

Vulnerability Analysis of MMS User Agents

Collin Mulliner and Giovanni Vigna
University of California, Santa Barbara, USA
{mulliner, vigna}@cs.ucsb.edu

Abstract

The Multimedia Messaging Service (MMS) is becoming more popular, as mobile phones integrate audio and video recording functionality. Multimedia messages are delivered to users through a multi-step process, whose end-points are the MMS User Agents that reside on the users' mobile phones. The security of these components is critical, because they might have access to private information and, if compromised, could be leveraged to spread an MMS-based worm. Unfortunately, the vulnerability analysis of these components is made more difficult by the fact that they are mostly closed-source and the testing has to be performed through the mobile phone network, which makes the testing time-consuming and costly. This paper presents a novel approach to the security testing of MMS User Agents. Our approach takes into account the effects of the infrastructure on the delivery of MMS messages and then uses a virtual infrastructure to speed up the testing process. Our testing approach was able to identify a number of previously unknown vulnerabilities, which, in one case, allowed for the execution of arbitrary code.

Keywords: Mobile devices, Mobile phones, Multimedia Messaging Service, Vulnerability Analysis, Fuzzing.

1 Introduction

Multimedia messaging is becoming increasingly popular among mobile phone users. Almost all new mobile phones support multimedia messaging, with the exception of phones specifically targeting the low-cost market. In addition, mobile phone service providers heavily subsidize multimedia messaging-enabled phones, because service fees represent an additional source of revenue.

Unfortunately, the Multimedia Messaging Service (MMS) is also open to abuse, and several mobile phone viruses exist which use multimedia messages to spread. None of the currently known mobile phone viruses exploit actual vulnerabilities, and, instead, they rely on social engi-

neering techniques to spread from phone to phone. However, it is just a matter of time before phone-based malware will be able to exploit flaws in mobile phone applications to spread from device to device without requiring any user action. Therefore, it is necessary to develop tools and techniques to improve the security of mobile phone applications.

In this paper, we present a novel approach to the vulnerability analysis of MMS User Agents, which are MMS client applications. To the best of our knowledge, no attempt has been made before to analyze or test Multimedia Messaging Service *User Agents* for vulnerabilities.

Analyzing mobile phone applications is difficult for several reasons. First of all, these applications lack decent documentation, and, in addition, mobile phone operating systems often do not provide sophisticated development kits or support for debugging. Further, analyzing a service like MMS requires access to a special infrastructure, namely the phone service network. As a consequence, the testing is costly (because one has to pay a fee for each message sent) and time-consuming (because of the delays introduced by the infrastructure).

We addressed these problems by building a *virtual MMS system* that fully simulates MMS message transfer to and from smart phone User Agents. The virtual MMS system is completely software-based and can be easily used by others who intend to perform the same kind of testing.

Vulnerability analysis of MMS User Agents was conducted using fuzzing. We chose fuzzing as the testing technique because we did not have access to the source code of the target application. In this paper, we present a detailed study of the MMS message format and the possibilities for fuzzing it. Further, we present our fuzzing methodology and the fuzzing tool we have developed. Our approach is general enough to be reusable for analyzing other MMS User Agent implementations.

So far, we have found several buffer overflow vulnerabilities in the tested MMS User Agent implementation, some of which are *security critical*, because they allow one to manipulate the program counter of the application's process. We exploited one of the vulnerabilities to inject

arbitrary code on the target device using an MMS message. This exploits represents *the first mobile phone-related remote code-injection attack*.

The contributions of this paper are the following:

- We introduce a novel methodology to test MMS-based applications that takes into account the sanitization characteristics of the MMS infrastructure.
- We developed a testing environment that allows one to perform MMS User Agent testing at higher speeds and without costs.
- We developed a tool that performs security testing of MMS User Agents through fuzzing. The tool found a number of previously-unknown vulnerabilities.
- We developed the first MMS-based remote exploit for mobile phones.

The rest of this paper is structured as follows: Section 2 presents related work. Section 3 describes the MMS architecture, its components, and how messages are transferred between clients. Section 4 gives an overview of what an MMS User Agent is, how it works, and some specifics of the User Agent implementation we tested. In Section 5, we identify the various inputs to a User Agent and describe our virtual MMS system. In Section 6 we present our MMS fuzzing tool, the methodology, and the results of our fuzzing approach. Section 7 presents a SMIL-based MMS exploit against PocketPC-based phones and in Section 8 we briefly conclude.

2 Related Work

Previous research on MMS client security was mostly conducted by companies that develop anti-virus products to detect malware that spreads over MMS messages (e.g., Commwarrior [5]). These viruses do not exploit vulnerabilities in the MMS software. Rather, they rely on social engineering techniques to lure the user into executing a malicious payload.

The tools developed so far use simple signature-based techniques to detect malicious SMS and MMS messages either on the phone or in the infrastructure. These tools suffer from the same limitations of OS-based anti-malware tools: a signature-based approach can only detect known malware based on samples collected “in the wild,” and, thus, needs continuous updating.

Other research works mainly focused on two components of the mobile phone infrastructure: the Short Message Service (SMS) and the Wireless Application Protocol (WAP) [19]. Three studies were performed on different mobile phone models from different manufacturers [8, 3, 11],

which revealed problems with the handling of binary SMS messages that lead phones to hang or reboot. These bugs could be used for Denial-of-Service (DoS) attacks against the vulnerable devices.

A study on the security of the SMS infrastructure [16] revealed that SMS messages sent from the Internet could be used to perform a Distributed Denial-of-Service (DDoS) attack against the mobile communication infrastructure of a large city. The attack leverages the delays in the store-and-forward message delivery architecture to overload the network. The study clearly demonstrated the difference between standard vulnerability analysis techniques and the novel techniques that need to be employed when testing applications over mobile phone networks.

Multiple security studies have been conducted on the WAP infrastructure, covering both client-side and server-side components of the architecture. Of particular interest are FuzzServer [10] and the PROTOS [12] test suite, which demonstrated the effectiveness of fuzzing in the security testing of mobile phone infrastructure components. FuzzServer [10] is a very simple fuzzer to analyze the gateway components of the WAP infrastructure by generating faulty headers fields (e.g., containing unusually long strings or strings containing formatting directives) in response to queries from a WAP gateway. The goal of these messages is to generate in the gateway application faults that might be associated with exploitable flaws. The PROTOS [12] test suite is a general fuzzing framework, which supports a number of different protocols. PROTOS uses message grammars to generate test cases that are likely to trigger faults in the tested application. In 2000, the creators of PROTOS conducted a study on multiple WAP Gateways and WAP-based browsers and managed to find flaws in most tested products [9].

Even though the results obtained are promising, both the aforementioned approaches do not address the security analysis issues that are characteristic of mobile phone infrastructures. In particular, our approach takes into account the modification and sanitization performed by the infrastructure to perform more focused security testing. In addition, our work focuses on the MMS infrastructure, which has been overlooked by previous research. We believe that there is a great need for effective tools that support third-party security testing of mobile phones and mobile phone network components. The rest of this paper presents a novel testing approach for this class of applications and a tool based on the approach.

3 The MMS Architecture

The goal of the Multimedia Messaging Service (MMS) is to support the exchange of messages between User Agent applications, which usually reside on mobile phones and are

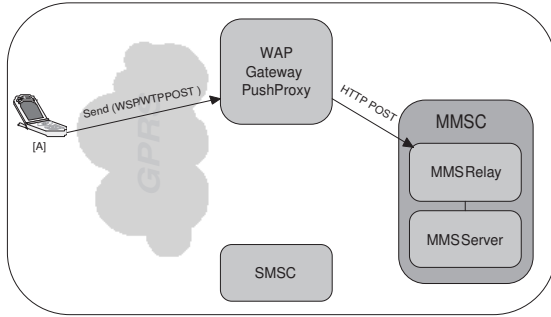


Figure 1. The MMS architecture and the message send process.

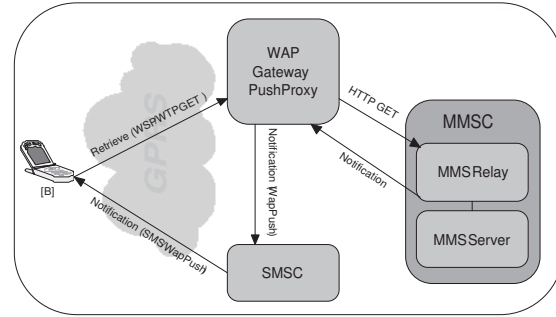


Figure 2. The MMS architecture and the message retrieval process.

operated by the users.

The MMS architecture is almost completely IP-based, and relies on both the HTTP [6] protocol and the protocols defined by the WAP architecture [19]. These protocols, in turn, rely on the transport mechanisms provided by the phone network to interact with the User Agent on the mobile phone.

The delivery of messages between User Agents is carried out by four components: the *MMS Server*, the *MMS Relay*, the *WAP Gateway/PushProxy*, and the *Short Message Service Center (SMSC)*. The MMS Server and MMS Relay together are commonly referred to as the *Multimedia Message Service Center (MMSC)*. The components and their relationships are shown in Figures 1 and 2 and explained hereinafter.

MMS Server. The MMS Server is responsible for storing the messages sent from the users and for deciding when the messages should be delivered to the recipients (e.g., based on service-level agreement parameters).

MMS Relay. The MMS Relay handles the actual message transfer using a number of different mechanisms, depending on the characteristics of the recipient. More precisely, it will use the WAP Gateway/PushProxy if the message is intended for a mobile phone user in the same network, an SMTP server if the message is intended for an email account, and the MMS Relay of another provider if the message is intended for a user of another network.

WAP Gateway/PushProxy. The WAP Gateway/PushProxy has two functions. First, it serves as a gateway between the user's mobile phone and the HTTP-based infrastructure. Second, it serves as a WAP PushProxy and delivers notifications (via *WAP Push* messages)

that are used to notify the user that a multimedia message is ready to be retrieved.

SMSC. The MMS Relay and the WAP PushProxy deliver WAP Push notifications to the user phones via the SMSC (Short Message Service Center).

3.1 MMS Message Transfer

The process of transferring an MMS message between a sender (A) and a receiver (B) is separated into two parts: send and retrieve. The send process, carried out by A, is shown in Figure 1, while the retrieval process, carried out by B, is shown in Figure 2. Our description assumes that both users' phones are using a GPRS connection in order to access the IP-based network of the phone service provider, and some details are omitted for clarity (e.g., the use of status information messages). A complete description of the delivery process can be found in [20, 22]. The message types mentioned in the description below are explained in more detail in the next section.

When sending an MMS, the user first creates a message and then requests the User Agent to deliver the message to the intended recipient. The User Agent then sends a WTP/WSP POST to the WAP gateway, which translates the WTP/WSP POST into an HTTP POST and forwards it to the MMS Relay.

The MMS Relay receives the message and then forwards it to the MMS Server. After that, the MMS Relay sends the reply to the POST request back to the User Agent using the WAP gateway as an intermediary. The reply contains information about the success or failure of the message submission. If the submission is successful, the reply contains a reference code that can be used later to match delivery notifications with a previously-sent message. The MMS message type used for sending a message is *M-Send.req* and the message type of the confirmation is *M-Send.conf*.

Transaction	Request Type	Result Type
Sending a message	M-Send.req	M-Send.conf
Receiving a message	WTP/WSP/HTTP Get.req	M-Retrieve.conf
New message notification	M-Notification.ind	M-NotifyResp.ind
Delivery Report	M-Delivery.ind	
Acknowledgment	M-Acknowledge.ind	

Table 1. MMS message types.

The delivery of the message to the final recipient is performed in two steps. First, the recipient's User Agent is notified that a new message is waiting for retrieval. The notification is generated by the MMS Relay and delivered as a WAP Push message transported via SMS to the recipient's phone. Second, the User Agent retrieves the message using a WTP/WSP GET request directed to the MMS Relay. The WTP/WSP GET is translated into an HTTP GET by the WAP gateway. The URL contained in the request (e.g., `http://mmsc.telco.com/mmsc/?msgid=47110815`) is used by the MMS Relay to retrieve the actual message from the MMS Server (the URL is part of the notification message). The message is returned to the User Agent in the body of the reply to the GET request. The messages used for notification and retrieval are called *M-Notification.ind* and *M-Retrieve.conf*, respectively.

3.2 MMS Messages

MMS messages are structured in a way similar to Internet email messages, and consist of a header and a body. The header contains control information, while the body represents the message content. The body is encoded using the MIME multi-part encoding scheme and mostly uses a `multi-part/related` structure. Messages transferred within the MMS infrastructure are encoded in plain text, while messages sent to and from a User Agent are in binary format (to reduce the size of the data during over-the-air transport). The encoding schema is the one defined by the WAP architecture [21].

The MMS architecture defines eight MMS message types or protocol data units (PDUs). These eight message types can be categorized in three groups: requests (denoted by the suffix `req`), confirmations (denoted by the suffix `conf`), which are used to indicate the result of a request, and indications (denoted by the suffix `ind`), which are used for asynchronous notifications. The types and formats are specified in [22]. Table 1 shows the message types associated with each operation.

We will focus on the two messages *M-Notification.ind* and *M-Retrieve.conf*, because these are the messages that are sent to a User Agent and could be leveraged to exploit a vulnerability in that component. Of these two messages,

Field Name	Content	Encoding
X-Mms-Message-Type	message type	1 byte
X-Mms-Transaction-ID	id string	string
X-Mms-MMS-Version	mms version	1 byte
X-Mms-Message-Class	message class	1 byte
X-Mms-Expiry	expiry time	long-integer
X-Mms-Message-Size	message size	long-integer
X-Mms-Content-Location	URL	string
From	sender	encoded-string
Subject	subject	encoded-string

Figure 3. The *M-Notification.ind* header.

Field Name	Content	Encoding
X-Mms-Message-Type	message type	1 byte
X-Mms-Transaction-ID	id string	string
X-Mms-MMS-Version	mms version	1 byte
From	sender	encoded-string
Content-Type	content-type	string and binary
Date	date	long-integer
To	receiver	encoded-string
Cc	carbon copy	encoded-string
Bcc	blind carbon copy	encoded-string
Subject	subject	encoded-string
X-Mms-Message-Class	message class	1 byte or string
X-Mms-Expiry	expiry date or delta	long-integer
X-Mms-Delivery-Time	date	long-integer
X-Mms-Priority	message priority	1 byte
X-Mms-Sender-Visibility	show sender	1 byte
X-Mms-Delivery-Report	delivery report	1 byte
X-Mms-Read-Reply	read indication	1 byte
Message-ID	message id	string

Figure 4. The *M-Retrieve.conf* header.

only the *M-Retrieve.conf* message has a body.

The format of the *M-Notification.ind* message and the type of binary encoding used when sent over-the-air is shown in Figure 3. The *M-Retrieve.conf* message is more complex than the *M-Notification.ind* message. The header fields of this message are shown in Figure 4.

4 The MMS User Agent

The MMS User Agent is the sending and receiving endpoint in the MMS system; it encodes, decodes, and renders MMS messages for the user. Due to the nature of the system, the User Agent application needs to interact with two different kinds of networks: First, the phone network for receiving WAP Push messages (via SMS), and, second, the IP-based network for sending and receiving the actual MMS messages using WTP/WSP/HTTP. Since the User Agent is not the only application that needs to receive WAP Push messages, an intermediate component handles all WAP Push messages and routes each message to the specific destination application, according to its content-type or WAP-Application-ID. The intermediate component is often called the *PushRouter*.

MMS User Agents normally have a few standard configuration options. With these options the user can decide if

messages should be downloaded immediately after receiving the notification, or if the download has to be explicitly requested by the user. These options are described in [20] as immediate and delayed retrieval, respectively. Other options concern the MMSC address (e.g., <http://mmsc.telco.com/mms>) and the WAP gateway IP address and port.

4.1 The PocketPC MMS User Agent

The User Agent we analyzed is *MMS Composer* (Version 2.0.0.13) from ArcSoft [2], which is the standard User Agent that is shipped with our test device, an i-mate PDA2k Phone¹. Other PocketPC-based smart phones use the same application, in different versions.

As mentioned above, the PushRouter handles all WAP Pushes on the device. Configuration information and the list of target applications for WAP Push messages can be found in the WindowsCE Registry at `HKEY_LOCAL_MACHINE\Security/PushRouter`. The User Agent application executable is `tmail`, which is executed by the PushRouter for each received WAP Push message with a content-type of `application/vnd.wap.mms-message`.

An important feature of the PocketPC PushRouter application is that it accepts WAP Pushes via both SMS and UDP on port 2948, which is the IANA-assigned WAP Push port. More interestingly, the UDP port is open on all network interfaces (e.g., the wireless LAN interface). This feature is leveraged by our virtual MMS system, which is described in Section 5.3.

The delivery of an MMS message to a PocketPC-based device is performed through a series of steps. First, the incoming WAP Push notification (*M-Notification.ind*) is delivered to the `tmail` application by the PushRouter. If the `tmail` application is configured for immediate download, it retrieves the message and displays the “new message” symbol in the status bar. If the application, instead, is configured for delayed retrieval, it first displays the “new message” symbol and then lets the user decide if he/she wants to download the message or not. The message download itself is performed through a WTP/WSP GET of the message URL, using the configured WAP gateway.

5 Analyzing the User Agent

The first step in the analysis of the MMS User Agent was to determine what kinds of inputs or attack vectors to the application existed. These inputs would then be used for

¹The i-mate PDA2k [7] is an OEM version of the HTC Blue Angel, a so-called “smart phone” running the Windows Mobile 2003 Second Edition operating system. The device is based on an Intel XScale PXA263 processor, which is an ARM CPU. The device is equipped with a wireless LAN (802.11b) interface, a Bluetooth interface, and multi-band GSM and GPRS services.

fuzzing the User Agent application. The second step was to determine if and how the messages used in the testing procedure were modified by the MMS infrastructure. The third step was to use the information gathered during the previous two steps to implement a virtual MMS system that would allow us to perform the security testing of the User Agent application without depending on the mobile phone network.

In the following sections we describe in more detail the vectors used to test the User Agent, the analysis performed to determine the effects of the MMS delivery infrastructure on the messages, and the design of the virtual MMS system.

5.1 Input to the User Agent

We identified four main input methods to an MMS User Agent. These four methods can be separated into two different categories: *active* and *passive*. Active methods can be triggered directly from a remote device, while passive methods require that the User Agent request the data (e.g., by initiating a GET request). The four input methods are described below. The first two belong to the active category, while the last two are passive. Note that the two passive inputs are two sections of one message, the first is the header and the second is the body. We consider them separately because the MMS infrastructure treats them in different ways. Also, none of the other message types have a body.

New Message Notification. This is the *M-Notification.ind* MMS message. The User Agent receives this message through a WAP Push. The message contains multiple strings specifying: sender, receiver, and the download URL for the actual message.

Delivery Indication. The *M-Delivery.ind* MMS message type, as the notification, is delivered through a WAP Push. The message has a simple structure, since it just indicates the delivery status of a sent message.

Message Header. This is the header of the *M-Retrieve.conf* MMS message. The message is delivered to the User Agent through the reply to a GET request. The header contains multiple fields with different formats.

Message Body. This is the body part of the *M-Retrieve.conf* MMS message. We considered the header and the body of the message separately because they are treated differently by the infrastructure. More precisely, while the MMS headers are actually checked by the various parts of the MMS infrastructure (and may lead to the message being rejected), the message body can be arbitrarily complex, and, therefore, it is more difficult to verify or sanitize.

The other MMS message types are output generated by the User Agent, and, thus, cannot be used to provide inputs to the application.

5.2 Sanitization in the MMS Infrastructure

All messages submitted to an MMS Relay are subject to verification and possible modification before being accepted for delivery. Messages failing the verification step are rejected and thus not delivered to their destination.

Because of this sanitization, a particular vulnerability may not be exploitable, because the message part that is used for an attack could cause the verification process to fail. In order to successfully attack an MMS User Agent, sanitization has to be avoided, and, thus, its effects have to be known. Identifying the sanitization rules of an MMS Relay, therefore, is an important step in the analysis process.

To identify the sanitization rules and the message parts that are not affected by the sanitization process, we tested each message part (e.g., header fields and body parts) individually by submitting specially-crafted messages to an MMS Relay.

Our fuzzing-like testing process works as follows: first, we create a list of message parts to be tested (e.g., the header fields *Subject*, *X-Mms-Message-ID*, and *Content-Type*); second, we define a number of applicable mutation methods for each message part (e.g., string generator, binary-string generator, or number generator); and, third, each part is individually tested.

The test procedure assigns one of five modes to each message part: *unusable*, *truncated*, *scrubbed*, *deleted*, and *not modified*. These modes specify how the infrastructure affects a message part. Initially all parts are marked as *not modified*. The test output consists of the list of message parts with the corresponding modes.

Testing each message part is done by first generating a value for the message part using one of the mutation methods. The message containing the generated value is then sent through the infrastructure, and, finally, the result of the submission is analyzed. If the submission is rejected, a new value for the message part is generated using the same mutation method and the message is sent again. If the message is rejected again, the mutation method is changed. If all mutation methods have been tried without success, the message part is considered unusable and the next message part is tested.

Accepted messages are retrieved and further analyzed. If the message part is deleted, truncated, or modified, the result is recorded and the next message part is tested. If the message part is not modified, then the same part is tested again using a value generated by the next mutation method. The next message part is tested after all mutation methods

have been tried or the part value is modified by the infrastructure.

At the end of the testing procedure, one has a precise idea of the effects of the delivery process on the contents of an MMS message.

5.3 The Virtual MMS System

The results collected by testing the sanitization process performed by the MMS infrastructure were used to design a virtual MMS system. The virtual MMS system is a testing harness that allows one to test a User Agent application using operational parameters that are identical to the ones observed when using the actual mobile phone network. The obvious advantages of using a virtual MMS infrastructure are the ability to control every parameter of the delivery process, the avoiding of usage fees, and the speeding up of the actual testing. By using our virtual MMS system we could speed-up the testing around 10 times. Delivery of a single MMS message only takes about 6 seconds in the virtual infrastructure while it takes at least 60 seconds when using a real service provider infrastructure. A further advantage of our virtual infrastructure is the possibility of testing message-parts that would normally be filtered/sanitized in a real infrastructure. Although these message-parts may not be exploitable in the real world, it is still important to test how this information is handled by the User Agent.

The virtual MMS system consists of three components: an HTTP server that acts as the MMS Relay, a WAP gateway, and the MMS message generator. The User Agent interacts with these components using a wireless LAN.

HTTP Server. We used Apache [1] with the addition of the MMS MIME type to the configuration, so that files with the `mms` extension are assigned the expected content-type (namely, `application/vnd.wap.-mms-message`).

WAP Gateway. We used the open-source WAP gateway software Kannel [15] without any custom configuration.

MMS Message Generator. The MMS message generator/fuzzer is based on MMSLib [14] a light-weight MMS encoder library. The fuzzer generates binary-encoded MMS messages and stores them in a directory accessible by the HTTP server, so that a client can access them.

To be able to use our virtual MMS system, the mobile phone needs to be configured to connect to the testing infrastructure instead of a regular mobile phone network. This is done by pointing the phone to the test WAP gateway and, for message access, to the web server. In addition, the phone

has to be configured to use the wireless LAN connection as the means to send and receive MMS messages.

To send a message to the phone, a message notification (*M-Notification.ind*) is transmitted using a WAP Push message encapsulated in a UDP datagram. The phone, in turn, connects to the WAP gateway of the virtual MMS system and receives the MMS message from our HTTP server.

6 Fuzzing MMS User Agents

We concentrated our fuzzing efforts around the *M-Notification.ind* and the *M-Retrieve.conf* messages types. In the *M-Retrieve.conf* case, we looked at the message body and the multi-part header. We also tested the SMIL implementation, since SMIL is an MMS-specific format [17].

The main component of our MMS fuzzing tool is the *fault-generator*, which generates the actual content that is encoded into the different fields of an MMS message. The *fault-generator* is part of the *MMS Message Generator*.

Setting up a fuzzing session involves two steps (assuming the device is already connected to the local wireless LAN). First, the User Agent application has to be started, and second, the debugger has to be attached to the target process (*tmail*). After these two steps have been completed, the fuzzing can be started. As soon as the application crashes, the two steps have to be repeated over before the fuzzing can be continued.

6.1 Fuzzing MMS Header Fields

The MMS header, as shown in Figures 3 and 4, is composed out of multiple variable-length fields besides the simple 1-byte-wide fields. The different fields use various kinds of encoding schemas in the binary MMS message format. The encoding schemas used are the main focus in this section.

6.1.1 Number Formats

We looked at three number formats: the `long-integer`, the `uintvar` (variable length unsigned integer) and the `value-length` (all specified in [21]). The `value-length` is heavily used for `encoded-strings`, and, therefore, needs special attention.

Long Integer. The `long-integer` is a multi-byte value where the first byte indicates the number of bytes composing the value. These bytes must be interpreted as a big-endian unsigned integer.

Variable Length Unsigned Integer. The `uintvar` format separates the number into 7 bit blocks with the remaining bit (the most significant bit) as a continue flag. If a value requires more than 7 bits, multiple bytes are

used where all but the last byte have the most significant bit set, to indicate a following byte. The maximum value length is 32 bit encoded in 5 bytes.

Value Length. The `value-length` format is either exactly 1 byte, or multiple bytes long. In the 1-byte format, a number between 0 and 30 can be represented. In the multi-byte format the first byte needs to be 31 and is followed by a `uintvar`.

The number formats are somehow complex and implementation errors in the corresponding parsing routines seem likely. Therefore, we designed test cases for each format. Along with the parsing tests, standard boundary condition tests were implemented. Below, we first present the parsing tests, followed by a short description of the boundary condition tests.

The `long-integer` format consists of a length byte followed by a number of data bytes. We anticipate that incorrectly written parsers overwrite a static buffer due to blindly copying the number of bytes given by the length field. Therefore, for each test case, the length and the number of data bytes is increased by one, starting with a length of zero.

The `uintvar` format does not have a range limit since it is terminated by a special character (like a NULL-terminated string), and, therefore, the format is tested like a string. Tests are divided into two parts, relatively short strings ranging in length from 1 to 255 and long strings matching common buffer sizes (256, 512, 1024, 2048, 4096, ...65535).

The boundary condition tests were conducted by legally encoding the test values into the relevant fields (e.g., the length field of an `encoded-string`). Our test values are based on the advice given by [13]. Basically three groups are tested: very small numbers (e.g., -1, 0, 1, 2, 10, 20, 30), very big numbers (e.g., 0xffff, 0x7fffffff, 0xffffffff) and numbers around the byte boundaries (e.g., 2^8 , $2^8 - 1$, $2^8 + 1$, 2^{16}).

6.1.2 String Formats

Strings are basically encoded as a sequence of characters terminated by a NULL character, with a few exceptions for special cases. The earlier-mentioned `encoded-string` is a combination of a normal string and a length field. The different string types are specified in [21] and are explained below.

Text-String. A `text-string` is a NULL-terminated string of characters. A leading quote character (decimal 127) is required if the first character is between 128 and 255.

Encoded-String. The `encoded-string` basically is an extension of the `text-string`. The `encoded-string` is either a `text-string` or a `value-length` followed by a `character-set` identifier and a `text-string`, to indicate the total length and the character-set used by the string, respectively.

String fuzzing is performed by a number of fuzzing tools [4, 10], and, therefore, we do not provide many details. In general all string fields were tested for buffer overflows using sequences of printable and non-printable characters of various lengths. Special test strings containing format directives, such as “%n”, were used to trigger possible format string vulnerabilities.

Encoded-strings also required more tests. Here, the length field is set to indicate fewer bytes than the string actually contains. Parsers that blindly accept the length indication would allocate a buffer to hold exactly the number of bytes indicated, and then use `strcpy` (since the string is NULL-terminated) to copy the string.

6.1.3 The Content Type Field

The *Content-Type* field has a special format and requires particular attention. The format of the field is defined in [21] (p.90), and consists of several subfields (parameters): a `value-length` (for the complete field) followed by the `content-type` itself, followed by the parameters. The parameters are encoded like message header fields: first the field name (encoded as 1-byte-value) and then the value (e.g., a NULL-terminated string). The `content-type` itself is either a 1-byte-value (in case of a “well-known” type) or a string.

We anticipated that User Agent implementations would be “optimized” for standard cases and would likely misbehave in non-standard cases. In our test cases, all parameters are treated as string fields and were tested by using the string length and format string tests described earlier.

6.2 Fuzzing the MMS Message Body

The MMS message body consists of `multi-part` entries. This means that each body part consists of a small header right before the actual content data. The individual body parts are concatenated. The format of the multi-part entry header is almost the same as the format of the *Content-Type* field in the message header and has two additional length fields.

For testing the multi-part entry header we used the exact same test cases that we created for the *Content-Type* field. We did not perform any content data fuzzing besides SMIL, which is described below.

```
<smil>
<head>
  <meta name="title" content="mms" />
</head>
<layout>
  <root-layout width="229" height="226" />
  <region id="Image" left="4%" top="2%" width="92%"
    height="80%" fit="hidden" />
  <region id="Text" left="4%" top="81%" width="87%"
    height="16%" fit="hidden" />
</layout>
</head>
<body>
  <par dur="5000ms" >
    <text src="1.txt" region="Text"/>
  </par>
</body>
</smil>
```

Figure 5. A SMIL file generated by MMS Composer.

6.3 Fuzzing SMIL

SMIL [17] along with WML [18] is the presentation layer of an MMS message, it describes how the multiple parts of a message (e.g., text, image, audio or video) are presented to the user. In other words, it is the HTML of MMS messages. Figure 5 shows a SMIL file generated by MMS Composer, the PocketPC MMS User Agent.

We concentrated our analysis on the most obvious problem: the length of field values. We ignored fields with common formats like `width` or `height`, since these also exist in HTML. We tried to avoid testing reused (tested) code by only testing SMIL-specific parts, where the code could not have been reused. Also, we only looked at fields that do not need any parsing, because of possible buffer size checks. We decided to focus our efforts on the `id` parameter of the `region` field and the `region` and `src` parameter of the `text` field. These fields were tested using the string tests described earlier.

6.4 Fuzzing Results

We used our virtual MMS system to deliver “fuzzed” MMS messages to the User Agent, and we found numerous string-length-related buffer overflows. We also found that the parser that handles the binary *Content-Type* values does not behave well and crashes when fed with unexpected values. In total we discovered more than 10 different fields whose parsing routines contain buffer overflows. Some of the buffer overflows are security-critical since they reach the stored return address on the stack, and allow one to hijack the program’s control flow. To demonstrate that some of the attacks found were exploitable we developed a proof-of-concept exploit for one of the vulnerabilities.

In the *M-Notification.ind* message we found that the length of the three header fields *X-Mms-Content-Location*,

Subject, and *X-Mms-Transaction-ID* is not handled correctly, leading to buffer overflows. We could use these overflows to perform a denial of service attack, only, since they do not allow one to overwrite the return address saved on the stack.

In the *M-Retrieve.conf* message parsing routines we found three buffer overflows. As in the *M-Notification.ind* message, the *Subject* can be used to crash the application. The other two overflows were found in the *Content-Type* field. In this case, the *content-type* itself and the *start-info* parameter trigger a stack overflow. Also, the *content-type* part overflow reaches the return address stored on the stack.

Additionally, three buffer overflows were found in the multi-part entry header of the message body. Here, the *content-type*, *Content-ID*, and *Content-Location* fields are not handled correctly. All three overflows reach the return address stored on the stack and can be used for gaining control over the program counter.

We further found multiple string-length-related overflows in the SMIL parser. In this case, the *id* parameter of the *region* tag and the *region* parameter of the *text* tag can be used to overflow the stack.

The results gathered from our tests clearly show that security was neglected while the particular MMS User Agent was developed; even very basic string length checks were not implemented.

7 Attacking MMS User Agents

Through our tests of the sanitization performed by the MMS Relay (see Section 5.2), we found it more likely that message-body-related vulnerabilities could be exploited in the real world. The reason for this is that the MMS Relay sanitizes some fields and converts the header fields of an MMS message to plain text. Thus, the infrastructure would remove the exploit from the message or reject the message altogether.

We have further investigated possibilities for circumventing the mobile phone service provider infrastructure in order to deliver malformed MMS messages to victim devices. The easiest way to accomplish this task is to run a malicious MMSC (e.g., an HTTP server with the configuration described in Section 5.3). Then, one would only need to send a notification message to the victim device containing a URL pointing to the malicious MMSC. The problem is that some phone service providers run closed MMS systems, where the WAP gateway cannot connect to any IP address other than the one of the MMS Relay. Therefore, closed MMS systems implicitly protect their users.

To be able to attack a device using a real-world infrastructure, we leveraged the lack of checks on the MMS message body. Therefore, we created a proof-of-concept exploit

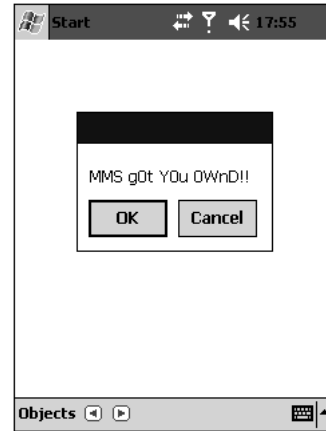


Figure 6. The Proof-of-Concept MMS Composer Exploit.

that executes code on the target device using the buffer overflow vulnerability found in the SMIL parser. We exploited the buffer overflow in the parser for the *id* parameter of *region* tag to overwrite the return address and hijack the program's control flow. The exploit displays a message box to show that the code was executed. Figure 6 shows the result of the execution of the MMS Composer exploit.

The malicious MMS message that contains the exploit/shellcode is sent like any other MMS message: it is submitted to the MMS Relay of the sender's mobile phone service provider, forwarded to the MMS Relay of the recipient's service provider, and delivered to the recipient. The MMS message is actually generated on and sent from a desktop computer that accesses the service provider's network via a GPRS dial-up connection.

Our exploit is the *first* to perform a remote code execution attack against a mobile phone using an MMS message as the attack vector. Vulnerabilities like the one we exploited for our proof-of-concept attack have serious implications, because they do not require user interaction in order to activate their payload, and, therefore, can be leveraged by worms that spread using MMS messages. In addition, vulnerabilities in MMS User Agents pose a more serious threat than the ones in traditional applications. In fact, MMS User Agents are always running, and, therefore, can always be attacked, while traditional network applications can only be attacked when the phone is connected to an IP-based network.

8 Conclusions

We presented a novel method for performing vulnerability analysis of smart phones that takes into account the side-

effects of the service infrastructure. Our method relies on a simulated infrastructure to avoid the cost and time factors normally associated with the use of mobile phone service networks. We developed a security testing tool that used the simulated infrastructure to identify a number of vulnerabilities in a commonly used implementation of an MMS User Agent. One of the vulnerabilities was exploited to create the first remote code injection attack that uses MMS messages as the delivery vector.

Future work will focus on the analysis of other User Agent implementations. Devices that support MMS transfer using wireless LAN can be easily tested using a setup like the one we presented in this paper. For testing devices that do not support a setup like ours, additional ways have to be found for delivering MMS messages to devices without using the infrastructure of a service provider.

Acknowledgments

This research was supported by the Army Research Office, under agreement DAAD19-01-1-0484, and by the National Science Foundation, under grants CCR-0238492 and CCR-0524853.

References

- [1] Apache Software Foundation. Apache HTTP Server. <http://httpd.apache.org>.
- [2] ArcSoft. MMS Composer. <http://www.arcsoft.com>, 2002.
- [3] B. Jury XFocus Team. Siemens Mobile SMS Exceptional Character Vulnerability. <http://www.xfocus.org/advisories/200201/2.html>, January 2002.
- [4] D. Aitel Immunity Inc. SPIKE The Advantages of Block-Based Protocol Analysis for Security Testing. <http://immunity.com>, 2003.
- [5] F-Secure. F-Secure Virus Descriptions : Commwarrior.A. <http://www.f-secure.com/v-descs/commwarrior.shtml>, 2005.
- [6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1, 1999.
- [7] i-mate. i-mate PDA2k. http://imate.com/t-DETAILSP_DA2K.aspx.
- [8] J. de Haas. Mobile Security: SMS and a little WAP. <http://www.itsx.com/hal2001/hal2001-itsx.ppt>, August 2001.
- [9] M. Laakso, M. Varpiola. Vulnerabilities Go Mobile. May 2002.
- [10] O. Whitehouse @stack Inc. FuzzServer. <http://www.blackops.cn/tools/FuzzerServer.zip>, January 2002.
- [11] O. Whitehouse @stack Inc. Nokia Phones Vulnerable to DoS Attacks. http://www.infoworld.com/article/03/02/26/HNnokiados_1.html, February 2003.
- [12] Oulu University Secure Programming Group. PROTOS Security Testing of Protocol Implementations. <http://www.ee.oulu.fi/research/ouspg/protos/>, 2002.
- [13] P. Oehlert. Violating Assumptions with Fuzzing. *IEEE Security and Privacy*, April 2005.
- [14] S. Hellkvist. MMSLib. <http://www.hellkvist.org/software/#MMSLIB>, 2004.
- [15] The Kannel Group. Kannel: Open Source WAP and SMS Gateway. <http://www.kannel.org>.
- [16] P. M. W. Enck, P. Traynor and T. L. Porta. Exploiting Open Functionality in SMS-Capable Cellular Networks. In *Conference on Computer and Communications Security*, 2005.
- [17] W3C. Synchronized Multimedia Integration Language (SMIL 2.1). <http://www.w3.org/TR/2005/REC-SMIL2-20051213/>, December 2005.
- [18] WAP Forum. Wireless Application Protocol Wireless Markup Language Specification. <http://www.wapforum.org>.
- [19] WAP Forum. WAP-210-WSP Wireless Application Protocol Architecture Specification. <http://www.wapforum.com>, 2000.
- [20] WAP Forum. WAP-206-WSP Wireless Application Protocol Multimedia Messaging Service Client Transactions Specification. <http://www.wapforum.com>, 2001.
- [21] WAP Forum. WAP-230-WSP Wireless Application Protocol Wireless Session Protocol Specification. <http://www.wapforum.com>, 2001.
- [22] WAP Forum. WAP-209-WSP Wireless Application Protocol MMS Encapsulation Protocol. <http://www.wapforum.com>, 2002.