AN OPENBTS GSM REPLICATION JAIL FOR MOBILE MALWARE

Axelle Apvrille
Fortinet, 120, rue Albert Caquot, 06410 Biot
Sophia Antipolis, France

Email aapvrille@Fortinet.com

ABSTRACT

There is one golden rule in the anti-virus industry that all AV analysts are very cautious about: making sure they do not spread samples which are under study. On PCs, vendors commonly use replication hosts in a very restricted environment (virtual machines, firewalls, limited network connection etc.).

Unfortunately the task is more complicated on mobile phones, both because fewer tools are available and because nearly all mobile viruses require either GSM or an Internet connection to operate correctly.

We have consequently built a fake GSM operator using the open source OpenBTS project to help us analyse mobile malware live while being sure the malicious programs are not inadvertently propagated on the network of a real operator.

This paper explains how we set up our GSM network and then how to use it for the analysis of mobile malware. Using recent mobile malware samples, we show how to trace calls or sniff SMS messages. We also enhance this GSM network with a firewalled Wi-Fi and explain how to deal with more advanced mobile malware which communicates with remote hosts on the Internet. Finally, we conclude with the current limitations and future work concerning this replication architecture.

1. INTRODUCTION

All anti-virus vendors follow a few implicit rules of good conduct. Among these, an important one is not to spread malware. A corollary is not to leak any information the malware gang might be using. For the daily routine of AV analysts, static analysis of malware complies with these rules and is unrestricted. At some point, however, all experienced analysts know they need to see the malware actually 'run', whether it is to help them reverse difficult parts or just go faster in their job. This requires much more caution and can only be done in a strictly confined environment. Typically, PC samples are run in virtual machines (*VirtualBox*, *VMware* etc.) or on physical replication hosts which belong to a specific and separate network, without any connection to the Internet or intranets.

The task is unfortunately much more complicated for malware on mobile phones. The main reason is that most mobile malware uses or propagates on the GSM network, which cannot be confined as easily as the IP network. GSM is based on SS7 [1], which is totally different from the traditional IP protocol stack. Hence, standard firewalls, proxies, DNS, packet sniffers etc. do not work over GSM. Furthermore, AV vendors cannot easily prevent access to GSM networks: it is not a matter of disconnecting a cable (GSM is 'over the air'). Calling the telecommunications operator to ask him not to cover the AV lab is obviously not possible either. The limited computing power of mobile phones, the variety of network interfaces (Bluetooth, USB, infrared, Wi-Fi...), operating systems and phone models are some of the other obstacles to controlling mobile malware.

Facing those problems in our AV lab, we decided to work on a solution and build our own fake GSM operator with a range limited to our lab. This fake operator is based on Universal Software Peripheral Radio hardware (USRP) and on an open source project named OpenBTS. The overall cost of the equipment is a little over US$1,000. We register our test lab phones to this fake operator and configure OpenBTS to retain all GSM traffic within our walls. Thus, we build a replication jail for mobile malware.

In this paper, we demonstrate the usefulness of such a jail for mobile malware analysis. In Section 2, we examine other ways to retain mobile malware and why they are either insecure or impractical. Then, we provide some background information on GSM networks and what functionalities the OpenBTS project provides on that network.

In Section 4, we present the architecture of our OpenBTS-based jail. We then explain how to trace calls or inspect SMSs a piece of malware sends. Some advanced mobile malware does, however, require access to the Internet to operate correctly. So, in Section 6 we explain how to deal with such malware without breaking golden AV rules. Finally, in Section 7, we present the results of our experimentation when using this mobile malware jail to analyse a set of malicious samples. In our conclusion we show how useful the work has been and the limitations we encountered, leading onto ideas for enhancing the work.

Throughout this paper, malware names correspond to *Fortinet*'s naming and can be looked up in *Fortinet*'s encyclopedia [2] for more details.

2. ALTERNATIVES

For the control and confinement of mobile malware during its analysis, several solutions can be contemplated.

We list below some solutions we considered and why we believe they are not fully adequate.

- **Removing the SIM**. A radical solution to keep the malware on the phone is to remove the phone's SIM card. In fact, this is not fully secure as Wi-Fi, Bluetooth and cable connections do not require the SIM card. However, this is not a blocking issue because those network interfaces are usually easy to turn off (by use of a specific menu or hardware button) and, in addition, they are seldom used by modern mobile malware. The last well-known worm using Bluetooth was SymbOS/BeSeLo, which already dates back to 2008.

  The real issue with removing the SIM card is simple: some phones, such as the *Nokia 6600*, just refuse to work without a SIM. In other cases, when the phone will operate without a SIM, the major problem is that many pieces of malware do not run properly without access to a GSM network, hence the SIM is required. This point is detailed in the next alternative, putting the phone offline, which shares the same problem.
- **Putting the phone offline or into flight mode**. Most phones offer a quick way to put the phone offline or into flight mode. The solution is apparently secure, except on

some phones the offline mode does not turn off Wi-Fi, Bluetooth and other interfaces, and in both offline and flight modes, a malicious program could switch the phone back online. On *Symbian* phones, for example, the offline mode is known as a profile (EProfileOffLineId) and it may be reverted to EProfileGeneralId (or EProfileSilentId for silent mode) by the function MProEngEngine::SetActiveProfileL(). This requires the WriteDeviceData privilege, but this is not usually a problem for malware as users tend to install any application they badly want whatever it asks for. Malware such as SymbOS/Downsis, SymbOS/NMPlugin, SymbOS/Yxes and SymbOS/Zitmo use this privilege. We haven't encountered any malware switching from one profile to another yet, but there is nevertheless a risk.

For analysts, the real problem with putting the phone offline is that many pieces of malware do not behave the same way when they have no GSM network available. On an offline phone, SymbOS/Album does not show it is trying to send two SMSs, SymbOS/Acallno – a trojan spyware – cannot be activated and hence does not show its malicious intent, and only one SMS (instead of two) is sent by SymbOS/Feixiang because the code gets stuck sending the first SMS and never gets to the code that sends a second one. All these examples are detailed in Section 7.

So, putting a phone offline may be (reasonably) secure in terms of confining the sample, but, as a side effect, it also hides many of its features.

- **Using Faraday cages**. This is physics: Faraday cages block electric fields and they can consequently be used to isolate a contaminated phone from the rest of the world. The problem with Faraday cages is a practical one: how to build a Faraday cage from which the AV analyst can see and operate the mobile phone.

  A researcher, Jed Daniels, made an attempt at building a semi-transparent Faraday cage in which he could place his laptop [3]. The resulting cage does successfully block electric signals, but the screen is difficult to read (which would be even worse for mobile phones) and – another issue – there is no access to the keyboard. This is a critical problem for analysing mobile malware as the analyst needs to press the phone's keys.

  To remedy this issue, it is possible to build a very large Faraday cage, like a room, and place all the laboratory's mobile equipment in there. The Berlin Institute of Technology uses one like this [4]. Be aware that such a cage is very expensive, not to mention the weight of the cage which may have an impact on the architecture of the building.

  Bare Faraday cages all face another problem: inside the cage, the phone is unable to reach a GSM network, and as we said previously this alters malware's behaviour. Consequently, there is no other solution than placing a standalone GSM operator in the cage such as an OpenBTS-based operator as detailed in this paper.

- **Using a virtual machine**. Virtual machines have always been favoured by malware analysts because they contain malware and are so convenient: they run on a PC; logs, traces and screenshots are easy to make etc. For mobile phones, the *Android Emulator* (http://developer.android.com/guide/developing/tools/emulator.html) has been used for numerous malware analyses. It boots any version of *Android*, and offers additional features such as sending fake SMSs, calls, geolocalization and logging.

  For Java midlets, the *J2ME Emulator* (http://www.oracle.com/technetwork/java/download-135801.html) is quite handy too. It requires a .jad and .jar to run the midlet. If the .jad is unavailable (frequent), it is easy to create a fake one from the .jar. A *BlackBerry* smartphone simulator (http://www.blackberry.com/developers/simulators) exists too.

  There are, however, two problems with virtual machines. First, as with the other solutions, the malware does not have access to a real telecom operator, and as we have already explained this modifies the malware's behaviour. For instance, if Java/Konov is run in the J2ME emulator, it loops trying to send an SMS to short number 4124 and never shows it would send an SMS to 4125, 1171 and 5537 too. Additionally, there are a few cases where the malware is expecting real hardware and fails in a virtual machine (e.g. Android/DrdDream). Sooner or later, too, we are likely to see anti-VM tricks in mobile malware (as we frequently encounter with PC malware), hence it won't run properly.

  Second, virtual machines unfortunately do not exist on all systems. Notably, there aren't any for *Symbian*. A project named Syborg, consisting of *Symbian* on QEMU, was started a long time ago, but it now seems to have been abandoned; its installation is complex, with many obsolete and broken links. *Symbian* developers may be aware of *Carbide*'s simulator, but unfortunately it is only a developer's tool and cannot run real packages (compiled for ARM). As far as I know, *iOS* faces the same problem: the *iOS* simulator is for development code only, and third-party simulators, such as http://iphone4simulator.com, are only suitable for testing web applications. It is of little help with analysing malicious samples.

- **Using an OpenBSC-based mobile malware jail**. This solution is very similar to the one presented in this paper. It uses the same idea: to build a standalone local GSM operator for test phones. The only difference is at implementation level: it relies on the open source OpenBSC project (http://openbsc.osmocom.org/trac), not OpenBTS, and must be coupled with a BTS such as a *Siemens* BS11 microBTS or an ip.access nanoBTS.

  This architecture (OpenBSC+nanoBTS) has never been used for mobile malware analysis, but has been used to test and fuzz mobile phones [5]. The only problem with this alternative is its price: about six times the price of the solution presented in this paper, with the same level of results. Indeed, a nanoBTS is sold for approximately US$6,200. Second-hand units have been seen on the market at US$2,800.

Table 1 summarizes solutions for controlling mobile malware. It compares them regarding overall simplicity, cost, risk of spreading or leaking information, and accuracy of malware analysis in such an environment. Most alternatives, apart from the Android Emulator and the OpenBSC-based solution, fail regarding that last criterion because they alter the malware's

Alternatives	Simplicity	Cost	Spreading and leak risks	Analysis reliability
Remove SIM	√	√ (free)	√	Bad
Offline	√	√ (free)	Slight risk	Bad
Faraday cage	Requires space/architecture	Over 5,000 USD	√	Bad
Android Emulator	√	√ (free)	Slight risk (web)	√ (reasonable)
Java ME Emulator	√	√ (free)	√	Bad
OpenBSC + nanoBTS	Requires some initial set-up	Approx. US$6,200	√	√
OpenBTS + USRP (this paper)	Requires some initial set-up	Approx. US$1,000	√	√

Table 1: Evaluating alternatives to an OpenBTS-based mobile malware jail.

behaviour. The fact that the Android Emulator is able to send/receive SMSs and calls is generally sufficient not to disturb behaviour too much (though not perfect). OpenBSC and OpenBTS-based solutions are the best regarding that criterion. An OpenBTS-based solution is favoured over OpenBSC for cost reasons.

3. GSM AND OPENBTS BACKGROUND

OpenBTS [6] is an open source Unix application meant to offer a GSM access point to mobile devices. Let's immediately highlight that the project is poorly named and it actually offers much more than a BTS.

For readers unfamiliar with GSM architecture, a basic illustration is provided in the upper part of Figure 1, and explained below. The mobile phone (called Mobile Station, or MS, in GSM terminology) talks to a BTS (Base Transceiver Station) via an air interface called Um. The BTS is hardware equipment with little 'intelligence'. It consists of a transceiver, power amplifier, antenna etc. Several BTSs are managed by a BSC (Base Station Controller). The interface between the BTS and BSC is called Abis. A BSC is a (smart) software component that handles the low level radio functions, and routes calls to an MSC (Mobile Switching Centre). The MSC is the entity that actually connects the calls to the networking subsystem. To do so, it communicates with various components such as the Home Location Register (HLR), Visitor Location Register (VLR) or, for SMS messages, the SMSC (Short Message Service Centre). An SMSC is a server that stores and forwards SMS messages to the appropriate recipient. For more information, see [7, 5].

The OpenBTS project helps shortcut this MS-BTS-BSC-MSC architecture. To do so, it uses a USRP (Universal Software Radio Peripheral). The USRP presents the GSM air interface to mobile phones. The USRP handles most of the BTSs (see the middle part of Figure 1) in the GSM architecture we described previously. Then, the source code of OpenBTS basically consists of linking the USRP to an *Asterisk* PBX (www.asterisk.org). It implements a few missing parts such as a software transceiver, and maps the rest as much as possible to SIP connections that *Asterisk* can handle. For example, each mobile device (more precisely each IMSI – the IMSI is a unique number stored in the SIM card) is seen by *Asterisk* as a SIP client. GSM locations are mapped to SIP registrations, call connections to SIP transactions etc. SMS messages are handled by an instant messaging extension to SIP (RFC 3428). In the end, the roles of the MSC, HLR and VLR, i.e. phone switching functions, calls and mobility, are performed by *Asterisk*.

For comparison purposes, the lower part of Figure 1 also displays how a nanoBTS-OpenBSC architecture maps to GSM. It is quite different: the role of the BTS is fully assumed by a nanoBTS (or a microBTS), and the rest of the GSM network is assumed entirely by OpenBSC.

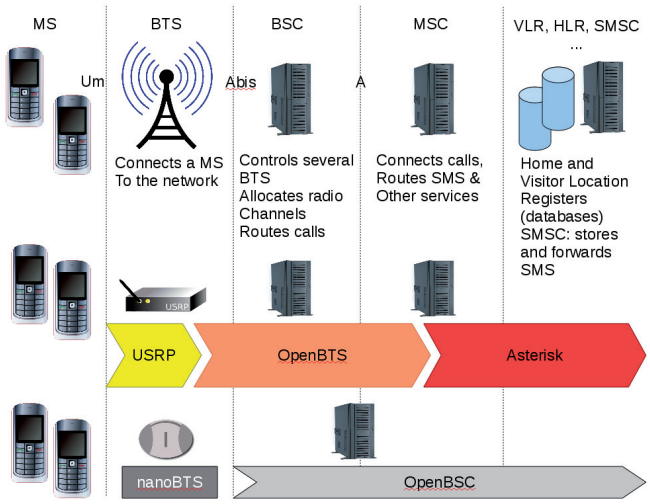

Figure 1: How OpenBTS maps the GSM architecture.

The OpenBTS project consists of approximately 100,000 lines of code. The project is alive and regularly maintained, but documentation is poor and the community is not always extremely responsive. Finally, for the purpose of mobile malware it is important to note that OpenBTS does not support GPRS, EDGE or UMTS, and there are no plans for it to do so (in the near future at least).

4. MOBILE MALWARE JAIL ARCHITECTURE

This section explains the architecture of our lab's mobile malware jail. It presents the general idea, and the main stages involved in building it, but does not go into the hardware and software installation details which are covered in [8].

Basically, the idea for this mobile malware jail is to create a standalone local GSM operator for a few test phones and without any connection to the external world. Consequently, when malicious samples are uploaded to the test phones (which correspond to mobile replication hosts) there is no risk of contaminating, damaging or disrupting anything other than those test phones.

In practice, the new GSM operator consists of a standard *Linux* host which runs *Asterisk* (VoIP PABX) and OpenBTS, attached to a USRP 1 motherboard with a daughterboard emitting in appropriate GSM frequencies (e.g. 1800MHz). The range of this local GSM operator is very limited (10 metres at most), which is perfect in our case because it restricts usage of this operator to our lab only (in-house, even neighbouring rooms do not see it). Hence, it does not disturb real operators and also complies with French regulations on the matter [9].

The equipment costs can be kept quite low: slightly above US$1,000 (excluding shipping costs), most of which is made up of the USRP mainboard and daughterboard. [8] provides an exact listing of all required elements. It also explains all the set-up procedures necessary to get such an operator up and working as outlined in the following steps:

1. Modify the USRP hardware to include a 52MHz clock[1], whose accuracy is compatible with GSM. With the default 64MHz clock onboard the USRP, the phones simply cannot place calls. Replacing the clock is strongly recommended.
2. Patch and build OpenBTS and its dependencies (e.g. GnuRadio, *Asterisk*). There are two important patches: one to support 52MHz clocks and the other to use a USRP with a single daughterboard[2].Overall, this step is quite time consuming because there are numerous dependencies, but all steps are explained in [8].
3. Get hold of a few SIM cards for the test phones. This step is optional because the OpenBTS network can work with real SIMs from real operators, but, from an AV analyst's point of view, it is a bad idea because there is a risk an infected test phone could be used in error on a real GSM network and not inside the mobile malware jail. With SIM cards using unused or test IMSIs, this risk does not exist. The generation of the IMSI on the SIM card can be done by a Python script such as [10].

Then, OpenBTS and *Asterisk* must be configured to achieve a GSM jail in which mobile malware is confined:

- Accept calls from the lab's test phones. This consists of configuring OpenBTS to use a GSM band the phones support, and operating with an MCC (Mobile Country Code) and MNC (Mobile Network Code) used by the test SIM cards. In *Asterisk*, the IMSI of each test phone is added to the configuration so that each of them is seen as a SIP user. Each IMSI is manually assigned a phone number. For example, below, IMSI 208301234567789 receives phone number 2102.

```
exten => 2102,1,Macro(dialSIP,IMSI208301234567789)
```

- Accept internal SMSs, i.e. SMSs from this network to a test phone (in this network). SMS messages are processed by a standalone executable, named smqueue, which is shipped with OpenBTS. Smqueue is not fully stable and I have usually had better results with the version from the achemeris/sms-split branch.
- Forbid (and monitor) any other call or SMS. Any call or SMS attempting to reach an undefined phone number in our local network or any phone number in another network must fail. Actually, this step is easy, because it is the default behaviour. By default, calls to unknown phone numbers are rejected and connection with another GSM network requires additional steps [11]. As for SMS confinement, configure OpenBTS to use a dummy SMSC address. The default value, 0000, is perfect for this.

```
# ISDN address of source SMSC when we fake out a
source SMSC.
SMS.FakeSrcSMSC 0000

# ISDN address of destination SMSC when a fake
value is needed.
SMS.DefaultDestSMSC 0000
```

The test phones require only a little configuration to register to this new GSM network. First, the phones must re-scan for available GSM operators – at which point, if OpenBTS is up and running and correctly configured, the phones should see a new operator. The name for the new GSM network may appear under different labels, depending on the handsets (see Figures 2 and 3).

Select the new GSM network. This procedure is possible on any mobile phone as long as it is not network locked. Search in menus such as 'Network' or 'Operators'.

Figure 2: iPhone scanning for local GSM operators. OpenBTS network is labelled '20830'.

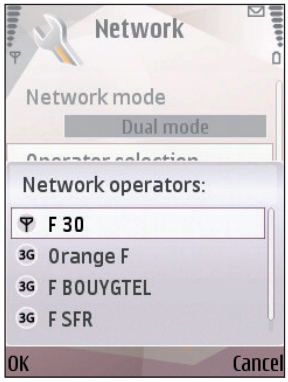

Figure 3: Nokia N95 scanning for local GSM operators. OpenBTS is labelled 'F30'.

Then, on each phone, set the address of the SMSC to be used (typically 0000) for the mobile malware jail. There are several ways to do this: send an AT command +CSCA [12]; use a code snippet to access a hidden menu (e.g. *#*#4636#*#* on

[1] This step isn't easy for a plain software engineer as it requires a few skills in electronics and equipment for very fine soldering. An electronics retail shop might be of some help at this stage.
[2] By default, OpenBTS uses two daughterboards, but it is possible to use a single one – which reduces costs.

some *Android* phones, or **5005*7672*SMSCNUMBER# on *iPhones*); or simply write a dummy SMS, try to send it and wait for the phone to ask which address of the SMSC to use.

From a security point of view, note that mobile malware which infects test phones is unable to propagate or affect other networks. On *Symbian*, in theory, a piece of malware could try to switch to another operator using the SelectNetwork method of the RMobilePhone class, but this API is only available from the *Symbian* Binary Kit and not available in the Public API. I have never encountered any malware using it. However, for more security, one can use a self-generated SIM card which has no authority over any real operator. In that case, even if the malware is shrewd enough to try to switch to another network, it won't ever actually manage to connect.

5. MONITORING GSM

Apart from confining mobile malware, an OpenBTS GSM jail is also useful for malware analysis to trace any calls and SMS messages the malware might silently send. Tracing SMS messages is usually the most interesting for mobile malware analysis. Whether the malware sends SMSs silently (without any pop-up on the phone) or not, the SMS cannot hide from smqueue's logs. A DEBUG log level is the safest way to avoid missing anything:

```
Log.Level DEBUG
Log.Level.smcommands.cpp DEBUG
```

In the logs shown in Figure 4, Java/GoSMS is a malicious midlet which trojans a car racing game. It is known to send SMS messages to short codes – at the victim's expense.

The smqueue logs show that the malware sends an SMS to the short code 3649 with the content #gubki 999. Actually, the logs display the contents of a bouncing SMS. Indeed, in that particular case, the short code 3649 does not correspond to any test phone in our lab, the SMS bounces and an error report is sent back to the originator.

Consequently, the sender (the infected phone – its IMSI is 208304424439206 and its phone number is 2103) receives a report saying the message could not be sent.

An AV analyst has two ways to read the content of the SMS: either check the smqueue logs (see Figure 4), or read new SMS messages in the inbox of the infected phone (Figure 5).

When using the achemeris/sms-split branch of the OpenBTS project, smqueue displays the fields of the SMS such as values for UDHI (User Data Header Indicator), DA (Destination Address), RD (Reject Duplicate), VPF (Validity Period Format), RP (Reply Path indicator) etc.

Figure 6 shows the smqueue logs of a phone infected with SymbOS/Zitmo.B!tr. This trojan spyware has the capability of intercepting SMS messages and forwarding them to a spy phone. It particularly targets mTANs used for online banking [13]. The logs show new information an analyst might not notice during static analysis: the trojan sends SMSs using a Data Coding Scheme equal to 8. This is not the default seven-bit encoding. It corresponds to UCS2 encoding, which is rather surprising, especially for an SMS being sent to a phone number in the United Kingdom (+44778XXXXXXX). This might eventually mean the malware author originates from Russia, or an Asian or Arabic country.

```
INFO 3074598592 smsc.cpp:121:sendSIP_init: from
        IMSI208304424439206 to 3649
..
DEBUG 3074598592 HLR.cpp:333:reloadDialplan:
        AsteriskHLR::reloadDialplan needReload=0 elapsed=1838992
DEBUG 3074598592 HLR.cpp:88:getAsteriskLine: getAsteriskLine
        cmd="dialplan show 3649@sip-local" tag="3649"
DEBUG 3074598592 HLR.cpp:58:getline: getline got: There is no
        existence of 3649@sip-local extension
..
NOTICE 3074598592 smqueue.cpp:1581:lookup_uri_imsi: Lookup
        phonenum '3649' to IMSI failed.
..
NOTICE 3074598592 smqueue.cpp:1162:bounce_message:
        Bouncing 191--xgtvp from 2103 to 3649: Phone not
                registered here. Message
DEBUG 3074598592 smqueue.cpp:845:set_qtag: Param tag=44740
DEBUG 3074598592 smqueue.cpp:626:validate_short_msg:
        MSG = MESSAGE sip:2103@127.0.0.1:5602 SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1:5063;branch=123
From: 411 <sip:411@127.0.0.1>;tag=44740
To: <sip:2103@127.0.0.1>
Call-ID: UuJ3J/@127.0.0.1
CSeq: 44740 MESSAGE
Content-Type: text/plain
Content-Length: 75
Can't send your SMS to 3649: Phone not registered here.
        Message: #gubki 999
```

Figure 4: Smqueue logs: short code 3649 does not exist – bounces SMS.

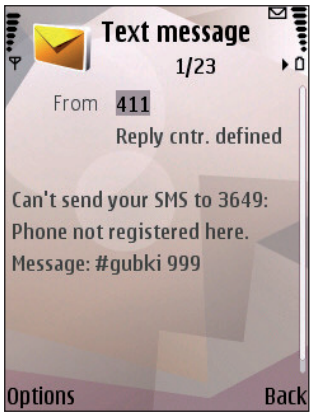

Figure 5: Bouncing SMS with original message sent by a phone infected with Java/GoSMS.

```
INFO 3074496192 smsc.cpp:231:submitSMS: from IMSI208304424439206
message: 1 RD=1 VPF=2 RP=0 UDHI=0 SRR=0 MR=254 DA=(type=
international plan=E.164/ISDN digits=44778XXXXXXX) PI=0 DCS=8
VP=(expiration=(Thu Jun 21 15:49:44 2012)) UD="DCS=8 UDHI=0
UDLength=32 UD=(0082000e000e00040096007600ce002e0086
0036003600a60026000400f600d6)"
```

Figure 6: SymbOS/Zitmo.B!tr sends an SMS using UCS2 encoding.

Apart from logging SMSs, the OpenBTS console[3] is also helpful for sending SMSs from any (fake) number. In the case of SymbOS/Zitmo.B!tr, it is thus possible to send SMS commands to the infected test phone. For example, we can set another test phone (extension 2102) to spy on SMSs from the infected phone.

```
OpenBTS> sendsms IMSI208304424439206 2102
enter text to send: set admin 2102
message submitted for delivery
```

Tracing calls can be done from *Asterisk*'s console. The example in Figure 7 shows one test phone calling another (ringing), then the status when the call is answered and a communication is taking place.

It is possible to display details for a given channel (sip show channel). For instance below, the transaction direction for the second channel is set as 'Incoming', which means that the corresponding user, IMSI 208300618462231, is the caller.

```
openbts*CLI> sip show channel 38480341@12
openbts*CLI>
 * SIP Call
 Curr. trans. direction:  Incoming
 Call-ID:                 38480341@127.0.0.1
 Owner channel ID:   SIP/IMSI208300618462231-08b3b208
 ...
```

The *Asterisk* console offers several other commands, such as 'calls' (another way to check for calls) and 'sip show peer' to get information related to a user. For example, it tells us IMSI 208300618462231 is assigned extension number 2111.

```
openbts*CLI> sip show peer IMSI208300618462231
openbts*CLI>
 * Name     :  IMSI208300618462231
 ...
 Callerid  :  "" <2111>
```

In the case of country-specific malware (e.g. SymbOS/Yxes in China), it might be useful to configure OpenBTS with a MCC/MNC of that particular country. This has not been tested yet.

Finally, although there hasn't been any use for sample analysis so far, it is possible to sniff all low level GSM packets. OpenBTS forwards GSM packets to the local host on the gsmtap port (4729). Figure 8 shows the details of a GSM Common Control Channel packet (CCCH). In particular, the packet shows it is using an MCC of 208 (France) and MNC 30 (Unused).

```
# asterisk -r
openbts*CLI> sip show channels
Peer        User/ANR     Call ID        Format              Hold  Last Message
127.0.0.1   IMSI208304   2339130f4c0    .. 0x80002 (gsm|h2  No    Init: INVITE
127.0.0.1   IMSI208300   38480341@12    .. 0x2 (gsm)        No    Rx: INVITE
2 active SIP channels

openbts*CLI> sip show channels
Peer        User/ANR     Call ID        Format              Hold  Last Message
127.0.0.1   IMSI208304   2339130f4c0    ..0x80002 (gsm|h2   No    Tx: ACK
127.0.0.1   IMSI208300   38480341@12    ..0x2 (gsm)         No    Rx: ACK
2 active SIP channels
```

Figure 7: A test phone calling another phone.

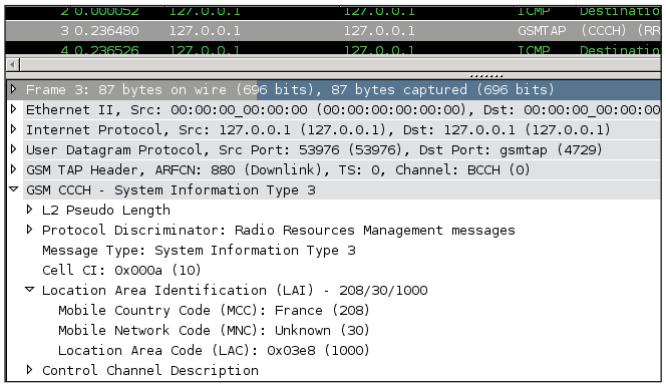

Figure 8: Wireshark v1.4.2 capture of GSM packets in the mobile malware jail.

6. MONITORING GPRS

Unfortunately OpenBTS does not support GPRS[4]. This is a problem when it comes to studying advanced malicious samples because they often require an Internet connection.

To counter this problem, our lab uses a Wi-Fi access point with a simulated Internet (see Figure 9). The lab test phones must be configured to use Wi-Fi when they need to connect to the Internet, and the Wi-Fi access point is placed behind a firewall which blocks all traffic to the Internet. The Internet can be simulated by a honeypot architecture or a fake DNS that redirects all packets to a fake web server. Any method used to contain PC trojans or botnets can be used.

In the case of Java/Vkonpass – a malicious midlet which fakes login to a famous Russian social network – the network capture (Figure 10) shows that the malware attempts to send the victim's credentials (identifier aa, password aa in the example) by email to ololoe2010@REMOVED.ru.

This GPRS workaround is not perfect. First, obviously, some old test phones do not support Wi-Fi at all. Second, some malware (e.g. SymbOS/Yxes) searches for an Internet connection where the data bearer matches WCDMA. As there is no such connection – only a Wi-Fi bearer access point – the malware is unable to connect to the Internet, which modifies its behaviour and consequently makes its analysis more difficult.

7. EXPERIMENTATION RESULTS

This section discusses the benefits and limitations of an OpenBTS-based malware jail as configured in the previous sections.

We measure the usefulness of this 'black box' analysis situation testing it against a recent set of malicious samples for *Symbian* and Java platforms. *Android* malware is not tested because (see Section 2) the Android Emulator is good enough for malware analysis purposes. The OpenBTS jail was not tested against *iPhone*, *Windows Mobile* or *BlackBerry* malware either. The reason is either because there are

[3] This feature only works in the main branch and is currently broken in the achemeris/sms-split branch.

[4] This feature is only available in a proprietary version of RangeNetworks. Note that OpenBSC supports GPRS/EDGE.

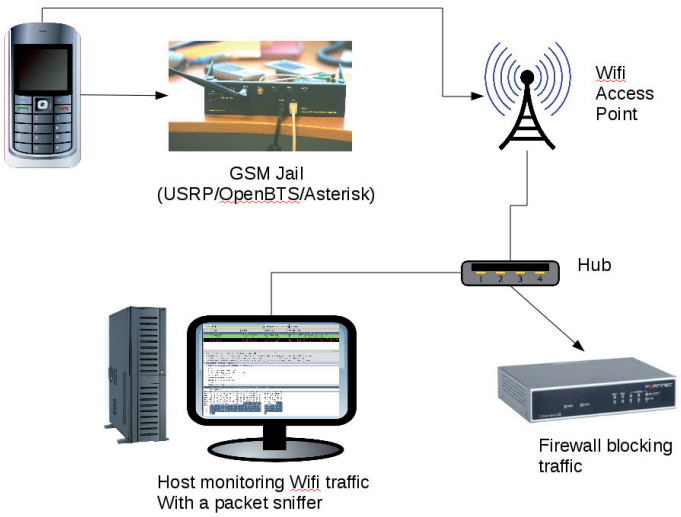

Figure 9: Workaround architecture for a GPRS jail.

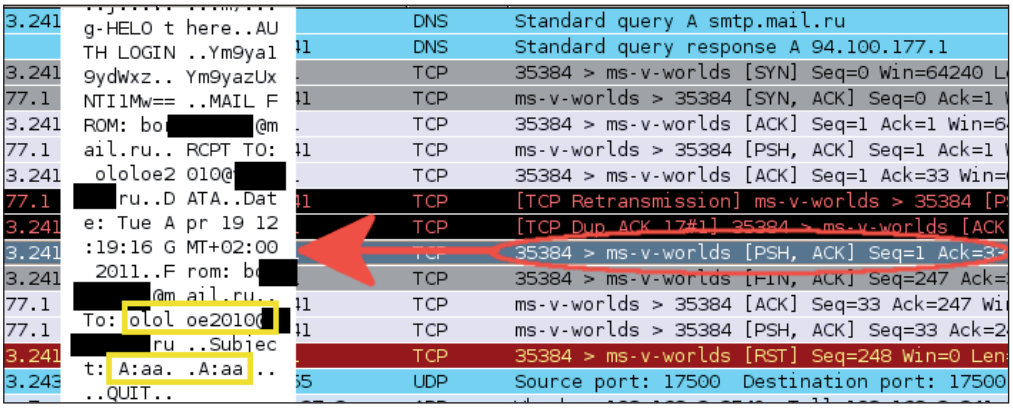

Figure 10: Wireshark capture of outgoing traffic from a test phone infected with Java/Vkonpass.

too few recent samples for those platforms, or because the samples we had were damaged or partial (not a fully installable package).

Figure 11 shows how much relevant information or benefit an analyst is able to gain from the study of a given sample in a given environment. By 'relevant information', we mean malicious key features the malware exposes, such as sending an SMS or contacting a remote web server. Bars in blue correspond to the amount of information found on a phone which is offline (no local operator). Orange corresponds to information found using an OpenBTS-based GSM jail (see Section 5). Yellow corresponds to information using an OpenBTS-based GSM jail and simulated Internet (see Section 6).

For example, a Java/ZoomSms sample on an offline phone displays a warning pop-up, asking for permission to send an SMS to 7122. From this behaviour, the analyst knows the malware sends SMS messages to 7122. However, as the phone is offline, even if the analyst grants permission to send the SMS, the phone is unable to successfully send the message. On some phones, this causes important system lags as the phone keeps trying to send the message. Furthermore, the analyst cannot see the content of the SMS: it is not stored in the Drafts box. But, when an OpenBTS-based GSM jail is used, the logs of smqueue show the contents of the SMS is vis 10326. This additional information is relevant for AV analysis. When a Wi-Fi access point is used in addition, the malware does not reveal any more information. As a matter of fact, Java/ZoomSms only sends SMS messages to short codes. It does not have any web activity, so all relevant information has already been found using the GSM jail.

So, to sum up findings for Java/ZoomSms, an offline environment sees it attempt to send an SMS, and an OpenBTS-based GSM jail sees the content of the SMS. For an analyst, one may consider that knowing a malware sends an SMS is a more important piece of information than knowing its content (this does not correspond to any rational formula, but more to a general feeling for malware sending SMSs to short codes), so Figure 11 displays a bigger blue bar and a smaller additional orange bar. There is no yellow bar (nothing new with Wi-Fi access), and the total of the bars add up to 100% as all relevant information has been found.

The experiment shows the following benefits of an OpenBTS-based GSM jail:

- It does not block SMSs. In several cases (e.g. SymbOS/Feixiang, Java/Konov, Java/GoSMS,

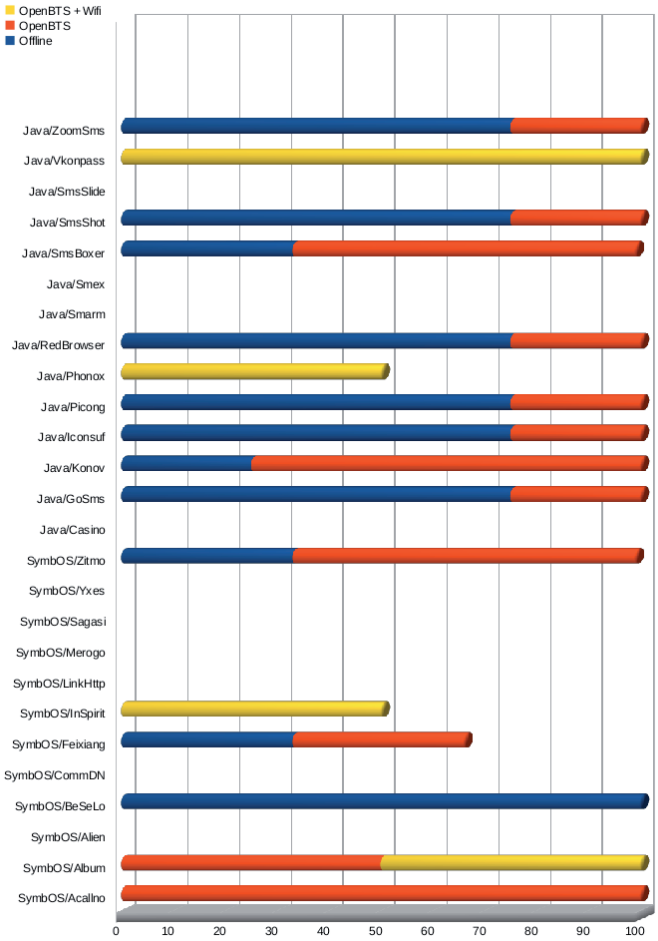

Figure 11: Approximate percentage of relevant AV analysis information gathered from samples on phones in three different environments: offline, OpenBTS, OpenBTS and Wi-Fi access points.

Java/RedBrowser, Java/SmsBoxer), the fact that the malware cannot send an SMS on an offline phone blocks the malware at this point. Consequently, AV analysts are unable to see the rest of the malicious behaviour of the malware – it sometimes also causes important system lags on the phone. When an OpenBTS-based GSM jail is used, the malware is not blocked, so the rest of the behaviour can be analysed. For example, one of our Java/Konov samples looped indefinitely trying to send an SMS to 4124 on an offline phone (and had to be switched off as this was so intense for the phone that it wasn't responding any longer). With the OpenBTS GSM jail, there was no such problem and the smqueue logs showed it was trying to send SMSs to 4124, then 4125, then 1171, then 5537 etc.

- Content of SMS. With the OpenBTS-based GSM jail, it is possible to read the content of the SMS, whereas it is usually not possible if the phone is offline. Smqueue logs show SymbOS/Feixiang sent a message: 'Hui fu #0# ; Yu zhong fei xiang', Java/IconSuf sends 'elzar lo', Java/ZoomSms sends 'vis 10326', Java/GoSms sends '#gubki 999', Java/Picong sends 'id96190264' etc.
- UCS2 encoding. It is possible to check the data coding scheme (and other parameters) of an SMS. The fact a message sends SMSs using UCS2 encoding is an indication that the malware targets or was written by an author in an Asian or Arabic country. For example, SymbOS/Album and SymbOS/Zitmo use UCS2.

If we access a simulated Internet via Wi-Fi, we gain:

- Name of remote hosts. For example, SymbOS/Album contacts hxxp://211.147.7.251 and hxxp://s.mmclick.com. SymbOS/InSpirit contacts hxxp://show.sj.91.com, Java/Phonox contacts REMOVEDnox.ru etc.
- Contents of the packets. The packet capture shows that SymbOS/InSpirit does HTTP POSTS. It also shows that Java/Vkonpass sends the victim's Vkonpass credentials to ololoe2010@REMOVED.ru.

Note, of course, that all this information can be found during static analysis. An OpenBTS GSM jail, with or without Wi-Fi access point, does not intend to replace static analysis (see Section 2), but black box analysis makes it easier than close reverse engineering of samples, especially when the malware uses encryption.

Figure 11 also shows some limitations to this mobile malware jail (blank lines). In some cases, the malicious behaviour of samples does not show at all or only partially. The reasons for this are:

- There is no test phone for that operating system. For example, SymbOS/NMPlugin requires a mobile phone running *Symbian OS 9.4* or greater and none were available at the time of the experimentation. In that case, the malware does not run at all, whenever offline or with an OpenBTS GSM jail.
- The sample is damaged or bugged. Static analysis of the malware may conclude that the malware has the capability to send an SMS, connect to HTTP etc., but because of a bug or a missing/damaged file in the package, this never occurs. This scenario happened with SymbOS/Shurufa for sure and probably with several other samples.
- MMSs are not handled. Consequently, any sample sending MMSs will fail to do so. This wasn't a big issue for SymbOS/BeSeLo though, because the message was deferred and consequently visible in the phone's pending outbox.
- Malicious behaviour only shows after the sample successfully contacts a given website or SMS short code, but they do not respond any longer. Indeed, malware is often active only during a short time frame. Afterwards, the remote ends are rented for other services, patched or simply deleted. The remote website and SMS short code can be simulated (fake web server, fake SMS), but only if we know what they were supposed to do, which is not always the case. If not, the full behaviour of the sample cannot be reproduced.
- Sample requires Internet connection using a WCDMA bearer, not a Wi-Fi access point. In that case, the sample fails to contact a remote server and is unable to show the rest of its behaviour to the analyst.

This limitation has already been discussed in Section 6 and it is the cause of several failures such as with SymbOS/LinkHttp and SymbOS/Yxes.

8. CONCLUSION

Mobile malware confinement is a big issue for anti-virus laboratories because mobile phones have numerous ways to escape the confinement, be it GSM, Wi-Fi or Bluetooth networks. So far, most solutions have shown big drawbacks: Faraday cages and nanoBTS are expensive, simulators or emulators are limited, and removing the SIM card or putting the phone offline alters the malware's malicious behaviour, making analysis less accurate and more difficult.

Building a local GSM operator which traps mobile malware looks like a better solution. This is now feasible, at a reasonable cost, using the OpenBTS project. It consists of having USRP hardware to present the GSM air interface to test phones in the laboratory, then the OpenBTS source code basically translates GSM communication into SIP connections which are handled by an *Asterisk* PABX. Mobile malware has no way to evade this local operator, especially if mobile phones use SIM cards which are only valid on that network.

This mobile malware jail has been tested on several malicious samples with good results. Malware analysis showed relevant information (SMSs sent to several short codes, message contained in the SMS, encoding used etc.) which would have been hidden in other replication environments. For malicious samples which require a connection to the Internet, the Internet is simulated via fake DNS and web servers behind a Wi-Fi access point. This works around the fact that OpenBTS does not support GPRS. The system is not perfect – in particular, some malware won't work over Wi-Fi – but provided interesting results in some cases, for example, Java/Vkonpass and SymbOS/Album.

The OpenBTS project is still evolving and there are several ways in which it can be improved to achieve a better mobile malware jail. Its first and most obvious limitation is that it does not support GPRS/EDGE. Thus, organizations with higher budgets might consider using OpenBSC instead because this other open source project supports those technologies. In practice, having real GPRS/EDGE support would only solve one among many other typical problems with analysing advanced mobile malware, such as the fact that malicious websites have a short lifespan.

Some other improvements would be useful to analysts, such as decoding UCS2 in SMSs, better logging of SMS messages, bug fixes, and an MMS server.

Globally, this mobile malware jail should be useful to AV analysts in addition to static analysis, particularly in situations where malware obfuscates its executables or uses encryption to conceal SMS messages, recipients or remote hosts it contacts.

9. ACKNOWLEDGEMENTS

I wish to thank Guillaume Lovet (*Fortinet*), who is always an excellent reviewer of my papers. I also thank Alexander Chemeris (*Fairwaves*), an active contributor to OpenBTS, for his review of the OpenBTS sections. Despite his own workload, he was kind enough to send me back helpful comments the next day! Finally, I shall always owe special thanks to Alexandre Becoulet (*Telecom Paristech*) and Andy Fung for technical help with setting up my USRP. I am not sure I would have made it without them.

REFERENCES

[1] ITU-T. Introduction to CCITT Signalling System No. 7, March 1993. Recommendation Q.700 (03/93).

[2] *Fortinet*. Fortiguard Encyclopedia. http://www.fortiguard.com/encyclopedia/index.html.

[3] Daniels, J. Faraday Cage, May 2007. http://www.jeddaniels.com/2007/faraday-cage-part-1/.

[4] Redon, K.; Borgaonkar, R. Femtocells : Inexpensive devices to test UMTS security. In Hackito Ergo Sum 2011, April 2011.

[5] Golde, N.; Mulliner, C. SMS-o-Death: from analysing to attacking mobile phones on a large scale. In CanSecWest 2011, March 2011. http://cansecwest.com/csw11/smsodeath_mulliner_golde_cansecwest2011.pdf.

[6] Burgess, D.A.; Samra, H.S. The Open BTS Project, August 2008. http://www.ahzf.de/itstuff/papers/OpenBTSProject.pdf.

[7] Yousaf, M.; Paudyal, U.; Vehkajarvi, T.; Wu, W.; Oza, T.; Muthuswamy, P.; Janagarajan, P.; Drake, J.; Zhu, W. OpenMSC. Technical report, Uppsala University, Department of Information Technology, 2010.

[8] Apvrille, A. OpenBTS for dummies v0.5, April 2011. http://gnuradio.org/redmine/attachments/215/fordummies.pdf.

[9] ARCEP. Les acteurs non soumis à la déclaration (in French), September 2005. http://www.arcep.fr/index.php?id=8055#c7795.

[10] Munaut, S. pySIM. http://cgit.osmocom.org/cgit/pysim.

[11] Independent Network With Roaming to Standard GSM. http://gnuradio.org/redmine/wiki/gnuradio/OpenBTSNetworkIntegrationIndependent.

[12] Setting or Reading the Service Center Address / SMSC Address (AT+CSCA). http://www.developershome.com/sms/cscaCommand.asp.

[13] Apvrille, A.; Yang, K. Defeating mTANS for Profit. Virus Bulletin, pp.4–10, March and April 2011.