the mobile phone while a class of 2 is used to signal that the messages should be stored on the SIM card. 2 bits in this field are used to encode the GSM alphabet [35] that was used to encode the *TP-User-Data* payload. This can be either the default 7 bit, 8 bit or 16 bit alphabet and a reserved value. Together with the *TP-User-Data* fields *TP-Protocol-Identifier* and *TP-Data-Coding-Scheme* are our main targets for fuzzing. The receiving mobile phone than reassembles the message based on this information.

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 byte

Figure 5. The User Data Header

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

The User-Data-Header (UDH) is used to provide additional control information just like headers in IP packets. This includes port addressing, headers for concatenated messages, text formatting and other information. UDHL specifies the length of the complete UDH and is followed by one or multiple headers. IEI are so called Information Elements [32]. A one byte identifier specifying the type of header, IEDL specifies the length of the header element and IEID is the actual header data. The UDH with a number if Information Elements is shown in Figure 5.

F. Fuzzing Test-cases

We've implemented 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 specific fields and not features. 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: 0x00, IEL: 0x03). 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 number. By fuzzing these three values we were mainly looking 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 the MMSC sends an MMS notification indication message [36] that contains the URL to the MMS content. The phone than starts a GPRS connection to download the content. This MMS indication is in fact a binary encoded WAP-push message sent via SMS. The notification can also contain additional 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. MMS indication messages 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: As already outlined the combination of both of these fields yields to 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 often unused by normal SMS traffic.

Flash SMS: Flash SMS 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 to this, 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.

G. 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 message and reproducing problems from log entries often could be done without any problems. However fuzzing phones is not great fun as we stumbled across various forms of strange behavior. Problems we faced included non-standard conform message replies and all sorts of weird behavior. Some phones weren't properly reporting memory exhaustion. Others didn't notice free memory until a reboot. Some didn't 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 one point because it became simply unusable for unknown reason or the mass of messages or a specific SMS needed to be deleted from the SIM card using another phone. The biggest problem with hard resets is that a lot of manufacturers don't seem to do what the name hard reset suggest, bring the phone into a fresh manufacturer 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 Corbi it was always enough to remove the offending SMS message from the phones SIM card while the Samsung S5230 needed an additional hard reset. Things like this have been very time consuming during the fuzzing but this is also noteworthy as it seems to be hard for phone customers to 100% get rid of personal information from the phone when i.e. selling the phone.

In summary we stumbled across a lot of annoyances which are not important from a security point of view but in our overall impression most of the feature phones we tested behaved rather fragile when handling SMS messages.

H. Results

During our fuzz-testing we discovered quite a number of bugs that lead to security vulnerabilities. The bugs mostly lead to phones crashing and rebooting and in the event disconnecting them from the mobile network, and thus interrupting established voice calls and data connections. During the testing we even managed to *bricked a couple of phones*. We didn't investigate the bricking in-depth because this would have gotten quite expensive. We have a good idea of what causes the bricking and we reported it to the manufacturer. Further 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 the bugs we discovered on each platform:

Nokia S40: On our Nokia test devices `6300, 6233, 6131 NFC` we found a bug in the flash SMS implementation. By sending a certain flash SMS the phone crashes and triggers a the "Nokia white-screen-of-death". This also results in the phone disconnecting and re-connecting again to the mobile phone network. The interesting thing about this issue is that the SMS actually never reaches the mobile phone. The phone will crash before it can fully process and 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. One 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: Sony Ericsson devices `W800i, W810i, W890i, Aino` have a problem similar to the Nokia S40 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. Just like in the Nokia case this SMS message will never be acknowledge by the phone thus the GSM network will continuously re-transmits the message to the victim when it re-associates with the network.

LG: When a subscriber receives an MMS (Multimedia Messaging Service) message the MMSC (Multimedia Messaging Service Center) sends an MMS indication message that contains the URL to the MMS content. The phone than starts a GPRS connection to download the content. This MMS indication is in fact a binary encoded WAP-push SMS message that can also contains additional variable length fields for subject, transaction ID and sender name. Our device the, `LG GM360`, seems to do insufficient bounds checking when parsing these values. This allows us to construct a multipart message with large field values that crash the phone and thus force an unexpected reboot when receiving the message or when trying to open the SMS message on the phone.

Motorola: Like mentioned before SMS supports telematic inter-working with other networks. By sending an SMS message that specifies an Internet electronic mail inter-working combined with certain characters in the payload it is possible to knock the phone off the mobile network. On receiving the message the phone shows a flashing white screen similar to the one shown by the Nokia phones. The phone is not completely rebooting but restarting the user interface and network connectivity. 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 `Razor, Rokr,` and the `SVLR L7` – older but extremely popular devices.

Samsung: Multipart UDH chunks are commonly used for payloads that span over multiple SMS messages. The header chunk for multipart messages is simple. It contains a reference number to identify the series of chunks and is used by the phone to reassemble the parts. It also consists of a number indicating the current part number (the parts are reassembled in order of their part numbers) and an additional field specifying the overall number of all parts belonging to a certain reference number.

Our Samsung phones `S5230 Star, Corbi, S3250` do not properly validating such multipart sequences. This enables one 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 between one and three 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 different. When sending a multipart SMS that contains a higher chunk ID than the overall number of chunks and a reference ID that hasn't 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. This process takes a couple of seconds after which the phone displays "Message not ready" for a short time. Besides this a victim won't notice the message.

I. Validation and Extended Testing

After the initial fuzz-testing we needed to validate our results. This is necessary since we tested in a closed environment – our own GSM network. We need to validate if the bugs and vulnerabilities found by us can be triggered in the real world. For the validation we put a active SIM card into our test phones and connected them to a real mobile phone network. We send the SMS PDUs that trigger the bugs using another mobile phone. Through this testing we validated all the bugs described in the last Section.

During our fuzzing tests we deactivated the security PIN on the SIM cards we used in the target phones so that we didn't have to enter the PIN on every reboot. We re-enabled the security PIN and fired the problematic SMS messages towards the target phones. We wanted to determine if the reboot of the phones is in a way that it also reboots 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.

J. Bug Characterization

We group the discovered bugs depending on the software layer they trigger. We later use this for designing our attacks.

The first group are bugs that *require user interaction* such as the bug we discovered in the Samsung mobile phones. Here 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 trigger the moment 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.

V. Implementing the Attack

The attacks we are presenting attempt to interrupt mobile phone-based communication through crashing mobile handsets using SMS messages that trigger software bugs. *The attacks can be carried out on a large scale because we discovered bugs in the mobile phone platforms of all major handset manufacturers.*

In this Section we layout how one can implement such a large scale attacks. This Section is separated in to the following parts. First, we discuss hit-list generation and sending of SMS messages. Second, we show how to further optimize attacks. Finally, we discuss actual attack scenarios and the impact of those scenarios.

A. Build a Hit-List

To launch an attack phone numbers of mobile phones need to be acquired since just sending SMS messages to every possible number will fail. Further, 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 [37]. Further 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.

Creating a hit-list for different countries is another problem as number assignment works different everywhere. In this Section we discuss multiple methods for number harvesting to create a hit-list for our attacks.

Regulatory Databases: In [17] Enck at el. show multiple methods on how to create a hit-list of mobile phone numbers in the United States. They use databases from the *North American Numbering Plan* (NANP). Here one can check which operator manages a certain exchange code. If such an exchange code is managed by a known wireless operator such at AT&T Wireless it is likely that all numbers within this exchange are connected to a mobile phone. Given that a specific number is connected at all.

In Germany mobile network operators have their own area codes. In Germany the list of mobile network area codes can be easily acquired from the agencies website [38]. The same is true in many countries around the world, some examples are: United Kingdom[1], in Australia[2], in Italy[3], and in Korea[4].

The regulatory database information can be used for many purposes. It can be used as input when running queries against an HLR and when pulling data from address databases.

Web Scraping: Web Scraping is a technique to collect data from the Word Wide Web through automated querying of search engines using scripted tools. In [17] the authors easily found a number of phone numbers in the United States using this technique. Finding German mobile phone numbers can be easily done through queries like `"+49151*" site:.de`. Further online phonebooks [39] today also include mobile phone numbers. These sites often allow wildcard searches, and, therefore, can be abused to harvest mobile phone numbers.

HLR Queries: Some Bulk SMS providers [40] 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.

B. Sending SMS Messages

SMS messages can be sent by a mobile phone that provides either an API that allows to send arbitrary binary messages or thru it's 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 [41], [40], [42] offer mass SMS sending from 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 6 shows the various connections of an EMSE. All Bulk SMS operators operate in the same way. You pay them money and they deliver your arbitrary SMS messages to the given destination(s). No questions asked. Most of the APIs support sending a single message and a list of recipients. Prices

[1]http://en.wikipedia.org/wiki/Telephone_numbers_in_the_United_Kingdom
[2]http://en.wikipedia.org/wiki/Telephone_numbers_in_Australia
[3]http://en.wikipedia.org/wiki/Telephone_numbers_in_Italy
[4]http://en.wikipedia.org/wiki/Telephone_numbers_in_Korea

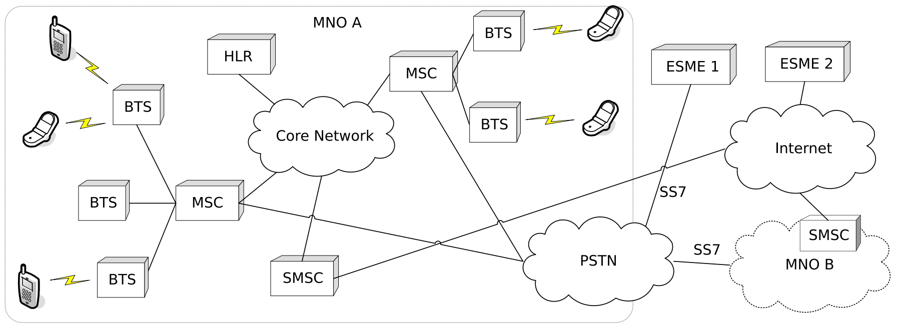

Figure 6. SMS relevant structure of a mobile network operator (MNO) network and the links to the PSTN, ESMEs, and other MNOs.

range from 0.1 to 0.01 Euro depending on the volume and destination of the messages.

Mobile Phone Botnets: A botnet consisting of hijacked mobile or smart phones [43] 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. Further 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 [44]. Figure 6 shows the basic network connections of a mobile network operator. SMS sending via SS7 further has the advantage of not being easily traceable, thus a attacker can stay hidden for a longer period of time. Further SS7 sent SMS messages are not restricted in terms of content or header information such as opposed by some of the Bulk SMS operators.

C. *Identifying Phones to Reduce 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 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 call 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 like this: 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 didn't receive the message because it crashed before completely accepting the message. Forth, we continue until all crash payloads are send. If non 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 build a hit-list with matching bug-to-phone-number. This optimized hit-list can be used for highly targeted attacks. For example against the network operator himself as we describe in Section V-E where we detail our attack scenarios.

D. *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 IV-J) the message RP-ACK is not send 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 re-delivery attempt is a perfect attack amplifier loosely similar to smurf attacks [45] on IP networks.

Through our testing trials of sending malicious SMS messages over real operator networks we discovered that

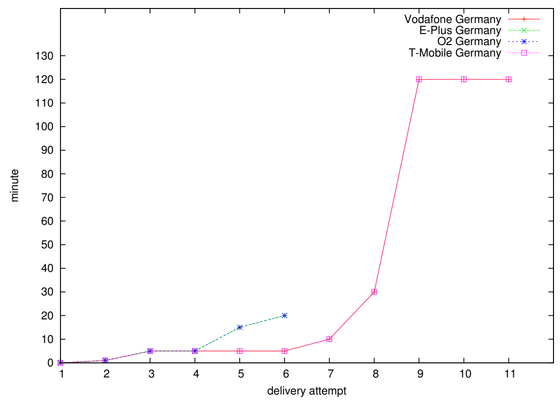

Figure 7. Timing of SMS message delivery attempts.

operators have different re-transmit timings. Further 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 IV-D.

Figure 7 shows the delivery timings for Vodafone, T-Mobile, O2 (Telefonica), and E-Plus. The initial delivery attempt is at minute 0. These show 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 7 shows Vodafone does an additional re-delivery 24 hours after the last delivery shown in the graph. O2 also attempts a 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 through sending multiple messages.

E. 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 pressure for money. Attacks could also be carried out for fun by script kiddys or a like. Below we discuss some scenarios that seem likely.

Individuals: Individuals could be pressed to pay some amount of money in order to keep their phone operational. Such as happened with the Ikke.A [43] 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 an premium rate number in order to be taken of 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 phone uses a SIM card with the security PIN enabled, as we describe in Section IV-I.

Operators: Operators could be threatened to have all their customers attacked. Such an attack would mainly kill the operators reputation as being reliable. Further the operator might loose money due to people being unable to call and send text messages – also for this to hurt the attack must be carried out on very large scale for a longer time. Maybe customers terminate their contract with the operator.

Further the operator's mobile network can be attacked directly or as an side effect of an large attack against it's users. This could work when thousands of attacked phones drop of the network and try to re-connect at the same time. This can cause an overload of the back-end infrastructure such as the HLR. A similar attack is described in [17]. This kind of attack seems likely since mobile networks are not optimized for this specific kind of requests. 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 6) 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 an replacement.

Public Distress: A careful placed attack during a time of public distress could lead to large scale problems and maybe even a panic. Such as happened to the country of Estonia [46] in 2007 when a group of people carried out a Denial-of-Service attack against the countries Internet infrastructure. Also in emergency situations of any kind having certain people without communication is bad. 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.

VI. Counter Measures

In this Section we present counter measures 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. Forth, we close with a brief rant on missing firmware updates for feature phones.

A. 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 is to monitor the phone 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 send 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 be possible to catch malicious SMS messages that cause phones to reboot and loose network connectivity. *This should especially help to catch unknown payloads that cause crashes.* Such a monitor is also capable to detect if a large attack is on the way by correlating multiple SMS-receive-disconnect events in a certain time-frame.

B. SMS Filtering

SMS filtering can be implemented in two ways. Directly on the phone and on the operator's network. Both possibilities have benefits and drawbacks, that we are going to discuss in the following:

Phone 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 just like an signature-based anti virus software or a packet filter firewall would do it. 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 like 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 in a other operators 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 it is called SMS homerouting, we discuss this briefly in the following paragraph. 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.

SMS Homerouting as specified in [47] only comes into play when the receiver phone is roaming.

Without homerouting the sender-side-SMSC queries the HLR for the SMSC of the receiver. The HLR tells the sender-SMSC that the receiver is currently roaming and the SMSC that is able to deliver the SMS message to the receiver. The sender-SMSC than delivers the SMS message to that SMSC which in turn delivers the message to the receiver.

With homerouting active the process looks like this. When sender-side-SMSC queries the HLR of the receiver the HLR always pretends that the receiver is in it's home network (not roaming), and, therefore, returns an fake-SMSC that is on it's own network. The sender-SMSC than sends the SMS message as if the receiver was not roaming. The SMSC on the receivers network than quires the HLR itself now the HLR returns the real SMSC for the receiver, one another operator's network. The SMS is forward to that SMSC which in turn delivers the message to the destination phone.

The important difference is that when homerouting is implemented all SMS messages that are send to the subscribers of an operator travel though that operator's SMSCs. Therefore, the operator is able to filter malicious SMS messages send to his customers. Without homerouting only phones that are on the home network can be protected.

C. Preventing Network Amplification

Attack amplification through re-transmissions of SMS messages should be avoided since this greatly helps an attacker. We suggest that operators should limit the number of re-transmissions. Some operators re-send the messages 10 times, this seems not necessary.

D. Firmware Updates

Firmware updates to patch known vulnerabilities is one of the best ways to eliminate security problems based on known issues. The problem with feature phones is that firmware updates are often not available and if available hard to install. If available, the operator often has a customized firmware, that again would need to be created after the firmware update. If an update exists for a particular device the user still has to know about it. This is rather difficult since feature phones, other than modern smart phones, don't notify the user about the existence of an update. Further, some phones can only be updated in specialized stores.

But firmware can also be updated Over-the-Air. This is called FOTA (Firmware Over-the-Air) defined in the Open Mobile Alliance Firmware Update Management Object (FUMO) [48]. Operators need to deploy and use this technology for pushing out firmware updates to feature phones that otherwise will not get updated by their owners.

VII. Conclusions

In this paper we have shown how to conduct vulnerability analysis of feature phones. Feature phones are not open in anyway, the hard and software are both closed and thus don't support any classical debugging methods. Throughout our work we have created analysis tools based on a small GSM basestation. We use the basestation 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 build 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. Further, 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 counter measures. 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 and Andreas Krennmair for reviewing early versions of this paper. Further we thank our colleague Borgaonkar Ravishankar and Simon Schoar for helping us buy mobile phones from India and supply us with phones for testing.

References

[1] Bernhard Müller. (2009) From 0 to 0-Day On Symbian. https://www.sec-consult.com/files/SEC_Consult_Vulnerability_Lab_Pwning_Symbian_V1.03_PUBLIC.pdf.

[2] Chuanxiong Guo, Helen J. Wang, and Wenwu Zhu, "Smartphone attacks and defenses," in *Third ACM Workshop on Hot Topics on Networks*, 2004.

[3] C. Mulliner and G. Vigna, "Vulnerability Analysis of MMS User Agents," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, Miami, FL, December 2006.

[4] Charlie Miller. (2007, August) Exploiting the iPhone. http://securityevaluators.com/content/case-studies/iphone/.

[5] Charlie Miller, Mark Daniel, and Jake Honoroff. (2008, October) Exploiting Android. http://securityevaluators.com/content/case-studies/android/index.jsp.

[6] C. Miller, C. Mulliner. (2009, August) Fuzzing the Phone in your Phone. http://www.blackhat.com/presentations/bh-usa-09/MILLER/BHUSA09-Miller-FuzzingPhone-SLIDES.pdf.

[7] Vincenzo Iozzo and Philipp Weinmann, "iPhone Safari vulnerability allowed to steal the SMS database," http://news.cnet.com/8301-27080_3-20001126-245.html, March 2010.

[8] 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.

[9] Tomi Ahonen, *Tomi Ahonen Almanac 2010 Mobile Telecoms Industry Review*, February 2010.

[10] (1996, October) Ping of Death. http://insecure.org/sploits/ping-o-death.html.

[11] C. Mulliner and 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.

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

[13] codenomicon. (2001) DEFENSICS. http://www.codenomicon.com/defensics/.

[14] Mobile Security Lab. (2009, January) SonyEricsson WAP Push Denial of Service. http://www.mseclab.com/?page_id=123.

[15] O. Whitehouse @stack Inc. (2003, February) Nokia Phones Vulnerable to DoS Attacks. http://www.infoworld.com/article/03/02/26/HNnokiados_1.html.

[16] B. Jurry XFocus Team. (2002, January) Siemens Mobile SMS Exceptional Character Vulnerability. http://www.xfocus.org/advisories/200201/2.html.

[17] W. Enck, P. Traynor, P. McDaniel and T. La Porta, "Exploiting Open Functionality in SMS-Capable Cellular Networks," in *Conference on Computer and Communications Security*, 2005.

[18] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, T. La Porta and 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.

[19] R. Farrow, “DNS Root Servers: Protecting the Internet,” 2003.

[20] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, “Inside the slammer worm,” *IEEE Security and Privacy*, vol. 1, pp. 33–39, 2003.

[21] Byers, Simon and Rubin, Aviel D. and Kormann, David, “Defending against an Internet-based attack on the physical world,” *ACM Trans. Internet Technol.*, vol. 4, no. 3, pp. 239–254, 2004.

[22] (2010, May) 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/.

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

[24] Tomi T. Ahonen. (2010, February) 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.

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

[26] ComScore. (2010, May) 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.

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

[28] Micromax mobile. http://www.micromaxinfo.com/.

[29] (2010, April) Micromax becomes the third largest handset manufacturer in India. http://www.topnews.in/micromax-becomes-third-largest-handset-manufacturer-india-2260105.

[30] ip.access Ltd. nanoBTS 1800. http://www.ipaccess.com/picocells/nanoBTS_picocells.php.

[31] Harald Welte. (2008) OpenBSC. http://openbsc.osmocom.org/trac/.

[32] 3rd Generation Partnership Project. (2004, September) 3GPP TS 23.040 - Technical realization of the Short Message Service (SMS). http://www.3gpp.org/ftp/Specs/html-info/23040.htm.

[33] 3GPP/ETSI. (1998) 3GPP TS 03.40 Technical realization of the Short Message Service. http://www.3gpp.org/ftp/specs/html-info/0340.htm.

[34] ——. (1998) 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.

[35] ——. (1998) 3GPP TS 03.38 Alphabets and language-specific information. http://www.3gpp.org/ftp/Specs/html-info/0338.htm.

[36] WAP Forum. (2002) WAP-209-WSP Wireless Application Protocol MMS Encapsulation Protocol. http://www.wapforum.com.

[37] (2005) Honeynet Project. http://project.honeynet.org.

[38] Bundesnetzagentur. Übersicht Numernraum. http://www.bundesnetzagentur.de/cln_1931/DE/Sachgebiete/Telekommunikation/RegulierungTelekommunikation/Nummernverwaltung/UebersichtNummernraum/UerbersichtNrnRaum_node.html.

[39] Das Örtliche. http://www.dasoertliche.de.

[40] Routo Messaging. http://www.routomessaging.com.

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

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

[43] 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.

[44] cellular-news. (2010, November) A ”rising Tide” of SS7 Based Mobile Network Fraud. http://www.cellular-news.com/story/46377.php.

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

[46] BBC News. (2007) Estonia hit by ’Moscow cyber war’. http://news.bbc.co.uk/2/hi/europe/6665145.stm.

[47] 3GPP. TR 23.840 Study into routeing of MT-SMs via the HPLMN.

[48] Open Mobile Aliance. (2007) Firmware Update Management Object. http://www.openmobilealliance.org/technical/release_program/docs/copyrightclick.aspx?pck=FUMO&file=V1_0_4-20090828-A/OMA-TS-DM_FUMO-V1_0_2-20090828-A.pdf.