Forensic Analysis On Android: A Practical Case

GIAC (GMOB) Gold Certification

Author: Angel Alonso-Parrizas, parrizas@gmail.com

Advisor: Jose Selvi

Accepted: October 13th, 2015

Abstract

Mobile platforms have grown in the last few years very quickly. At the same time, vulnerabilities and malware have evolved affecting the new mobile landscape. In order to respond to this new set of threats it is necessary that existing security techniques and tools adapt to the new situation. As a result, the current techniques, tools and processes to perform forensic analysis in networks and systems, need to cover also mobile platforms. In this paper it will be discussed how it is possible to perform forensic analysis in Android platforms covering the following aspects: the evidences in the logs, the network traffic, file system and in particular the analysis of the memory. A real malware case is investigated using the above aspects.

1. Introduction

"Digital forensic is a branch of forensic science focusing on the recovery and investigation of raw data residing in electronic or digital devices. Mobile forensics is a branch of digital forensics related to the recovery of digital evidence" (Bommisetty, S., & Tamma, R. (2014))".

The first part of the paper focuses on different processes that can be used to analyze Android from a forensic point of view. Through those steps, existing tools and techniques from traditional forensic analysis are used to investigate the systems logs, the network traffic, the file system and the memory. A key area of analysis is the memory, since it is possible to get many types of evidence and useful information, like crypto keys or unencrypted information.

In the second part of the paper, the set of steps described in the first part are applied to a real case. A fresh malware sample from the 'emmental' campaign (Cybercriminals to Online Banks: Check -. (2014, July 22)) is executed on an Android Device, and the forensic analysis is performed.

2. Android forensic analysis: the method

For the purpose of this research, the forensic analysis focuses on four core layers: device logs, network traffic, file system and memory. The evidence gathered in each step can cross-correlate with the information obtained in other steps in order to have a full picture.

The lab is composed of the following items: a MacBook Pro with the SDK Android toolkit, a Virtual Machine running Ubuntu 14.04, a rooted Nexus 5 running Android 5.1.1, and a WiFi pineapple which provides Internet access to the smartphone.

2.1. System logs

Android provides a full set of tools for log analysis; being 'adb logcat' (Logcat. (n.d.)) the most relevant. With this tool it is possible to monitor different kinds of logs, for example the events produced by the 'radio'. This information can be very useful to detect incoming and outgoing SMS/MMS messages. For example, malware that sends SMS in the background and deletes the messages in the SMS application, can be spotted.

2.2. Network traffic

The analysis of the network traffic is key in some forensic investigations, hence it is important to be able to monitor the traffic properly. In a corporate environment there might be existing tools like proxies or firewalls where the traffic is being monitored. A different approach to remotely monitor Network Traffic was discussed by the author on the paper "Monitoring network Traffic for Android Devices" (Alonso Parrizas, A. (2013, January 16)).

In a forensic lab it is worth to setup a very simple infrastructure to monitor all the network traffic. In the case of this paper, the author uses a WiFi pineapple Mark V (WiFi Pineapple (n.d.)) connected physically to a MacBook through Ethernet and also acting as a WiFi Access point for the smart phone. This setup permits to monitor the traffic with two main tools: Wireshark (Wireshark. (n.d.)) and Burp Suite (Burp Suite. (n.d.)). Burp Suite's certificated is imported in the Android device in order to do SSL inspection.

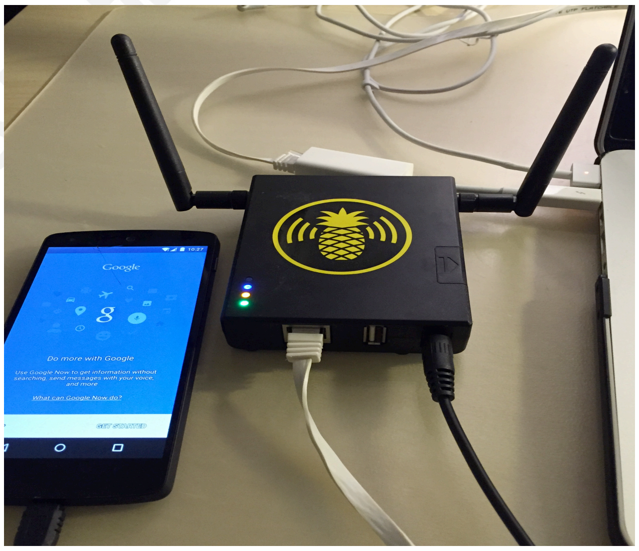


Figure 1: Networking setup

2.3. File system

In mobile devices, the data stored in the file system can be gathered in several ways as discussed in Practical Mobile Security (Bommisetty, S., & Tamma, R. (2014)): manual extraction, logical extraction, physical extraction (Hex dump), chip off and micro read.

In the case of this research the data is extracted logically. A full image of the file system can be gathered through different ways, but in the case of this project, and for simplicity reasons, the author boots the system in "recovery mode" with ClockworkMod (ClockworkMod (n.d.)). This permits to do a full image backup of the whole file systems and afterwards to pull the data through 'adb' (Android Debug Bridge (n.d.).

2.4. Memory

The analysis of the memory on a live system provides a lot of useful evidence and information, for example crypto keys or unencrypted data.

The first step is always to dump the memory of the target system. This can be done with some existent tools like LiME, Linux Memory Extractor (504ensicsLabs/LiME. (n.d.)), which can be cross compile for Android.

Moreover, the existence of memory analytics tools like Volatilty (Volatility Foundation. (n.d.)) enables to analyze the memories dumps. For that purpose, create a customized profile for the Android kernel running on the target device.

It is worth to mention that there are other approaches to analyze the memory in Android, like using the 'monitor' tool provided by Android. However, this method only allows monitoring one process at each time, instead of analyzing the whole memory of the system.

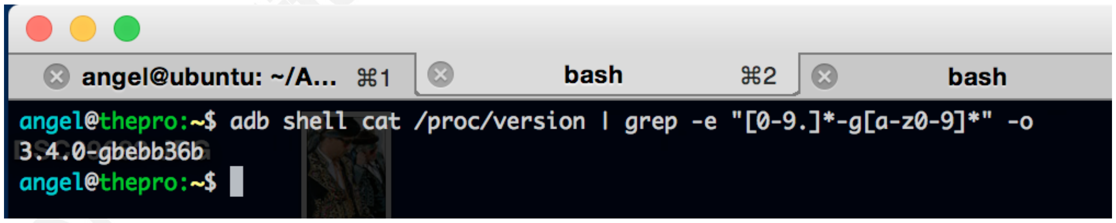
3. The forensic process: a real malware case

In order to test the proposed methodology, a real malware is executed on the smartphone. The APK to be executed, named CreditSuisse-SmsSecurity-v-20_08.apk (986d67fdff01c836be442fac5712ceaa), is a fresh and not detected sample (in VirusTotal) by the time of this analysis (5th of September 2015 19:51 CEST). With this, we ensure that the C&C server is up running and we get a real live communication. Two days after this investigation was performed, the malware was seen reported in VirusTotal (https://www.virustotal.com/es/file/b57b59e41c59c71e46699dd7219a1b2a64cf1d26b18c187427fe146dd7555acd/analysis/)

3.1. Preparation of the lab

Following a brief explanation of the architecture in section two, here a more detailed description is highlighted. The physical system is a MacBook Pro (MacOSX 10.10.15) with Android SDK installed through brew (Homebrew. (n.d.)) which permits to interact with the smartphone through ADB.

By default the kernel running in Nexus Android 5.1.1 (3.4-0-gbebb36b) does not support loading kernel modules, which it is necessary in order to load LiME. As a result it is required customize the kernel and compile it. Once the kernel is compiled, the smartphone needs to boot with the new kernel. The Ubuntu Virtual Machine is used for the compiling. The full set of commands for the compilation is described in the appendix.



```
angel@thepro:~$ adb shell cat /proc/version | grep -e "[0-9.]*-g[a-z0-9]*" -o
3.4.0-gbebb36b
angel@thepro:~$ 
```

Figure 2: Default kernel version running on Nexus 5 with Android 5.1.1

The Ubuntu VM is also used for the cross-compilation of LiME (the full set of commands is in the appendix). Once the LiME module has been compiled, the module ‘lime.ko’ can be pushed to the smartphone with ‘adb pull’ and loaded through ‘insmod’ (as explained in the appendix).

The last step is to create the Volatility profile based on the customized kernel (the commands are in the appendix).

On the other hand, to setup the network, it is necessary to share Internet in MacOSX with the WiFi Pinneaple as described in “How To: Configure a WiFi Pineapple For Use With Mac OS X” (How To: Configure a WiFi Pineapple For Use With Mac OS X. (n.d.)). The remaining commands that need to be executed are for redirecting all the HTTP/HTTPS traffic coming from the WiFi pineapple to the Burp Suite proxy.

```
echo "
rdr pass inet proto tcp from any to any port 80 -> 127.0.0.1 port 8080
rdr pass inet proto tcp from any to any port 443 -> 127.0.0.1 port 8080
" | sudo pfctl -ef -
```

3.1.1. Installing and running the APK

The next step is to install the APK file and run it, which is done with the command ‘adb install CreditSuisse-SmsSecurity-v-20_08.apk’. Then, run the newly installed application on the device while 1) monitoring the logs, 2) sniffing the network traffic and 3) capturing the memory dump.

3.1.2. Monitoring the logs: logcat

The main objective of logcat in this analysis is to gather evidence that SMS or MMS messages are sent. This can be detected through the radio logs. The key idea behind this is to detect malware abusing the SMS/MMS service, while monitoring the radio for some specific logs. The author noticed in the past some malware, which abused the SMS in order to subscribe to premium SMS services, but was not detected by common detection tool.

The following commands will detect any SMS going through the radio:

```
adb logcat -v threadtime -b radio RILJ:V GsmSMSDispatcher:V SMSDispatcher:V  *:S
adb logcat -v threadtime -b radio RILJ:V GsmInboundSmsHandler:V
```

```
SMSDispatcher:V  *:S
```

In the case of this specific APK, there was no SMS sent, hence no evidence on the logs. However, other information is gathered from the logs ‘main’ and ‘event’. Just to highlight some of them:

```
        09-05 19:51:23.218   772   873 I PackageManager: Running dexopt on:
/data/app/org.thoughtcrime.securesms-1/base.apk pkg=org.thoughtcrime.securesms
isa=arm vmSafeMode=false


09-05 19:51:57.703   772   867 I ActivityManager: Displayed
org.thoughtcrime.securesms/.FirstActivity: +746ms (total +27s201ms)
        09-05 19:52:51.292 17145 17440 W ImportFragment:
org.thoughtcrime.securesms.b.ab
        09-05 19:52:51.292 17145 17440 W ImportFragment:        at
org.thoughtcrime.securesms.b.af.a(Unknown Source)
        09-05 19:52:51.292 17145 17440 W ImportFragment:        at
org.thoughtcrime.securesms.b.af.a(Unknown Source)
        09-05 19:52:51.292 17145 17440 W ImportFragment:        at
org.thoughtcrime.securesms.cx.a(Unknown Source)
        09-05 19:52:54.320 17145 17441 W ImportFragment:
org.thoughtcrime.securesms.b.ab


09-05 19:52:26.711 17145 17441 W MmsSmsDatabase: Executing query: SELECT _id,
body, read, type, address, address_device_id, subject, thread_id, status, date_sent,
date_received, m_type, msg_box, part_count, ct_l, tr_id, m_size, exp, st, transport_type
FROM (SELECT DISTINCT date_sent * 1 AS date_sent, date * 1 AS date_received, _id,
body, read, thread_id, type, address, address_device_id, subject, NULL AS m_type,
NULL AS msg_box, status, NULL AS part_count, NULL AS ct_l, NULL AS tr_id,
NULL AS m_size, NULL AS exp, NULL AS st, 'sms' AS transport_type FROM sms
WHERE (read = 0) UNION ALL SELECT DISTINCT date * 1000 AS date_sent,
date_received * 1000 AS date_received, _id, body, read, thread_id, NULL AS type,
address, address_device_id, NULL AS subject, m_type, msg_box, NULL AS status,
part_count, ct_l, tr_id, m_size, exp, st, 'mms' AS transport_type FROM mms WHERE
(read = 0) ORDER BY date_received ASC)
```

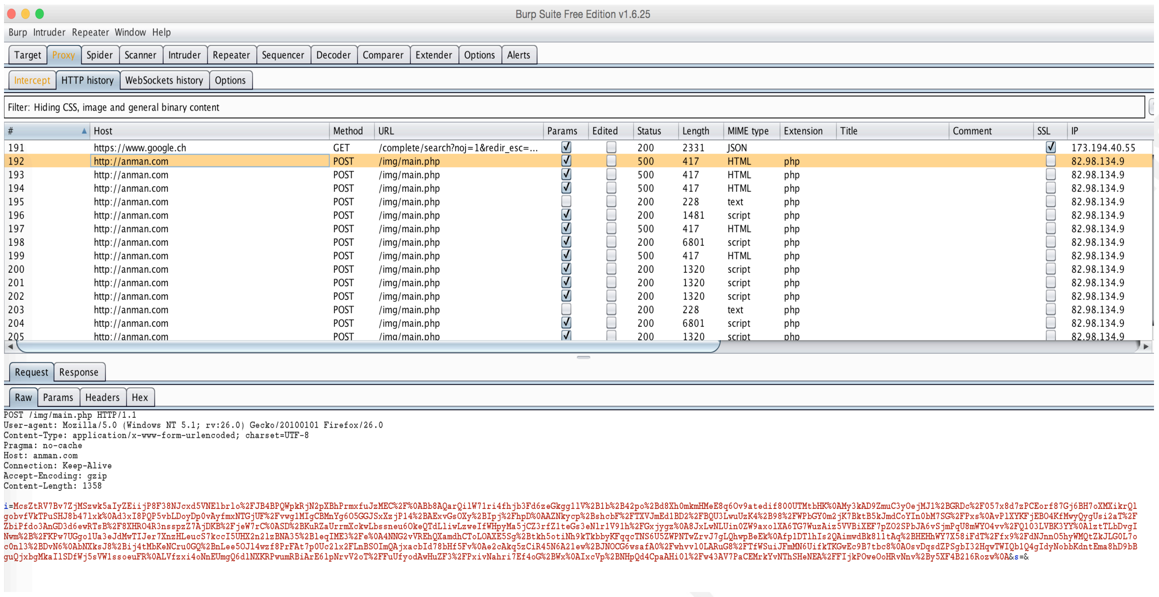
From this logs, something can be already detected:

- The name of the package installed: org.thoughtcrime.securesms. This application, securesms, is a real application created by the company WhisperSystems (https://github.com/WhisperSystems/TextSecure/tree/master/src/org/thoughtcrime/securesms)
- There are some references that do not appear to have standard names, for example org.thoughtcrime.securesms.**b.ab** or org.thoughtcrime.securesms.**cx.a**. This kind of names look like some obfuscation, similar to tools like Proguard (ProGuard. (n.d.). Retrieved September 15, 2015).
- There are some SQL queries accessing the SMS database as well.

The examples above indicate a few evidence gathered through the logs which are enough for the purpose of this research

3.1.3. Information gathered through network traffic

With Burp Suite and Wireshark it is possible to analyze in real time the traffic being sent and received. Burp suite focus on HTTP/HTTPs traffic (being able to decrypt HTTPs traffic), Wireshark captures the whole traffic. This setup guarantees that the whole traffic is being captured like for example HTTP traffic not going through standard ports or any non-HTTP traffic.



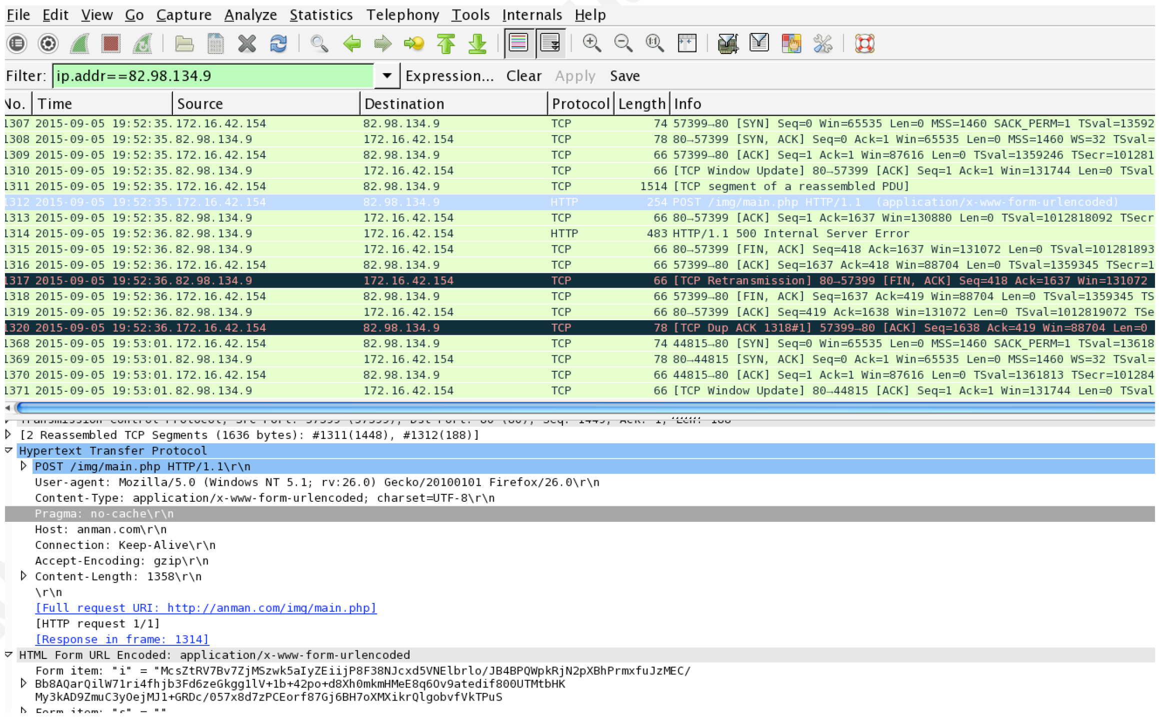
Burp Suite Free Edition v1.6.25

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	SSL	IP
191	https://www.google.ch	GET	/complete/search?noj=1&redir_esc=...	✓	☐	200	2331	JSON				✓	173.194.40.55
192	http://anman.com	POST	/img/main.php	✓	☐	500	417	HTML	php			☐	82.98.134.9
193	http://anman.com	POST	/img/main.php	✓	☐	500	417	HTML	php			☐	82.98.134.9
194	http://anman.com	POST	/img/main.php	✓	☐	500	417	HTML	php			☐	82.98.134.9
195	http://anman.com	POST	/img/main.php	☐	☐	200	228	text	php			☐	82.98.134.9
196	http://anman.com	POST	/img/main.php	✓	☐	200	1481	script	php			☐	82.98.134.9
197	http://anman.com	POST	/img/main.php	✓	☐	500	417	HTML	php			☐	82.98.134.9
198	http://anman.com	POST	/img/main.php	✓	☐	200	6801	script	php			☐	82.98.134.9
199	http://anman.com	POST	/img/main.php	✓	☐	500	417	HTML	php			☐	82.98.134.9
200	http://anman.com	POST	/img/main.php	✓	☐	200	1320	script	php			☐	82.98.134.9
201	http://anman.com	POST	/img/main.php	✓	☐	200	1320	script	php			☐	82.98.134.9
202	http://anman.com	POST	/img/main.php	✓	☐	200	1320	script	php			☐	82.98.134.9
203	http://anman.com	POST	/img/main.php	☐	☐	200	228	text	php			☐	82.98.134.9
204	http://anman.com	POST	/img/main.php	✓	☐	200	6801	script	php			☐	82.98.134.9
205	http://anman.com	POST	/img/main.php	✓	☐	200	1320	script	php			☐	82.98.134.9

```
POST /img/main.php HTTP/1.1
User-agent: Mozilla/5.0 (Windows NT 5.1; rv:26.0) Gecko/20100101 Firefox/26.0
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Pragma: no-cache
Host: anman.com
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Length: 1358

i=McsZtRV7Bv7ZjMSzwk5aIyZEiijP8F38NJcxd5VNElbrlo%2FJB4BPQWpkRjN2pXBhPrmxfuJzMEC%2F%0ABb8AQarQilW71ri4fhjb3Fd6zeGkgg1lV%2B1b%2B42po%2Bd8Xh0mkmHMeE8q6Ov9atedif800UTMtbHK%0AMy3kAD9ZmuC3yOejMJ1%2BGRDc%2F057x8d7zPCEorf87Gj6BH7oXMXikrQl
gobvfVkTPuSHJ8b47lxk%0Ad3xI8PQP5vbLDoyDp0vAyfmxNTGjUF%2Fvwg1MIgCBMnYg6O5GGJSxXzjP14%2BAExvGsOXy%2BIpj%2FhpD%0AAZNkycp%2BshcbF%2FTXVJmEd1BD2%2FBQU3LwuUzK4%2B98%2FWPbGY0m2jK7BktB5kJmdCoYIn0bM7SG%2FPxs%0AvPlXYKFjEBO4KfMwyQygUsi2aT%2F
ZbiPfdo3AnGD3d6ewRTsB%2F8XHRO4R3nsspzZ7AjDKB%2FjeW7rC%0ASD%2BKuRZaUrrmXckwLbssneu6OkeQTdLliwLzweIfWHpyMa5jCZ3rfZ1teGs3eNlr1V91h%2FGxjygz%0A8JxLwNLUin0ZW9axolXA6TG7WuzAiz5VVBiXEF7pZO2SPbJA6vSjmPqU8mWYO4vv%2FQ103LVBK3YY%0AlztTLbDvgI
Nwm%2B%2FKPw7UGgo1Ua3eJdMwTIJer7XnzHLeucS7kccI5UHX2n2lzBNA35%2BleqIME3%2Fe%0A4NNG2vVREhQXamdhCToLOAXE5Sg%2Btkh5otiNh9kTkbbyKFqqcTNS6U5ZWPNTwZrvJ7gLQhwpBeEk%0Afp1DTlhIs2QAimwdBk8l1tAq%2BHEHhWY7X58iFdT%2Ffx9%2FdNJnnO5hyWMQtZkJLG0L7o
c0nl3%2BDvN6%0AbNXksJ8%2Bij4tMbKeNCru0GQ%2BnLee5OJl4wzf8PrFAt7p0Uc2lx2FLnBSOImQAjxacbId78bHf5Fv%0Ae2cAkq5zCiR45N6A21ew%2BJNOCG6wsafA0%2Fwhvvl0LARuG8%2FTfWSuiJFmMN6UifkTKGwEc9B7tbc8%0AOsvDqsdZPSgbI32HqwTWIQb1Q4gIdyNobbKdntEma8hD9bB
guQjxbgMkaIlSDfWj5sVW1ssoeuFR%0ALVfzxi4oNnEUmgQ6dlNXKRPwumRBiArE6lpNrvV2oT%2FFuUfyodAwHuZF3%2FPxivNahri7Ef4oG%2BWx%0AIxcVp%2BNHpQd4CpaAHi0l%2Fw43AV7PaCEMrkYvNThSHeNEA%2FFIjkPOweOoHRvNnv%2By5XF4B216Rozw%0A&s=&
```

Figure 3: Traffic capture with Burp suite



Filter: ip.addr==82.98.134.9

```
No.  Time                    Source          Destination     Protocol Length Info
1307 2015-09-05 19:52:35.172.16.42.154  82.98.134.9     TCP        74 57399→80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1 TSval=13592
1308 2015-09-05 19:52:35.82.98.134.9    172.16.42.154   TCP        78 80→57399 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=32 TSval=
1309 2015-09-05 19:52:35.172.16.42.154  82.98.134.9     TCP        66 57399→80 [ACK] Seq=1 Ack=1 Win=87616 Len=0 TSval=1359246 TSecr=101281
1310 2015-09-05 19:52:35.82.98.134.9    172.16.42.154   TCP        66 [TCP Window Update] 80→57399 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TSval
1311 2015-09-05 19:52:35.172.16.42.154  82.98.134.9     TCP      1514 [TCP segment of a reassembled PDU]
1312 2015-09-05 19:52:35.172.16.42.154  82.98.134.9     HTTP      254 POST /img/main.php HTTP/1.1  (application/x-www-form-urlencoded)
1313 2015-09-05 19:52:35.82.98.134.9    172.16.42.154   TCP        66 80→57399 [ACK] Seq=1 Ack=1637 Win=130880 Len=0 TSval=1012818092 TSecr
1314 2015-09-05 19:52:36.82.98.134.9    172.16.42.154   HTTP      483 HTTP/1.1 500 Internal Server Error
1315 2015-09-05 19:52:36.82.98.134.9    172.16.42.154   TCP        66 80→57399 [FIN, ACK] Seq=418 Ack=1637 Win=131072 Len=0 TSval=101281893
1316 2015-09-05 19:52:36.172.16.42.154  82.98.134.9     TCP        66 57399→80 [ACK] Seq=1637 Ack=418 Win=88704 Len=0 TSval=1359345 TSecr=1
1317 2015-09-05 19:52:36.82.98.134.9    172.16.42.154   TCP        66 [TCP Retransmission] 80→57399 [FIN, ACK] Seq=418 Ack=1637 Win=131072
1318 2015-09-05 19:52:36.172.16.42.154  82.98.134.9     TCP        66 57399→80 [FIN, ACK] Seq=1637 Ack=419 Win=88704 Len=0 TSval=1359345 TS
1319 2015-09-05 19:52:36.82.98.134.9    172.16.42.154   TCP        66 80→57399 [ACK] Seq=419 Ack=1638 Win=131072 Len=0 TSval=1012819072 TSe
1320 2015-09-05 19:52:36.172.16.42.154  82.98.134.9     TCP        78 [TCP Dup ACK 1318#1] 57399→80 [ACK] Seq=1638 Ack=419 Win=88704 Len=0
1368 2015-09-05 19:53:01.172.16.42.154  82.98.134.9     TCP        74 44815→80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1 TSval=13618
1369 2015-09-05 19:53:01.82.98.134.9    172.16.42.154   TCP        78 80→44815 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=32 TSval=
1370 2015-09-05 19:53:01.172.16.42.154  82.98.134.9     TCP        66 44815→80 [ACK] Seq=1 Ack=1 Win=87616 Len=0 TSval=1361813 TSecr=101284
1371 2015-09-05 19:53:01.82.98.134.9    172.16.42.154   TCP        66 [TCP Window Update] 80→44815 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TSval
```

```
▷ [2 Reassembled TCP Segments (1636 bytes): #1311(1448), #1312(188)]
▽ Hypertext Transfer Protocol
  ▷ POST /img/main.php HTTP/1.1\r\n
    User-agent: Mozilla/5.0 (Windows NT 5.1; rv:26.0) Gecko/20100101 Firefox/26.0\r\n
    Content-Type: application/x-www-form-urlencoded; charset=UTF-8\r\n
    Pragma: no-cache\r\n
    Host: anman.com\r\n
    Connection: Keep-Alive\r\n
    Accept-Encoding: gzip\r\n
  ▷ Content-Length: 1358\r\n
    \r\n
    [Full request URI: http://anman.com/img/main.php]
    [HTTP request 1/1]
    [Response in frame: 1314]
▽ HTML Form URL Encoded: application/x-www-form-urlencoded
    Form item: "i" = "McsZtRV7Bv7ZjMSzwk5aIyZEiijP8F38NJcxd5VNElbrlo/JB4BPQWpkRjN2pXBhPrmxfuJzMEC/
  ▷ Bb8AQarQilW71ri4fhjb3Fd6zeGkgg1lV+1b+42po+d8Xh0mkmHMeE8q6Ov9atedif800UTMtbHK
    My3kAD9ZmuC3yOejMJ1+GRDc/057x8d7zPCEorf87Gj6BH7oXMXikrQlgobvfVkTPuS
  ▷ Form item: "s" = ""
```

Figure 4: traffic capture with Wireshark

For this case, the initial evidence gathered is:

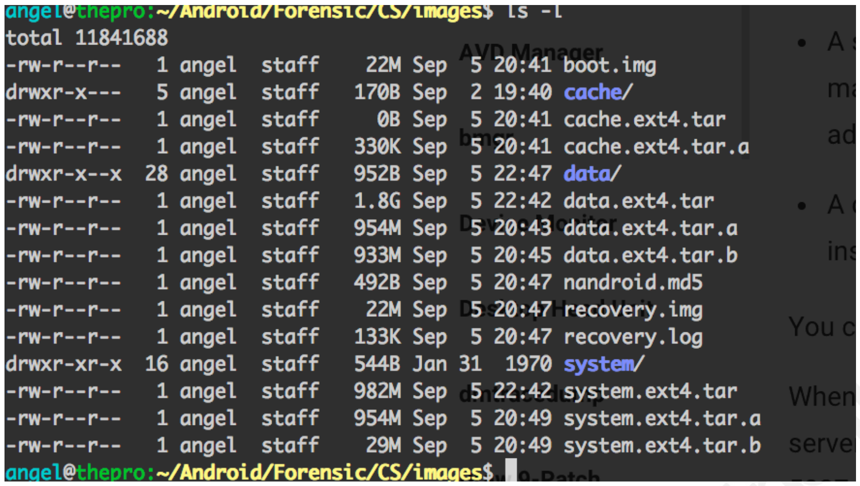
- URL accessed http://amann.com/img/main.php
- User-agent: Mozilla/5.0 (Windows NT 5.1; rv:26.0) Gecko/20100101 Firefox/26.0. The fake User-Agent is very suspicious for two reasons: it is a windows User-agent (instead of an Android) and the version of Firefox is quite old.
- Information sent through a HTTP POST request. The information sent in the paramenter "i" (Figure 3) is base64 encoded and encrypted.

3.1.4. Evidence on the file system

Next step is to do a full image of the file system by booting the device in recovery mode (Clockworkmod) and performing an image backup. This step generates a set of files, which needs to be pulled.

```
92eb1fa86e200e195aa43835d1a19189  boot.img
d41d8cd98f00b204e9800998ecf8427e  cache.ext4.tar
8da72bb5531c3ff0fa504aa312c03725  cache.ext4.tar.a
d41d8cd98f00b204e9800998ecf8427e  data.ext4.tar
b81641ca00b95697b7a1e4992249cf4a  data.ext4.tar.a
cdb4301242ebd71ebe5255c2e5bc4fab  data.ext4.tar.b
25e6a1d46cf5afb0e1f1e3495430c53e  recovery.img
d41d8cd98f00b204e9800998ecf8427e  system.ext4.tar
cf83aeff13d0c754428d08b08e5e1a76  system.ext4.tar.a
bdad0c5f92181d9cd51548a328f59320  system.ext4.tar.b
```

The command to pull the files is 'adb pull'. The files are split and need to be rebuilt, for example: 'cat data.ext4.tar.a data.ext4.tar.b > data.tar'. Following this task, the file is decompressed and the full directory structured is reconstructed. This same approach has to be done for /system.

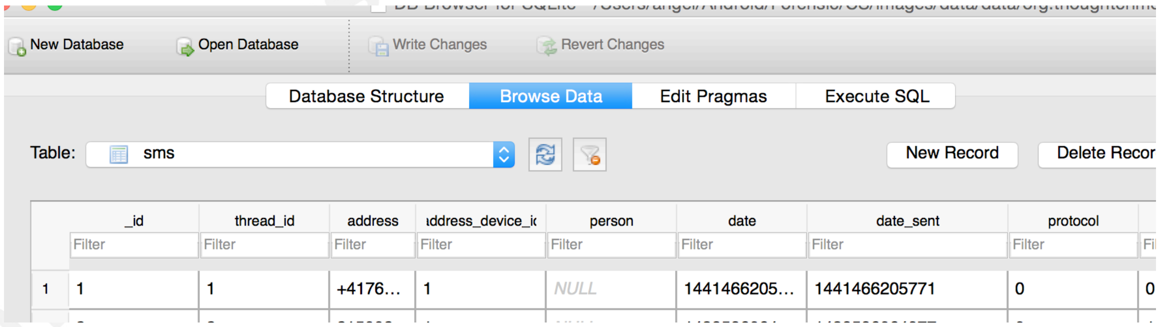


```
angel@thepro:~/Android/Forensic/CS/images$ ls -l
total 11841688
-rw-r--r--   1 angel  staff    22M Sep  5 20:41 boot.img
drwxr-x---   5 angel  staff   170B Sep  2 19:40 cache/
-rw-r--r--   1 angel  staff     0B Sep  5 20:41 cache.ext4.tar
-rw-r--r--   1 angel  staff   330K Sep  5 20:41 cache.ext4.tar.a
drwxr-x--x  28 angel  staff   952B Sep  5 22:47 data/
-rw-r--r--   1 angel  staff   1.8G Sep  5 22:42 data.ext4.tar
-rw-r--r--   1 angel  staff   954M Sep  5 20:43 data.ext4.tar.a
-rw-r--r--   1 angel  staff   933M Sep  5 20:45 data.ext4.tar.b
-rw-r--r--   1 angel  staff   492B Sep  5 20:47 nandroid.md5
-rw-r--r--   1 angel  staff    22M Sep  5 20:47 recovery.img
-rw-r--r--   1 angel  staff   133K Sep  5 20:47 recovery.log
drwxr-xr-x  16 angel  staff   544B Jan 31  1970 system/
-rw-r--r--   1 angel  staff   982M Sep  5 22:42 system.ext4.tar
-rw-r--r--   1 angel  staff   954M Sep  5 20:49 system.ext4.tar.a
-rw-r--r--   1 angel  staff    29M Sep  5 20:49 system.ext4.tar.b
angel@thepro:~/Android/Forensic/CS/images$
```

Figure 5: output of the file system

Now it is time to explore the full file system.

Although, there are many files that can be looked at, the analysis is focused on the relevant evidence for this research. In /data/data/org.thoughtcrime.securesms all the relevant data from the application is stored. For example, in the database subdirectory, it is found some SQLite data bases, which can be browsed with any tool like sqlitebrowser (DB Browser for SQLite). One of the SQLite files contains the data base of the SMS/MMS messages.



	_id	thread_id	address	ıddress_device_i	person	date	date_sent	protocol	Fi
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Fi
1	1	1	+4176…	1	NULL	1441466205…	1441466205771	0	0

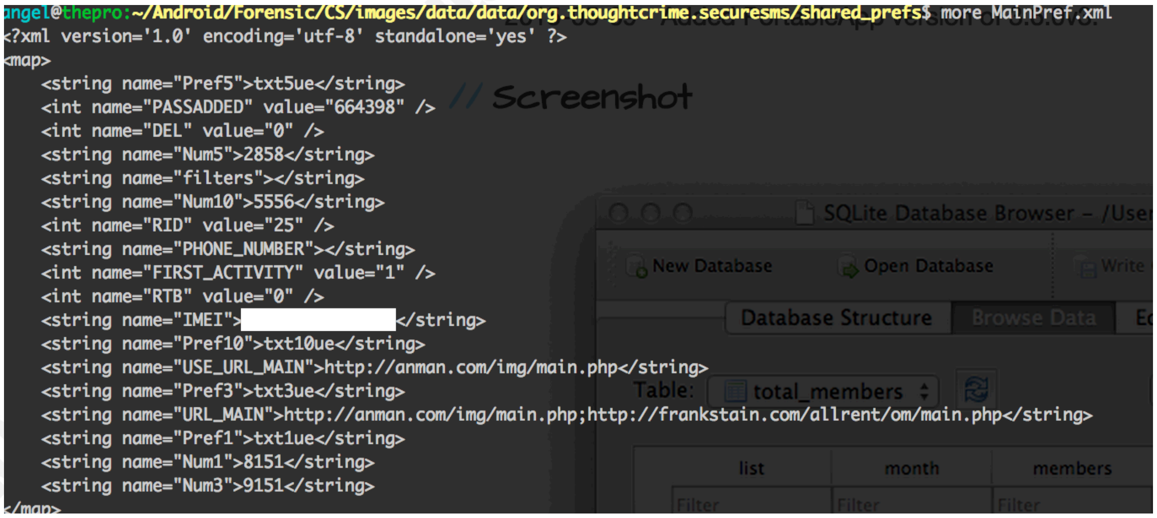
Figure 6: data extracted from the SQLite data base.

Basically, this can mean two things: the application is able to read the MMS/SMS databases from the default SMS application, or the new SQLite database has become the main SMS/MMS database. This evidence matches with the logs seen in point 3.1.3 (the SQL queries).

However going a step further, in the 'shared_prefs' folder, which normally is used by Android to keep configuration files for the application to work, exists some interesting XML files.

```
        angel@thepro:~/Android/Forensic/CS/images/data/data/org.thoughtcrime.secures
ms/shared_prefs$ ls -l

        total 32

        -rw-r-----  1 angel  staff   885B Sep  5 19:54 MainPref.xml

        -rw-r-----  1 angel  staff   951B Sep  5 19:52 SecureSMS-Preferences.xml

        -rw-r-----  1 angel  staff   165B Sep  5 19:52 SecureSMS.xml

-rw-r-----  1 angel  staff   117B Sep  5 19:52
org.thoughtcrime.securesms_preferences.xml
```

Looking across all of the files, the file MainPref.xml contains some valuable information:



```
angel@thepro:~/Android/Forensic/CS/images/data/data/org.thoughtcrime.securesms/shared_prefs$ more MainPref.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="Pref5">txt5ue</string>
    <int name="PASSADDED" value="664398" />
    <int name="DEL" value="0" />
    <string name="Num5">2858</string>
    <string name="filters"></string>
    <string name="Num10">5556</string>
    <int name="RID" value="25" />
    <string name="PHONE_NUMBER"></string>
    <int name="FIRST_ACTIVITY" value="1" />
    <int name="RTB" value="0" />
    <string name="IMEI">[redacted]</string>
    <string name="Pref10">txt10ue</string>
    <string name="USE_URL_MAIN">http://anman.com/img/main.php</string>
    <string name="Pref3">txt3ue</string>
    <string name="URL_MAIN">http://anman.com/img/main.php;http://frankstain.com/allrent/om/main.php</string>
    <string name="Pref1">txt1ue</string>
    <string name="Num1">8151</string>
    <string name="Num3">9151</string>
</map>
```

Figure 7: content of the MainPref.xml

Some fields to highlight:

- USE_URL_MAIN which contains the same URL we saw in Wireshark and Burp suite (http://frankstain.vom/allrent/om/main.php)

- URL_MAIN: contains some backup URL
- IMEI (which corresponds with the IMEI of the device).

In essence, this XML file looks like an initial configuration file used by the malware.

With this approach, looking into the filesystem, it would be also possible to obtain the original APK file installed in the device. However, in the case of this analysis we already had the APK file. (The file is stored in /data/app/org.thoughtcrime.securesms-1/base.apk)

Always it is possible to analyze further at file system level, but for the purpose of this research with evidence mentioned above is enough.

3.1.5. Memory analysis

The memory has been dumped through netcat as explained in the appendix. Now, volatility runs with the customize profile 'LinuxNexus5-511ARM', in order to list all the processes running:

```
python vol.py --profile=LinuxNexus5-511ARM  -f
~/Android2/CS_mem_image/lime2.dump  linux_psaux
```

As showed in the Appendix, the PID of the process to investigate is 17145. Worth to mention the existence of process 17686 (insmod lime.ko path=TCP:4444 format=lime), which it is the process for LiME.

Next step is to dump the details of all the memory allocated to process 17145.

```
        angel@ubuntu:~/Android2/volatility/volatility$ python vol.py --
profile=LinuxNexus5-511ARM  -f ~/Android2/CS_mem_image/lime2.dump  -p 17145
linux_proc_maps

        Offset          Pid     Name               Start            End              Flags     Pgoff
Major  Minor  Inode      File Path

        ------------------ -------- -------------------- ------------------ ------------------ ------ ------
---- ------ ------ ---------- ---------

        0x00000000ed175500    17145 crime.securesms      0x0000000012c00000
0x0000000012e01000 rw-          0x0      0      4       9397 /dev/ashmem/dalvik-main space
```

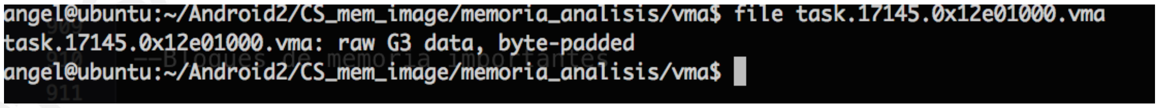
```
0x00000000ed175500    17145 crime.securesms      0x0000000012e01000
0x0000000013252000 rw-      0x201000      0      4       9397 /dev/ashmem/dalvik-main
space
…..
```

We can use the column "start" as a reference to dump the whole memory in different files. To achieve this is the content of column "start" is saves in a file named "pos_mem.txt".

```
        for i in `cat
/home/angel/Android2/CS_mem_image/memoria_analisis/pos_mem.txt`; do python
vol.py --profile=LinuxNexus5-511ARM  -f ~/Android2/CS_mem_image/lime2.dump
linux_dump_map -p 17145 -s $i --dump-dir
~/Android2/CS_mem_image/memoria_analisis/; done
```

Once the memory is dumped in different files, it is time to check which file contains the interesting information. In this case it is ‘‘task.17145.0x12e01000.vma’, which references to process:

"0x00000000ed175500   17145 crime.securesms   0x0000000012c00000 0x0000000012e01000 rw-   0x0   0   4   9397 /dev/ashmem/dalvik-main space"

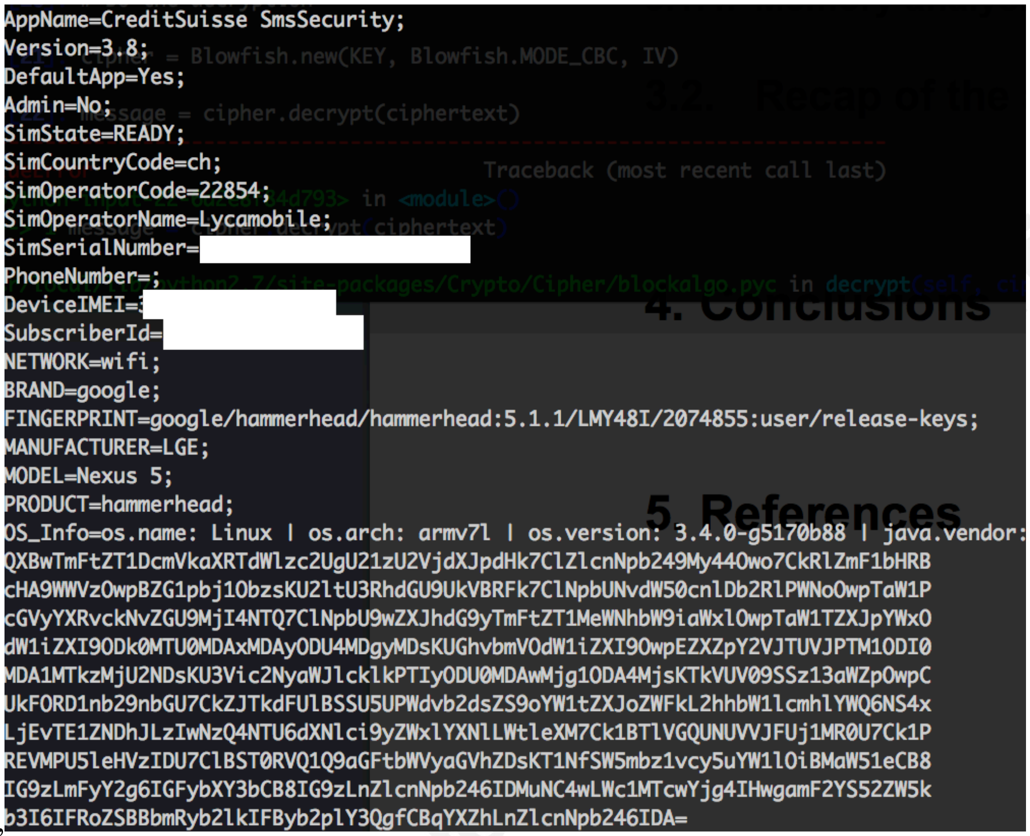


```
angel@ubuntu:~/Android2/CS_mem_image/memoria_analisis/vma$ file task.17145.0x12e01000.vma
task.17145.0x12e01000.vma: raw G3 data, byte-padded
angel@ubuntu:~/Android2/CS_mem_image/memoria_analisis/vma$
```

Figure 8: memory dump

Checking the file with 'strings', there is some information about the smartphone: IMEI, country, Provider, version of the device, kernel version, brand of the device, etc.

After that information, there is a string in base64 which starts with "QXB…==".



```
AppName=CreditSuisse SmsSecurity;
Version=3.8;
DefaultApp=Yes;
Admin=No;
SimState=READY;
SimCountryCode=ch;
SimOperatorCode=22854;
SimOperatorName=Lycamobile;
SimSerialNumber=[redacted]
PhoneNumber=;
DeviceIMEI=3[redacted]
SubscriberId=[redacted]
NETWORK=wifi;
BRAND=google;
FINGERPRINT=google/hammerhead/hammerhead:5.1.1/LMY48I/2074855:user/release-keys;
MANUFACTURER=LGE;
MODEL=Nexus 5;
PRODUCT=hammerhead;
OS_Info=os.name: Linux | os.arch: armv7l | os.version: 3.4.0-g5170b88 | java.vendor:
```

```
QXBwTmFtZT1DcmVkaXRTdWlzc2UgU21zU2VjdXJpdHk7ClZlcnNpb249My44Owo7CkRlZmF1bHRB
cHA9WWVzOwpBZG1pbj1ObzsKU2ltU3RhdGU9UkVBRFk7ClNpbUNvdW50cnlDb2RlPWNoOwpTaW1P
cGVyYXRvckNvZGU9MjI4NTQ7ClNpbU9wZXJhdG9yTmFtZT1MeWNhbW9iaWxlOwpTaW1TZXJpYWxO
dW1iZXI9ODk0MTU0MDAxMDAyODU4MDgyMDsKUGhvbmVOdW1iZXI9OwpEZXZpY2VJTUVJPTM1ODI0
MDA1MTkzMjU2NDsKU3Vic2NyaWJlcklkPTIyODU0MDAwMjg1ODA4MjsKTkVUV09SSz13aWZpOwpC
UkFORD1nb29nbGU7CkZJTkdFUlBSSU5UPWdvb2dsZS9oYW1tZXJoZWFkL2hhbW1lcmhlYWQ6NS4x
LjEvTE1ZNDhJLzIwNzQ4NTU6dXNlci9yZWxlYXNlLWtleXM7Ck1BTlVGQUNUVVJFUj1MR0U7Ck1P
REVMPU5leHVzIDU7ClBST0RVQ1Q9aGFtbWVyaGVhZDsKT1NfSW5mbz1vcy5uYW1lOiBMaW51eCB8
IG9zLmFyY2g6IGFybXY3bCB8IG9zLnZlcnNpb246IDMuNC4wLWc1MTcwYjg4IHwgamF2YS52ZW5k
b3I6IFRoZSBBbmRyb2lkIFByb2plY3QgfCBqYXZhLnZlcnNpb246IDA=
```

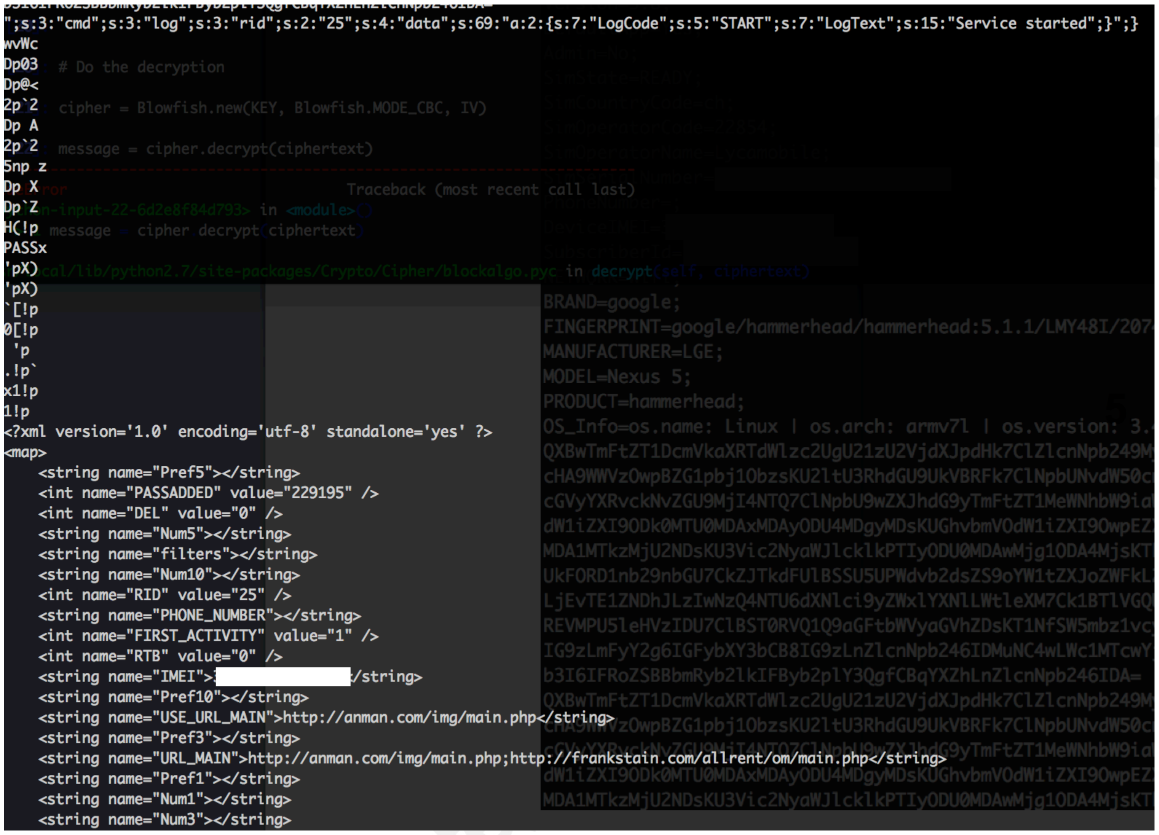
,

Figure 9: strings in memory with clear text information and base64 encoded

When executing: 'echo –n "QXB…DA==" | base64 –decode' the full information of the device above is showed. Basically, this means that the malware is dumping some information from the device and encoding it with base64, to likely send it through HTTP.

Angel Alonso- Parrizas, parrizas@gmail.com

Other interesting information are the following strings in memory:



```
";s:3:"cmd";s:3:"log";s:3:"rid";s:2:"25";s:4:"data";s:69:"a:2:{s:7:"LogCode";s:5:"START";s:7:"LogText";s:15:"Service started";}";}
wvWc
Dp03
Dp@<
2p`2
Dp A
2p`2
5np z
Dp X
Dp`Z
H(!p
PASSx
'pX)
'pX)
`[!p
0[!p
 'p
.!p`
x1!p
1!p
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="Pref5"></string>
    <int name="PASSADDED" value="229195" />
    <int name="DEL" value="0" />
    <string name="Num5"></string>
    <string name="filters"></string>
    <string name="Num10"></string>
    <int name="RID" value="25" />
    <string name="PHONE_NUMBER"></string>
    <int name="FIRST_ACTIVITY" value="1" />
    <int name="RTB" value="0" />
    <string name="IMEI">[redacted]</string>
    <string name="Pref10"></string>
    <string name="USE_URL_MAIN">http://anman.com/img/main.php</string>
    <string name="Pref3"></string>
    <string name="URL_MAIN">http://anman.com/img/main.php;http://frankstain.com/allrent/om/main.php</string>
    <string name="Pref1"></string>
    <string name="Num1"></string>
    <string name="Num3"></string>
```

Figure 10: C&C commands and initial config.

In the screenshot above, the first line, there is a set of commands which looks like C&C commands. Moreover, the exactly same initial configuration file analyzed through the file system (MainPref.xml) is also in memory.

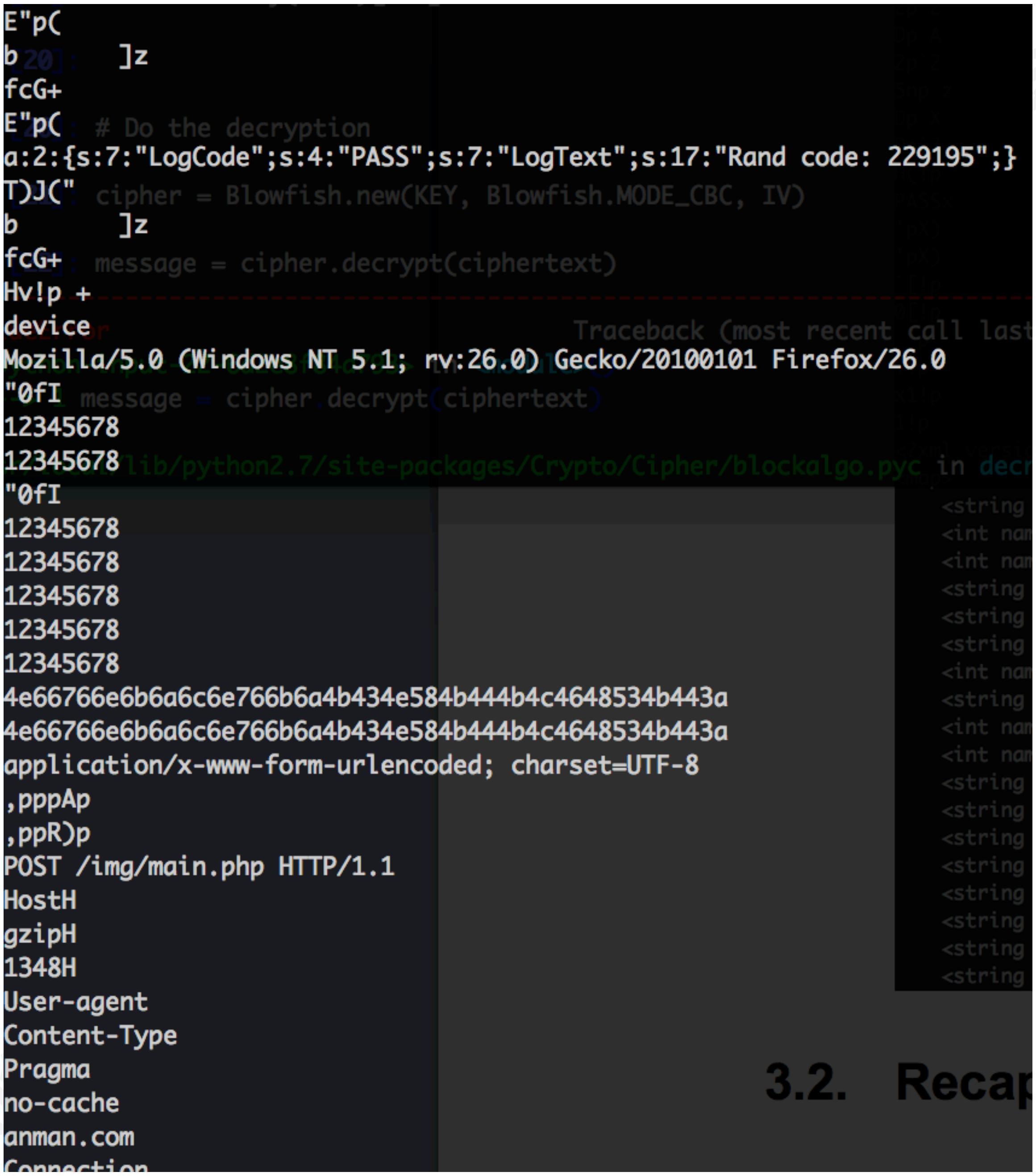
There are other existing C&C evidences:

- a:2:{s:7:"LogCode";s:4:"PASS";s:7:"LogText";s:16:"Rand code: 67302";}
- "a:4:{s:6:"device";s:750:"QXB…. ==" (this one will be analyzed later).

And other evidences which we already see:

- The Mozilla string which was detected in the network traffic
- The POST request with the base64 encoded information

However, what it is interesting and worth to investigate is the repeated string "4e66766e6b6a6c6e766b6a4b434e584b444b4c4648534b443a" and "12345678".



```
E"p(
b       ]z
fcG+
E"p(
a:2:{s:7:"LogCode";s:4:"PASS";s:7:"LogText";s:17:"Rand code: 229195";}
T)J("
b       ]z
fcG+
Hv!p +
device
Mozilla/5.0 (Windows NT 5.1; rv:26.0) Gecko/20100101 Firefox/26.0
"0fI
12345678
12345678
"0fI
12345678
12345678
12345678
12345678
12345678
4e66766e6b6a6c6e766b6a4b434e584b444b4c4648534b443a
4e66766e6b6a6c6e766b6a4b434e584b444b4c4648534b443a
application/x-www-form-urlencoded; charset=UTF-8
,pppAp
,ppR)p
POST /img/main.php HTTP/1.1
HostH
gzipH
1348H
User-agent
Content-Type
Pragma
no-cache
anman.com
```

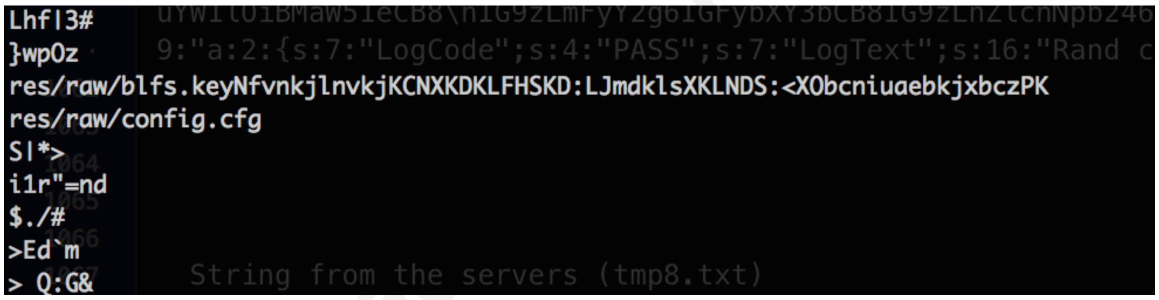
Figure 11: keys in memory

This is quite suspicious and it looks like some kind of crypto key used to encrypt/decrypt. Searching in Google for the first string (key) we can find two interesting articles from 2014.

Angel Alonso- Parrizas, parrizas@gmail.com

- http://blog.dornea.nu/2014/07/07/disect-android-apks-like-a-pro-static-code-analysis/
- http://maldr0id.blogspot.ch/2014/09/android-malware-based-on-sms-encryption.html

In both articles a Banking malware is investigated and same crypto key (4e66766e6b6a6c6e766b6a4b434e584b444b4c4648534b443a) and an Initialization Vector (12345678) is analyzed. Clearly the APK file is some malware which it is from the same family. In both articles some there is some interesting information: a file named blfs.key, where the encoded encrypted key is stored and config.cfg with keeps the initial configuration (the content is encrypted). Both files can be found references in the memory:



```
Lhf|3#
}wp0z
res/raw/blfs.keyNfvnkjlnvkjKCNXKDKLFHSKD:LJmdklsXKLNDS:<XObcniuaebkjxbczPK
res/raw/config.cfg
S|*>
i1r"=nd
$./#
>Ed`m
> Q:G&
```

Figure 12: crypto key file and initial configuration file

This means the malware is using the same exactly configuration for the encryption. Using the information from both blogs, the next step is to decrypt the data interchanged between the C&C and the compromised device. Obviously, that information should be in memory decrypted as well.

To check it, the information sent by the initial POST is going to be decrypted. As showed in the appendix, the string sent in the POST is decrypted as some C&C commands “a:4:{s:6:"device";s:750…” following with the string encoded with base64 and beginning with “QXBw..”. This is basically what was found in the memory matching the information from the device (in clear text).

With this approach it is possible to analyze further communication with the C&C, however this is out of the scope of this research.

3.1.6. A bit about the malware

It is out of the scope to perform the analysis of the malware. But is it worth to give some additional information about this malware. This specific APK is just one new version of the malware discovered in the campaign emmental (Cybercriminals to Online Banks: Check -. (2014, July 22)). The main target of this malware is to steal the 2FA tokens sent through SMS by some banks. The malware is using obfuscation in the code and using some anti-analysis techniques (like detecting virtual devices, if the device has a valid GSM network, or has a valid phone number). In order to make the malware stealthier it uses a well know SMS encryption application as a baseline, and on top of that the additional functional malware code is added and obfuscated. The malware is able to read the SMS received in order to forward them. Actually, one of the tests performed during this analysis was to send a text message to the phone and see how consequently a HTTP POST was set to the C&C. The original APK file contains obfuscated code, but the author found some previous versions of the malware which was not using this obfuscation techniques, but other techniques to make difficult the analysis with some standard tools. This specific version the malware was targeting a Swiss bank, however the previous version checked and reported in VirusTotal https://www.virustotal.com/es/file/06d6e5ac153ab5970385e998164503b9abfaa99f89730ee98618290785fd925d/analysis/ was hitting some Austrian banks.

The key point here is that even the malware has being updated to target other banks, the core malware code is using exactly the same encryption techniques, cryptographic keys and Initialization Vectors, which makes much easier the analysis and to spot it.

4. Conclusions

During this paper the author presented several techniques to perform forensic Analysis in Android. Each techniques focus on a different layer, and the evidences gathered can be easily cross-correlate between them.

It was presented how it is possible to analyze the memory of a live device and gather all the evidences in clear text before they are encrypted. This can be really useful is some scenarios where we the code is obfuscated and we need to see what information is being sent and received.

5. References

504ensicsLabs/LiME. (n.d.). Retrieved September 15, 2015, from https://github.com/504ensicslabs/lime

Alonso Parrizas, A. (2011, September 22). Securely deploying Android devices. Retrieved from http://www.sans.org/reading_room/whitepapers/sysadmin/securelydeploying-android-devices_33799

Android Debug Bridge. (n.d.). Retrieved September 15, 2015, from http://developer.android.com/tools/help/adb.html

Bommisetty, S., & Tamma, R. (2014). Practical mobile forensics dive into mobile forensics on iOS, Android, Windows, and BlackBerry devices with this action-packed, practical guide. Birmingham, UK: Packt Pub.

Burp Suite. (n.d.). Retrieved September 15, 2015, from https://portswigger.net/burp/

ClockworkMod. (n.d.). Retrieved September 15, 2015, from https://www.clockworkmod.com/

Cybercriminals to Online Banks: Check -. (2014, July 22). Retrieved September 15, 2015, from http://blog.trendmicro.com/finding-holes-operation-emmental/

DB Browser for SQLite. (n.d.). Retrieved September 15, 2015, from http://sqlitebrowser.org/

Homebrew. (n.d.). Retrieved September 15, 2015, from http://brew.sh
How To: Configure a WiFi Pineapple For Use With Mac OS X. (n.d.). Retrieved September 15, 2015, from https://www.youtube.com/watch?v=m7XUmfC8ESw

Installing the Android SDK. (n.d.). Retrieved September 15, 2015, from https://developer.android.com/sdk/installing/index.html?pkg=tools

Investigating Your RAM Usage. (n.d.). Retrieved September 15, 2015, from https://developer.android.com/tools/debugging/debugging-memory.html

Logcat. (n.d.). Retrieved September 15, 2015, from http://developer.android.com/tools/help/logcat.html

ProGuard. (n.d.). Retrieved September 15, 2015, from
http://developer.android.com/tools/help/proguard.html

Volatility Foundation. (n.d.). Retrieved September 15, 2015, from
https://github.com/volatilityfoundation

WiFi Pineapple. (n.d.). Retrieved September 15, 2015, from
https://www.wifipineapple.com/

Wireshark. (n.d.). Retrieved September 15, 2015,
from https://www.wireshark.org/

6. Appendix

6.1. Appendix A: Compiling Android Kernel with modprobe

Reference link:

http://kuester.multics.org/blog/2015/03/24/how-to-build-custom-kernel-with-kprobes/

```
#Getting the right kernel sources
$git clone https://android.googlesource.com/kernel/msm.git

#check the kernel version
$ adb shell cat /proc/version | grep -e "[0-9.]*-g[a-z0-9]*" –o

#Checkout the correct commit from that kernel version
$cd msm; git checkout bebb36b
```

```
# Create kernel configuration with kprobes to be able to load modules into kernel
# create the .config

$cd msm; ARCH=arm make hammerhead_defconfig

# Append the following lines to the .config in order to support modules:

CONFIG_KPROBES=y CONFIG_KPROBES_SANITY_TEST=y #
CONFIG_KPROBE_EVENT is not set # CONFIG_ARM_KPROBES_TEST is not set
CONFIG_NET_TCPPROBE=y # seems necessary otherwise kernel does not boot
CONFIG_MODULES=y # kprobes requires module support and makes no sense without
CONFIG_MODULE_FORCE_LOAD=y CONFIG_MODULE_UNLOAD=y
CONFIG_MODULE_FORCE_UNLOAD=y CONFIG_MODVERSIONS=y # possible
to use modules compiled for different kernels #
```

```
CONFIG_MODULE_SRCVERSION_ALL is not set
```

```
# compile the kernel phase
# obtained the toolchains for cross-compile in Android >5.0  (arm-eabi-)

$git clone https://android.googlesource.com/platform/prebuilts/gcc/linux-x86/arm/arm-eabi-4.7

# cross compile the kernel
$cd msm; ARCH=arm CROSS_COMPILE=../arm-eabi-4.7/bin/arm-eabi- make
# if any question is asked for the new options of the kernel just choose the default
```

```
# Preparing the boot image (kernel + ramdisk) from the existing one in your phone

# obtaining the existing one

$adb shell su -c "ls -al /dev/block/platform/msm_sdcc.1/by-name/boot" lrwxrwxrwx root
root             1970-09-29 07:33 boot -> /dev/block/mmcblk0p19

$ adb shell su -c "dd if=/dev/block/mmcblk0p19 of=/sdcard/my_nexus5_boot.img"
$ adb pull /sdcard/my_nexus5_boot.img
```

```
# Unpack original image and repack with our customize kernel

$adb shell su -c "ls -al /dev/block/platform/msm_sdcc.1/by-name/boot" lrwxrwxrwx root
root             1970-09-29 07:33 boot -> /dev/block/mmcblk0p19
$ adb shell su -c "dd if=/dev/block/mmcblk0p19 of=/sdcard/my_nexus5_boot.img"
$ adb pull /sdcard/my_nexus5_boot.img

# it is necessary the tools unmkbootimg (unpacking) and mkbootimg (packing)
$ git clone https://github.com/pbatard/bootimg-tools.git $ cd bootimg-tools; make

# Unpack the boot image
$./bootimg-tools/mkbootimg/unmkbootimg -i my_nexus5_boot.img

# rebuild the  boot image
$mkbootimg --base 0 --pagesize 2048 --kernel_offset 0x00008000 --ramdisk_offset
0x02900000 --second_offset 0x00f00000 --tags_offset 0x02700000 --cmdline
'console=ttyHSL0,115200,n8 androidboot.hardware=hammerhead user_debug=31
maxcpus=2 msm_watchdog_v2.enable=1' --kernel kernel --ramdisk ramdisk.cpio.gz -o
my_nexus5_boot.img
```

Angel Alonso- Parrizas, parrizas@gmail.com

```
# if everything went fine you should have the files “kernel” and ramdisk.cpio.gz
# Now it is is necessary to repack everything

$booting-img/bootimg-tools/mkbootimg/mkbootimg --base 0 --pagesize 2048 --
kernel_offset 0x00008000  --ramdisk_offset 0x02900000 --second_offset 0x00f00000 --
tags_offset 0x02700000 --cmdline 'console=ttyHSL0,115200,n8
androidboot.hardware=hammerhead user_debug=31   maxcpus=2
msm_watchdog_v2.enable=1' --kernel ./msm/arch/arm/boot/zImage-dtb   --ramdisk
ramdisk.cpio.gz -o my_nexus5_kprobes_boot.img
```

```
# example of the final files

angel@ubuntu:~/Android2$ ls

arm-eabi-4.7  booting-img  kernel  lime  msm  my_nexus5_boot.img

my_nexus5_kprobes_boot.img  ramdisk.cpio.gz
```

```
# Finally boot the device with the new boot command
# This is done from the MacOSX system

$ adb reboot bootloader
$ sudo fastboot boot my_nexus5_kprobes_boot.img
```

6.2. Appendix B: Compiling LiME

Reference link:
https://code.google.com/p/volatility/wiki/AndroidMemoryForensics

```
# Get the source code
$ git clone https://github.com/504ensicsLabs/LiME.git

# Backing up the Makefile and editing it

$ cd /home/angel/Android2/lime/LiME/src
$ cp Makefile Makefile.bkp
$ vim.tiny Makefile
```

```
# The content of the Makefile needs to reference to the Android kernel ‘msm’ directory
# Also it needs  the cross-compiler arm-eabi-4.7 installed when compiling Android
# kernel. An example of Make file is as follow

bj-m := lime.o
lime-objs := tcp.o disk.o main.o

# KDIR where the kernel source code is
KDIR := ~/Android2/kernel/msm

KVER := $(shell uname -r)

PWD := $(shell pwd)

default:
	$(MAKE) ARCH=arm CROSS_COMPILE=~/Android2/arm-eabi-4.7/bin/arm-eabi-
-C $(KDIR) M=$(PWD) modules
	strip --strip-unneeded lime.ko
	mv lime.ko lime-$(KVER).ko

# Now it is time to compile with Make. Although there might be some errors, if the in the
‘lime.ko’ module is presented, the compilation is success

# example of compiling logs
```

```
angel@ubuntu:~/Android2/lime/LiME/src$ make
make ARCH=arm CROSS_COMPILE=~/Android2/arm-eabi-4.7/bin/arm-eabi- -C
~/Android2/msm M=/home/angel/Android2/lime/LiME/src modules
make[1]: Entering directory `/home/angel/Android2/msm'
  CC [M]  /home/angel/Android2/lime/LiME/src/tcp.o
  CC [M]  /home/angel/Android2/lime/LiME/src/disk.o
  CC [M]  /home/angel/Android2/lime/LiME/src/main.o
  LD [M]  /home/angel/Android2/lime/LiME/src/lime.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/angel/Android2/lime/LiME/src/lime.mod.o
  LD [M]  /home/angel/Android2/lime/LiME/src/lime.ko
make[1]: Leaving directory `/home/angel/Android2/msm'
```

```
strip --strip-unneeded lime.ko
strip: Unable to recognise the format of the input file `lime.ko'
make: *** [default] Error 1

# checking the module is ‘lime.ko’ exists

angel@ubuntu:~/Android2/lime/LiME/src$ ls -l
total 1484
-rw-rw-r-- 1 angel angel    290 Sep  4 08:20 Makefile
-rw-rw-r-- 1 angel angel   1646 Sep  4 08:15 Makefile.bkp
-rw-rw-r-- 1 angel angel   1723 Sep  4 08:14 Makefile.sample
-rw-rw-r-- 1 angel angel      0 Sep  4 08:21 Module.symvers
-rw-rw-r-- 1 angel angel   2379 Sep  4 08:14 disk.c
-rw-rw-r-- 1 angel angel 158756 Sep  4 08:21 disk.o
-rw-rw-r-- 1 angel angel   1821 Sep  4 08:14 lime.h
-rw-rw-r-- 1 angel angel 491708 Sep  4 08:21 lime.ko
-rw-rw-r-- 1 angel angel   1203 Sep  4 08:21 lime.mod.c
-rw-rw-r-- 1 angel angel  18380 Sep  4 08:21 lime.mod.o
-rw-rw-r-- 1 angel angel 474393 Sep  4 08:21 lime.o
-rw-rw-r-- 1 angel angel   5303 Sep  4 08:14 main.c
-rw-rw-r-- 1 angel angel 162856 Sep  4 08:21 main.o
-rw-rw-r-- 1 angel angel     50 Sep  4 08:21 modules.order
-rw-rw-r-- 1 angel angel   3543 Sep  4 08:14 tcp.c
-rw-rw-r-- 1 angel angel 160484 Sep  4 08:21 tcp.o
```

```
# Now it is time to push the module to the device and install it
$ adb push lime.ko /sdcard/lime.ko
$ adb shell 'ls -l /sdcard/lime.ko'
-rw-rw---- root     sdcard_r   491708 2015-09-04 17:52 lime.ko
```

```
# Two different ways of pushing the module and dumping the memory

# option 1: dumping the memory to the sdcard –

$ insmod /sdcard/lime.ko "path=/sdcard/lime.dump format=lime”

 # option 2: dumping the memory through ‘netcat ‘ into other host

# preparing port forwarding through ADB
$ adb forward tcp:4444 tcp:4444
$ adb shell
$ insmod /sdcar/lime.ko "path=tcp:4444 format=lime"

# In the destination host it is necessary to run netcat
nc localhost 4444 > lime2.dump
```

6.3. Appendix C: Creating a Volatility profile

Reference link:
https://code.google.com/p/volatility/wiki/AndroidMemoryForensics

```
#Install dwarfdump
$ apt-get install dwarfdump

# download volatility source

$git clone https://github.com/volatilityfoundation/volatility.git
$ cd volatility/volatility/tools/linux

# Edit Makefile to reference to the ARM cross compiler 'arm-eabi-4.7/bin'.

$cd volatility/volatility/tools/linux

# An example of Makefile:
obj-m += module.o
KDIR := ~/Android2/msm
KVER ?= $(shell uname -r)
CCPATH := ~/Android2/arm-eabi-4.7/bin
DWARFDUMP := /usr/bin/dwarfdump
-include version.mk
all: dwarf
dwarf: module.c
	$(MAKE) ARCH=arm CROSS_COMPILE=$(CCPATH)/arm-eabi- -C $(KDIR)
CONFIG_DEBUG_INFO=y M=$(PWD) modules
	$(DWARFDUMP) -di module.ko > module.dwarf

# compile with make.
$ make
# example of output:

make ARCH=arm CROSS_COMPILE=~/Android2/arm-eabi-4.7/bin/arm-eabi- -C
~/Android2/msm CONFIG_DEBUG_INFO=y
M=/home/angel/Android2/volatility/volatility/tools/linux modules
make[1]: Entering directory `/home/angel/Android2/msm'
  CC [M]  /home/angel/Android2/volatility/volatility/tools/linux/module.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/angel/Android2/volatility/volatility/tools/linux/module.mod.o
  LD [M]  /home/angel/Android2/volatility/volatility/tools/linux/module.ko
make[1]: Leaving directory `/home/angel/Android2/msm'
/usr/bin/dwarfdump -di module.ko > module.dwarf
```

```
# Check that the header is similar to this:

$ head module.dwarf

.debug_info

<0><0x0+0xb><DW_TAG_compile_unit> DW_AT_producer<"GNU C 4.7">
DW_AT_language<DW_LANG_C89>
DW_AT_name<"/home/angel/Android2/volatility/volatility/tools/linux/module.c">
DW_AT_comp_dir<"/home/angel/Android2/msm">
DW_AT_stmt_list<0x00000000>
<1><0x1d><DW_TAG_typedef> DW_AT_name<"__s8">
DW_AT_decl_file<0x00000001 include/asm-generic/int-ll64.h>
DW_AT_decl_line<0x00000013> DW_AT_type<<0x00000028>>
<1><0x28><DW_TAG_base_type> DW_AT_byte_size<0x00000001>
DW_AT_encoding<DW_ATE_signed_char> DW_AT_name<"signed char">
<1><0x2f><DW_TAG_typedef> DW_AT_name<"__u8">
DW_AT_decl_file<0x00000001 include/asm-generic/int-ll64.h>
DW_AT_decl_line<0x00000014> DW_AT_type<<0x0000003a>>
<1><0x3a><DW_TAG_base_type> DW_AT_byte_size<0x00000001>
DW_AT_encoding<DW_ATE_unsigned_char> DW_AT_name<"unsigned char">
<1><0x41><DW_TAG_typedef> DW_AT_name<"__s16">
DW_AT_decl_file<0x00000001 include/asm-generic/int-ll64.h>
DW_AT_decl_line<0x00000016> DW_AT_type<<0x0000004c>>
<1><0x4c><DW_TAG_base_type> DW_AT_byte_size<0x00000002>
DW_AT_encoding<DW_ATE_signed> DW_AT_name<"short int">
```

```
# Now combine module.dwarf and the System.map from your android kernel source code
# into a zip file

$zip ~/Android2/volatility/volatility/volatility/plugins/overlays/linux/Nexus5-511.zip
module.dwarf ~/Android2/msm/System.map
```

```
# Check the new profile exists (in this case is the first one)
```

```
$~/Android2/volatility/volatility$ python vol.py  --info | grep Linux
Volatility Foundation Volatility Framework 2.4
LinuxNexus5-511ARM - A Profile for Linux Nexus5-511 ARM
linux_banner              - Prints the Linux banner information
linux_yarascan             - A shell in the Linux memory image
```

6.4. Appendix D: Processes gathered with Volatility

```
        angel@ubuntu:~/Android2/volatility/volatility$ python vol.py --
profile=LinuxNexus5-511ARM  -f ~/Android2/CS_mem_image/lime2.dump
linux_psaux

        Volatility Foundation Volatility Framework 2.4

        *** Failed to import volatility.plugins.malware.apihooks (NameError: name
'distorm3' is not defined)

        *** Failed to import volatility.plugins.ssdt (NameError: name 'distorm3' is not
defined)

        *** Failed to import volatility.plugins.mac.apihooks (ImportError: No module
named distorm3)

        *** Failed to import volatility.plugins.malware.threads (NameError: name
'distorm3' is not defined)

        *** Failed to import volatility.plugins.mac.apihooks_kernel (ImportError: No
module named distorm3)

        *** Failed to import volatility.plugins.mac.check_syscall_shadow (ImportError:
No module named distorm3)

        ^[[HPid    Uid    Gid    Arguments

        1      0      0      /init

        2      0      0      [kthreadd]

        3      0      0      [ksoftirqd/0]
```

```
7     0     0     [kworker/u:0H]

8     0     0     [migration/0]

13    0     0     [khelper]

14    0     0     [netns]

18    0     0     [modem_notifier]

19    0     0     [smd_channel_clo]

20    0     0     [smsm_cb_wq]

22    0     0     [rpm-smd]

23    0     0     [kworker/u:1H]

24    0     0     [irq/317-earjack]

37    0     0     [sync_supers]

38    0     0     [bdi-default]

39    0     0     [kblockd]

40    0     0     [vmalloc]

41    0     0     [khubd]

42    0     0     [irq/102-msm_iom]

43    0     0     [irq/102-msm_iom]

44    0     0     [irq/102-msm_iom]

45    0     0     [irq/79-msm_iomm]

46    0     0     [irq/78-msm_iomm]

47    0     0     [irq/78-msm_iomm]

48    0     0     [irq/74-msm_iomm]

49    0     0     [irq/75-msm_iomm]

50    0     0     [irq/75-msm_iomm]
```

Angel Alonso- Parrizas, parrizas@gmail.com

```
51    0    0    [irq/75-msm_iomm]
52    0    0    [irq/75-msm_iomm]
53    0    0    [irq/273-msm_iom]
54    0    0    [irq/273-msm_iom]
55    0    0    [irq/97-msm_iomm]
56    0    0    [irq/97-msm_iomm]
57    0    0    [irq/97-msm_iomm]
58    0    0    [l2cap]
59    0    0    [a2mp]
60    0    0    [cfg80211]
62    0    0    [qmi]
63    0    0    [nmea]
64    0    0    [msm_ipc_router]
65    0    0    [apr_driver]
67    0    0    [kswapd0]
68    0    0    [fsnotify_mark]
69    0    0    [cifsiod]
70    0    0    [crypto]
88    0    0    [ad_calc_wq]
89    0    0    [hdmi_tx_workq]
90    0    0    [anx7808_work]
91    0    0    [k_hsuart]
92    0    0    [diag_wq]
93    0    0    [diag_cntl_wq]
```

```
94     0      0     [diag_dci_wq]
95     0      0     [kgsl-3d0]
97     0      0     [f9966000.spi]
108    0      0     [usbnet]
109    0      0     [irq/329-anx7808]
110    0      0     [k_rmnet_mux_wor]
111    0      0     [f_mtp]
112    0      0     [file-storage]
113    0      0     [uether]
114    0      0     [synaptics_wq]
115    0      0     [irq/362-s3350]
117    0      0     [msm_vidc_worker]
118    0      0     [msm_vidc_worker]
119    0      0     [msm_cpp_workque]
120    0      0     [irq/350-bq51013]
122    0      0     [dm_bufio_cache]
123    0      0     [dbs_sync/0]
124    0      0     [dbs_sync/1]
125    0      0     [dbs_sync/2]
126    0      0     [dbs_sync/3]
127    0      0     [cfinteractive]
128    0      0     [irq/170-msm_sdc]
129    0      0     [binder]
130    0      0     [usb_bam_wq]
```

```
131   0   0   [krfcommd]
132   0   0   [bam_dmux_rx]
133   0   0   [bam_dmux_tx]
134   0   0   [rq_stats]
135   0   0   [deferwq]
136   0   0   [irq/361-MAX1704]
138   0   0   [mmcqd/1]
139   0   0   [mmcqd/1rpmb]
140   0   0   [wl_event_handle]
141   0   0   [dhd_watchdog_th]
142   0   0   [dhd_dpc]
143   0   0   [dhd_rxf]
144   0   0   [dhd_sysioc]
145   0   0   [vibrator]
146   0   0   [max1462x]
147   0   0   [irq/310-maxim_m]
148   0   0   [irq/311-maxim_m]
149   0   0   /sbin/ueventd
151   0   0   [jbd2/mmcblk0p25]
152   0   0   [ext4-dio-unwrit]
155   0   0   [flush-179:0]
157   0   0   [jbd2/mmcblk0p28]
158   0   0   [ext4-dio-unwrit]
162   0   0   [jbd2/mmcblk0p27]
```

```
163   0     0     [ext4-dio-unwrit]
164   0     0     [jbd2/mmcblk0p16]
165   0     0     [ext4-dio-unwrit]
188   1036  1036  /system/bin/logd
189   0     0     /sbin/healthd
190   0     0     /system/bin/lmkd
191   1000  1000  /system/bin/servicemanager
194   0     0     /system/bin/vold
195   0     0     [IPCRTR]
196   1000  1003  /system/bin/surfaceflinger
197   9999  3004  /system/bin/rmt_storage
198   0     0     [sb-1]
199   0     0     [ipc_rtr_q6_ipcr]
200   1000  1000  /system/bin/qseecomd
202   0     0     [ngd_msm_ctrl_ng]
203   0     0     /system/bin/netd
204   0     0     /system/bin/debuggerd
205   1001  1001  /system/bin/rild
206   1019  1019  /system/bin/drmserver
207   1013  1005  /system/bin/mediaserver
208   1012  1012  /system/bin/installd
210   1017  1017  /system/bin/keystore /data/misc/keystore
212   1001  1001  /system/bin/bridgemgrd
213   1001  1001  /system/bin/qmuxd
```

```
        214    1001   1000   /system/bin/netmgrd

        215    9999   3004   /system/bin/sensors.qcom

        218    0      1001   /system/bin/thermal-engine-hh

        221    0      0      [msm_slim_qmi_cl]

        222    0      0      [msm_qmi_rtx_q]

        225    0      0      [irq/288-wcd9xxx]

        230    0      0      zygote

        235    0      0      [kauditd]

        241    1000   1000   /system/bin/qseecomd

        242    1023   1023   /system/bin/sdcard -u 1023 -g 1023 -l /data/media
/mnt/shell/emulated

        243    1006   1006   /system/bin/mm-qcamera-daemon

        244    1000   3004   /system/bin/time_daemon

        261    2000   2000   /sbin/adbd --root_seclabel=u:r:su:s0

        306    0      0      [msm_thermal:hot]

        307    0      0      [msm_thermal:fre]

        346    0      0      [mdss_fb0]

        511    0      0      daemonsu:mount:master

        541    0      0      [IPCRTR]

        543    0      0      [ipc_rtr_smd_ipc]

        574    0      0      daemonsu:master

        772    1000   1000   system_server

        912    1010   1010   /system/bin/wpa_supplicant -iwlan0 -Dnl80211 -
c/data/misc/wifi/wpa_supplicant.conf -I/system/etc/wifi/wpa_supplicant_overlay.conf -N
```

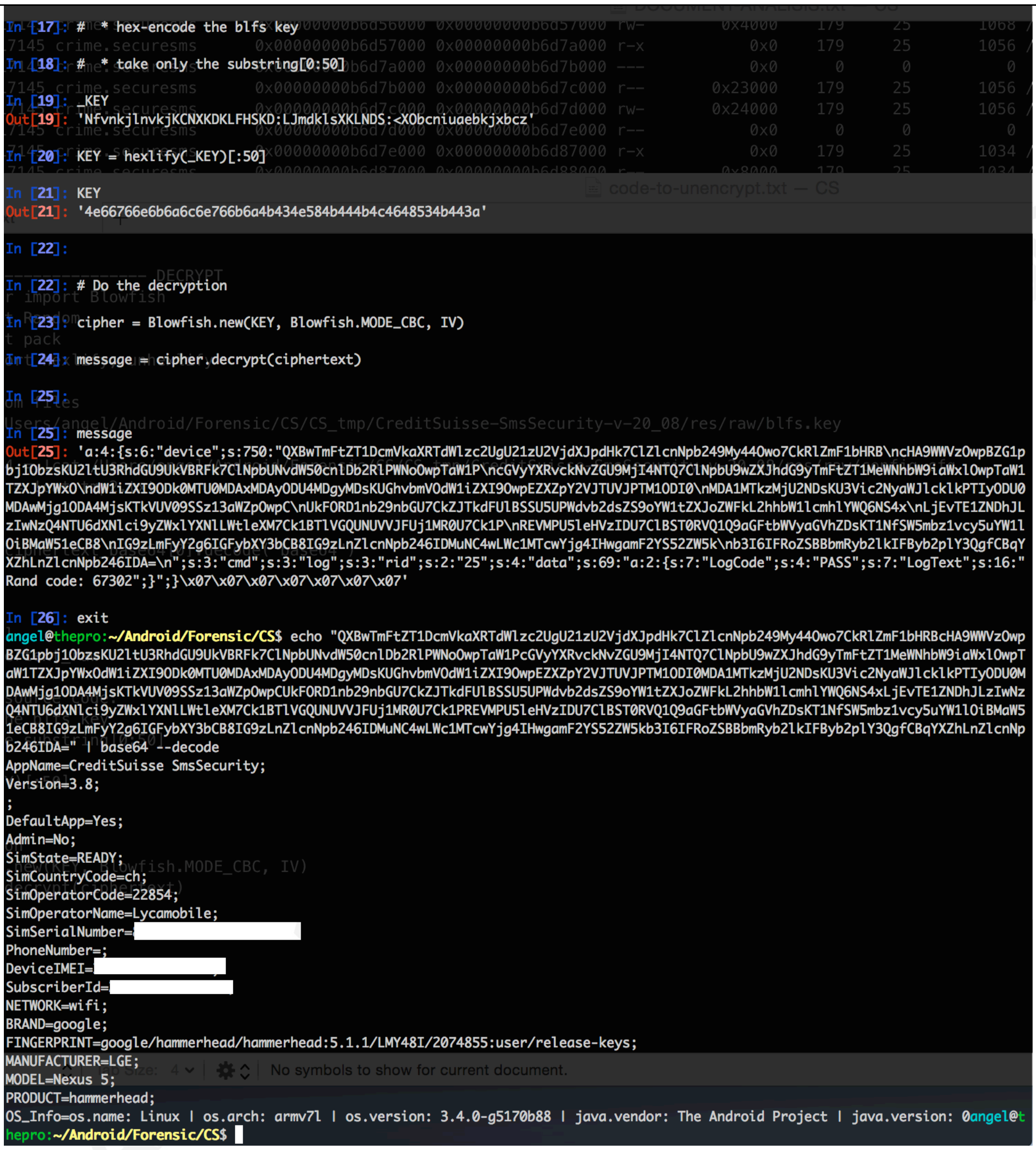
```
-ip2p0 -Dnl80211 -c/data/misc/wifi/p2p_supplicant.conf -
I/system/etc/wifi/p2p_supplicant_overlay.conf -puse_p2p_group_interface=1 -
e/data/misc/wifi/entropy.bin -g@android:wpa_wlan0
        954    10022  10022  com.android.systemui
        1093   10024  10024  com.google.android.googlequicksearchbox:interactor
        1117   10056  10056  com.google.android.inputmethod.latin
        1166   1027   1027   com.android.nfc
        1192   1001   1001   com.redbend.vdmc
        1213   1001   1001   com.android.phone
        1261   10024  10024  com.google.android.googlequicksearchbox
        1350   10009  10009  com.google.process.gapps
        1720   10009  10009  com.google.android.gms
        1742   10009  10009  com.google.android.gms.persistent
        1858   0      0      /system/bin/mpdecision --no_sleep --avg_comp
        2352   0      0      daemonsu:10087
        4084   0      0      daemonsu:0
        4086   0      0      daemonsu:0:4081
        4300   0      0      tmp-mksh -
        6311   10067  10067  com.google.android.apps.plus
        7463   10024  10024  com.google.android.googlequicksearchbox:search
        9547   0      0      daemonsu:10088
        24285  10006  10006  android.process.media
        24315  10065  10065  com.google.android.apps.photos
        28743  10008  10008  com.google.android.apps.gcs
        28796  10061  10061  com.google.android.apps.magazines
        32564  0      0       [kworker/0:0H]
        3447   0      0      [kworker/u:4]
        5478   0      0      [kworker/0:1H]
        7409   0      0      [kworker/0:0]
        9669   0      0      [kworker/u:7]
        10520  10014  10014  com.google.android.partnersetup
        10861  0      0       [kworker/u:12]
        12900  10035  10035
        13558  1000   1000   com.android.settings
        13964  0      0       [kworker/u:14]
        14116  1014   1014   /system/bin/dhcpcd -aABDKL -f
/system/etc/dhcpcd/dhcpcd.conf -h android-173db3c715e97b6 wlan0
        15087  10005  10005  com.android.defcontainer
        15575  99028  99028  com.android.chrome:sandboxed_process7
        15600  0      0       [kworker/0:1]
        16038  2000   2000   /system/bin/sh -
        16043  2000   2000   su -
        16046  0      0       daemonsu:0:16043
        16050  0      0       [kworker/0:3H]
        16063  10087  10087  eu.chainfire.supersu
        16185  0      0       tmp-mksh -
        16354  0      0       [kworker/0:2]
        16358  10091  10091  org.mozilla.firefox
        16626  10004  10004  android.process.acore
        16662  10017  10017  com.android.musicfx
        16681  10018  10018  com.android.vending
        16717  0      0      [kworker/0:3]
        16740  10009  10009  com.google.android.gms:car
        16763  10009  10009  com.google.android.gms.wearable
        16787  10041  10041  com.google.android.apps.docs
        16967  0      0      [kworker/u:0]
        16968  0      0      [kworker/u:1]
        17145  10093  10093  org.thoughtcrime.securesms
        17372  0      0      [kworker/u:2]
        17424  0      0      [kworker/0:2H]
        17439  0      0      [kworker/u:3]
        17686  0      0      insmod lime.ko path=TCP:4444 lime2.dump format=lime
        17687  0      0      [migration/1]
        17688  0      0      [kworker/1:0]
        17689  0      0      [kworker/1:0H]
        17690  0      0      [ksoftirqd/1]
        17691  0      0      [kworker/1:1H]
17692  0      0      [kworker/1:2H]
```

6.5. Appendix E: information decrypted with ipython and base64

```
# ipython script obtained from http://blog.dornea.nu/2014/07/07/disect-android-apks-like-a-pro-static-code-analysis/
```

```
from Crypto.Cipher import Blowfish
from Crypto import Random
from struct import pack
from binascii import hexlify, unhexlify

# Read content from files
blfs_key = !cat /Users/angel/Android/Forensic/CS/CS_tmp/CreditSuisse-
SmsSecurity-v-20_08/res/raw/blfs.key

#ciphertext_base64 = !cat
/Users/angel/Android/Forensic/CS/CS_tmp/CreditSuisse-SmsSecurity-v-
20_08/res/raw/config.cfg
ciphertext_base64 = !cat tmp2.txt # we change this file with the information to
decrypt



ciphertext_raw = ciphertext_base64[0].decode("base64")
#ciphertext_raw
# Some settings
IV = "12345678"
_KEY = blfs_key[0]
ciphertext = ciphertext_raw

# As seen in the source code:
#  * hex-encode the blfs key
#  * take only the substring[0:50]
_KEY
KEY = hexlify(_KEY)[:50]
KEY

# Do the decryption
cipher = Blowfish.new(KEY, Blowfish.MODE_CBC, IV)
message = cipher.decrypt(ciphertext)
message
```

```
# Output of the ipython script and the base64 and the decoding of the encoded string in
the URL
```



```
In [17]: # * hex-encode the blfs key

In [18]: # * take only the substring[0:50]

In [19]: _KEY
Out[19]: 'NfvnkjlnvkjKCNXKDKLFHSKD:LJmdklsXKLNDS:<XObcniuaebkjxbcz'

In [20]: KEY = hexlify(_KEY)[:50]

In [21]: KEY
Out[21]: '4e66766e6b6a6c6e766b6a4b434e584b444b4c4648534b443a'

In [22]:

In [22]: # Do the decryption

In [23]: cipher = Blowfish.new(KEY, Blowfish.MODE_CBC, IV)

In [24]: message = cipher.decrypt(ciphertext)

In [25]:

In [25]: message
Out[25]: 'a:4:{s:6:"device";s:750:"QXBwTmFtZT1DcmVkaXRTdWlzc2UgU21zU2VjdXJpdHk7ClZlcnNpb249My44Owo7CkRlZmF1bHRB\ncHA9WWVzOwpBZG1p
bj1ObzsKU2ltU3RhdGU9UkVBRFk7ClNpbUNvdW50cnlDb2RlPWNoOwpTaW1P\ncGVyYXRvckNvZGU9MjI4NTQ7ClNpbU9wZXJhdG9yTmFtZT1MeWNhbW9iaWxlOwpTaW1
TZXJpYWxO\ndW1iZXI9ODk0MTU0MDAxMDAyODU4MDgyMDsKUGhvbmVOdW1iZXI9OwpEZXZpY2VJTUVJPTM1ODI0\nMDA1MTkzMjU2NDsKU3Vic2NyaWJlcklkPTIyODU0
MDAwMjg1ODA4MjsKTkVUV09SSz13aWZpOwpC\nUkFORD1nb29nbGU7CkZJTkdFUlBSSU5UPWdvb2dsZS9oYW1tZXJoZWFkL2hhbW1lcmhlYWQ6NS4x\nLjEvTE1ZNDhJL
zIwNzQ4NTU6dXNlci9yZWxlYXNlLWtleXM7Ck1BTlVGQUNUVVJFUj1MR0U7Ck1P\nREVMPU5leHVzIDU7ClBST0RVQ1Q9aGFtbWVyaGVhZDsKT1NfSW5mbz1vcy5uYW1l
OiBMaW51eCB8\nIG9zLmFyY2g6IGFybXY3bCB8IG9zLnZlcnNpb246IDMuNC4wLWc1MTcwYjg4IHwgamF2YS52ZW5k\nb3I6IFRoZSBBbmRyb2lkIFByb2plY3QgfCBqY
XZhLnZlcnNpb246IDA=\n";s:3:"cmd";s:3:"log";s:3:"rid";s:2:"25";s:4:"data";s:69:"a:2:{s:7:"LogCode";s:4:"PASS";s:7:"LogText";s:16:"
Rand code: 67302";}";}\x07\x07\x07\x07\x07\x07\x07'

In [26]: exit
angel@thepro:~/Android/Forensic/CS$ echo "QXBwTmFtZT1DcmVkaXRTdWlzc2UgU21zU2VjdXJpdHk7ClZlcnNpb249My44Owo7CkRlZmF1bHRBcHA9WWVzOwp
BZG1pbj1ObzsKU2ltU3RhdGU9UkVBRFk7ClNpbUNvdW50cnlDb2RlPWNoOwpTaW1PcGVyYXRvckNvZGU9MjI4NTQ7ClNpbU9wZXJhdG9yTmFtZT1MeWNhbW9iaWxlOwpT
aW1TZXJpYWxOdW1iZXI9ODk0MTU0MDAxMDAyODU4MDgyMDsKUGhvbmVOdW1iZXI9OwpEZXZpY2VJTUVJPTM1ODI0MDA1MTkzMjU2NDsKU3Vic2NyaWJlcklkPTIyODU0M
DAwMjg1ODA4MjsKTkVUV09SSz13aWZpOwpCUkFORD1nb29nbGU7CkZJTkdFUlBSSU5UPWdvb2dsZS9oYW1tZXJoZWFkL2hhbW1lcmhlYWQ6NS4xLjEvTE1ZNDhJLzIwNz
Q4NTU6dXNlci9yZWxlYXNlLWtleXM7Ck1BTlVGQUNUVVJFUj1MR0U7Ck1PREVMPU5leHVzIDU7ClBST0RVQ1Q9aGFtbWVyaGVhZDsKT1NfSW5mbz1vcy5uYW1lOiBMaW5
1eCB8IG9zLmFyY2g6IGFybXY3bCB8IG9zLnZlcnNpb246IDMuNC4wLWc1MTcwYjg4IHwgamF2YS52ZW5kb3I6IFRoZSBBbmRyb2lkIFByb2plY3QgfCBqYXZhLnZlcnNp
b246IDA=" | base64 --decode
AppName=CreditSuisse SmsSecurity;
Version=3.8;
;
DefaultApp=Yes;
Admin=No;
SimState=READY;
SimCountryCode=ch;
SimOperatorCode=22854;
SimOperatorName=Lycamobile;
SimSerialNumber=[redacted]
PhoneNumber=;
DeviceIMEI=[redacted]
SubscriberId=[redacted]
NETWORK=wifi;
BRAND=google;
FINGERPRINT=google/hammerhead/hammerhead:5.1.1/LMY48I/2074855:user/release-keys;
MANUFACTURER=LGE;
MODEL=Nexus 5;
PRODUCT=hammerhead;
OS_Info=os.name: Linux | os.arch: armv7l | os.version: 3.4.0-g5170b88 | java.vendor: The Android Project | java.version: 0angel@t
hepro:~/Android/Forensic/CS$
```