Mobile Threats Incident Handling (Part II)

Toolset, Document for students

1.0

SEPTEMBER 2015

About ENISA

The European Union Agency for Network and Information Security (ENISA) is a centre of network and information security expertise for the EU, its member states, the private sector and Europe's citizens. ENISA works with these groups to develop advice and recommendations on good practice in information security. It assists EU member states in implementing relevant EU legislation and works to improve the resilience of Europe's critical information infrastructure and networks. ENISA seeks to enhance existing expertise in EU member states by supporting the development of cross-border communities committed to improving network and information security throughout the EU. More information about ENISA and its work can be found at www.enisa.europa.eu.

Authors

This document was created by Yonas Leguesse, Christos Sidiropoulos, and Lauri Palkmets in consultation with S-CURE[1] (The Netherlands), ComCERT[2] (Poland), and DFN-CERT Services[3] (Germany).

Contact

For contacting the authors please use cert-relations@enisa.europa.eu.
For media enquires about this paper, please use press@enisa.europa.eu.

[1] Don Stikvoort, Michael Potter, and Alan Robinson
[2] Tomasz Chlebowski, Mirosław Maj, Piotr Szeptyński, and Michał Tatar
[3] Mirko Wollenberg

Table of Contents

1. What Will You Learn?

1.1 Mobile forensics

Mobile forensics are a set of complex techniques aiming at the delivery of digital evidence based on data extracted from mobile devices. As such, it utilises approaches, technologies and tools known from computer forensics. Some of the concepts and solutions are common for both, while others are specifically for mobile forensics. Mobile forensic investigations (and digital forensic investigations in general) can be split into several phases: identification of a target mobile device, its seizure and data acquisition, examination and analysis, reasoning and reporting. Over the course of the investigation, activity must be documentation and gather evidence properly and securely stored.

This exercise will focus on the following phases: data acquisition (excluding physical approach) and examination and analysis (in terms of mobile device contents, application-specific data, malware and network communications).

1.2 Network forensic

Mobile network connectivity through technologies like GSM, UMTS, LTE is not commonly used in computer environments and thus has to be dealt with in a special way in a lab situation. One way is to use specialised commercial systems[4] available to law enforcement agencies. Alternatives are open-source implementations of management and data forwarding applications which, when combined with software defined radio (SDR) hardware, can be used to create a closed mobile network suitable for analysis.

The students will be given prepared samples of malware traffic captured with tcpdump[5] and mitmproxy[6] for analysis. After the exercise and accompanying studies the students should be aware of tools and techniques to build an environment to capture and analyse network traffic generated by mobile malware.

1.3 Mobile malware reverse-engineering

In this exercise the task will be to analyse malicious applications developed for mobile platforms (Android, iOS) and use a variety of tools to identify information leading to the development of countermeasures. Extracting the applications from a mobile device will not be part of this exercise.

We will demonstrate the analysis of two mobile malware applications (Android.Trojan.SLocker.DZ for Android and iOS.Oneclickfraud) using a couple of publically available tools and make the students acquainted with them.

[4] Cellular Intercept and Cellular Monitoring technologies give Law Enforcement and Government Agencies a technological edge, http://www.cellularintercept.com/, last accessed on: 2015-09-14
[5] Tcpdump: network traffic sniffer, http://www.tcpdump.org/, last accessed on: 2015-09-14
[6] MitM Proxy: An interactive console program that allows traffic flows to be intercepted, inspected, modified and replayed, https://mitmproxy.org/, last accessed on: 2015-09-14

2. Exercise Task

2.1 Task 2.1: Analysis of sample application's permissions on an Android device

2.1.1 Introduction

In this task, the students will use native Linux instrument called **AAPT**[7] which allows to take a look into permissions of the sample application. The AAPT tool can be used to list, add or remove resource files from apt packages (i.e. Android applications). If can also dump specific data from the packages.

2.1.2 Details

In the exercise directory (/home/enisa/D2/2.6_T1) students will find an APK application file com.androidream.secretdiary.free.apk. For the analysis of this file students will have to use the pre-installed AAPT tool.

2.1.3 Task walk-through

In this section a possible approach to permissions' analysis is explained.

2.1.3.1 Take a look for tool's options by running aapt.

```
Android Asset Packaging Tool

Usage:
 aapt l[ist] [-v] [-a] file.{zip,jar,apk}
   List contents of Zip-compatible archive.

 aapt d[ump] [--values] WHAT file.{apk} [asset [asset ...]]
   strings          Print the contents of the resource table string pool in the
APK.
   badging          Print the label and icon for the app declared in APK.
   permissions      Print the permissions from the APK.
   resources        Print the resource table from the APK.
   configurations   Print the configurations in the APK.
   xmltree          Print the compiled xmls in the given assets.
   xmlstrings       Print the strings of the given compiled xml assets.
```

Figure 1

2.1.3.2 Use command "aapt d permissions" to view the permissions of `com.androidream.secretdiary.free.apk` application.

```
enisa@ENISA-VirtualBox:~/Desktop/2.6 - task 1$ aapt d permissions com.androidream.secretdiary.free.apk

package: com.androidream.secretdiary.free
uses-permission: android.permission.VIBRATE
uses-permission: android.permission.INTERNET
uses-permission: android.permission.ACCESS_NETWORK_STATE
uses-permission: android.permission.PROCESS_OUTGOING_CALLS
uses-permission: android.permission.GET_ACCOUNTS
uses-permission: com.android.vending.BILLING
uses-permission: android.permission.WRITE_EXTERNAL_STORAGE
```

Figure 2

[7] Build System Overview, http://developer.android.com/sdk/installing/studio-build.html, last accessed on: 2015-09-14

Compare permissions granted to this application to all available permissions for Android applications[8] and describe what this specific application can do.

2.2 Task 2.2: Analysis of sample application's Mach-o header on an iOS device

2.2.1 Introduction

In this simple task the students will use tool called **OTOOL** which allows to take a look into Mach-O header of the sample iOS application. This application is available only for Mac OS X platform.

2.2.2 Details

You will have to use the otool on sample iOS application. You will find information about the FAT header to identify the supported CPU architecture to run an application. It's important to know how to run the application if it's needed to put the application into a sandbox.

2.2.3 Task walk-through

Students will need to download .IPA file directly from an iPhone or from the Internet. After that they will need to unzip .IPA file then check information from the FAT file header.

2.2.3.1 Unzip .IPA file.

```
iMac-mic:enisa enisa$ unzip Google_Maps.ipa
Archive:  Google_Maps.ipa
From Widow@iphonecake.com on 84 with RC325 (2015-08-06)  LP
  inflating: Payload/Google Maps.app/Info.plist
  inflating: iTunesMetadata.plist
  inflating: iTunesArtwork
  inflating: Payload/Google Maps.app/Google Maps
 extracting: Payload/Google Maps.app/Google Maps.crc
 extracting: Payload/Google Maps.app/Widow@iphonecake.com
   creating: Payload/Google Maps.app/_CodeSignature/
  inflating: Payload/Google Maps.app/_CodeSignature/CodeResources
 extracting: Payload/Google Maps.app/_CodeSignature/ResourceRules
```

Figure 3

[8] Android applications permissions, http://developer.android.com/preview/features/runtime-permissions.html, last accessed on: 2015-09-14

2.2.3.2 Use otool to check FAT header.

```
iMac-mic:enisa enisa$ otool -f Payload/Google\ Maps.app/Google\ Maps
Fat headers
fat_magic 0xcafebabe
nfat_arch 2
architecture 0
    cputype 12
    cpusubtype 9
    capabilities 0x0
    offset 16384
    size 12323936
    align 2^14 (16384)
architecture 1
    cputype 16777228
    cpusubtype 0
    capabilities 0x0
    offset 12353536
    size 15075584
    align 2^14 (16384)
```

Figure 4

Compare FAT_MAGIC value with Mach-O documentation[9] and answer the question: is the binary for a 32-bit platform, 64-bit platform, or are the binaries universal?

- 32-bit (ARMv6, ARMv7) – 0xFEEDFACE
- 64-bit – 0xFEEDFACF
- Universal binaries – 0xCAFEBABE

2.3 Task 3.1: A quick evaluation of knowledge regarding mobile devices

Please answer the following questions. Only one answer is correct in each question.

1. What information is contained in the IMEI number?
   a) The manufacturer's code
   b) Name of operator
   c) The number of home network
   d) None of the above

2. The ICCID number is:
   a) The number identifying the SIM card
   b) The serial number of the SIM card
   c) Number, which can be read without knowing the PIN
   d) All of the above

3. The IMSI identifies:
   a) Subscriber
   b) Phone
   c) The SIM card

[9] Universal Binaries and 32-bit/64-bit PowerPC Binaries, https://developer.apple.com/library/mac/documentation/DeveloperTools/Conceptual/MachORuntime/index.html#//apple_ref/c/tag/fat_header, last accessed on: 2015-09-14

d) The telephone

4. To disable communication capabilities of a seized mobile device which is turned on:
   a) Insert the device into the overvoltage bag
   b) Separate it from the network by pulling out SIM card
   c) Turn it off
   d) Put it in a Faraday's bag and analyse as soon as possible

5. How to check the IMEI of a device which is turned on?
   a) By entering *#06#
   b) By entering *##06#
   c) By entering *#08#
   d) By entering *##08#

6. What does "post mortem" extraction mean?
   a) Device is bricked
   b) Device is turned off
   c) Extraction will damage the device
   d) Device is turned on

7. How can the integrity of electronic evidence be ensured?
   a) By burning extracted data to a read-only medium
   b) By a checksum
   c) By following chain of custody
   d) All of the above

2.4 Task 4.1: Logical data extraction from Android devices

2.4.1 Introduction

In this task the students will use the AF Logical OSE tool to make a logical extraction from Android device. The trainer will give a short introduction to the usage of the Android AVD's and AF Logical OSE tool.

2.4.2 Tools used

- AVD
- AF Logical OSE

2.4.3 Details

Students have to prepare the Android Virtual Machine with the Android AVD tool. After that, they'll have to fill some data into Android Virtual Machine and once that is done, students can make a logical extraction. If they are any problems with creation of Virtual Machine and / or populating it with sample data students can use AVD called *Android_VM_ENISA*.

2.4.4 Task walk-through

The following steps explain how to make a logical extraction of an Android device.

2.4.4.1 Create new AVD machine. Open Linux Terminal and type android. You will see the SDK Manager window. Navigate to Tools and go to Manage AVDs. Click on Create and create a new AVD as shown in the picture below.

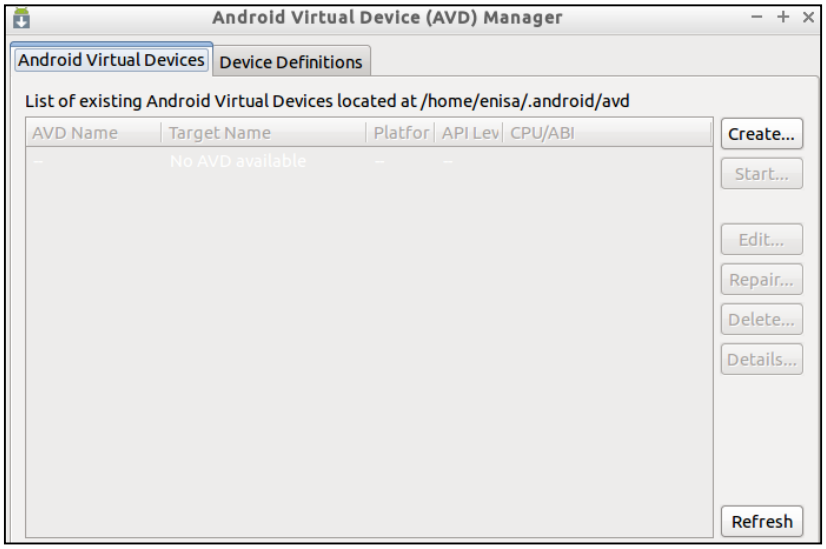

Figure 5 AVD Manager

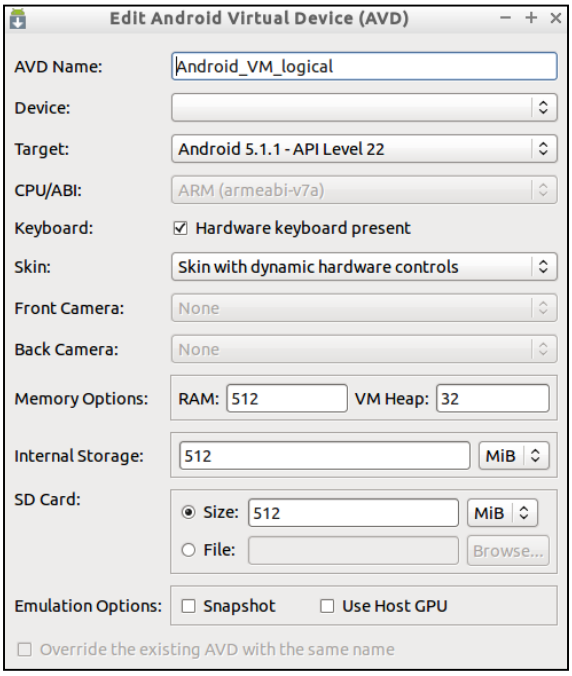

Figure 6 Create new AVD

2.4.4.2 Fill in sample data (e.g. add some contacts, try to send few SMS messages, try to call any number, open Internet browser, save some images on the internal memory and send some images by MMS message).

2.4.4.3 On Android Virtual Machine go to Settings -> Developer options and turn on USB debugging.

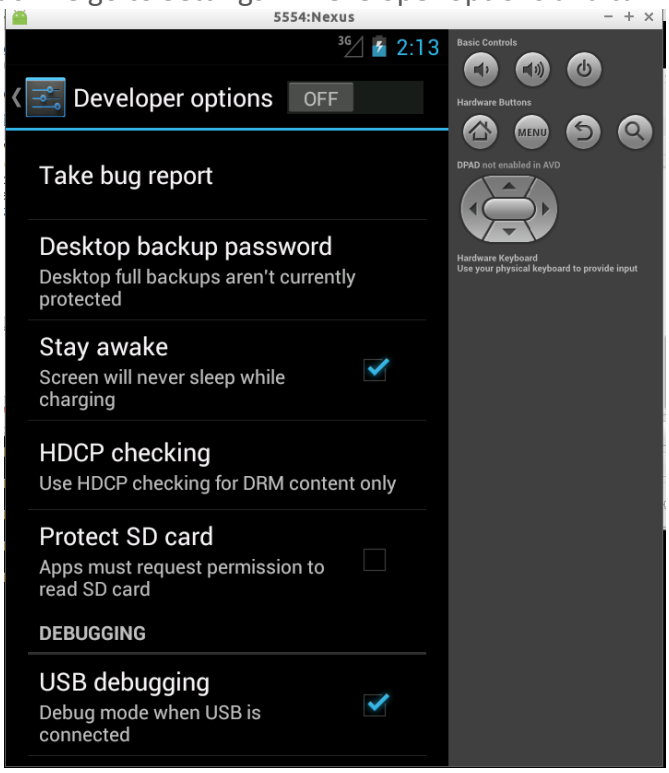

Figure 7 Enable USB debugging

2.4.4.4 Run the aflogical-ose command via terminal. By running this command you will push to the device a small application which tries to download data from the device.

```
enisa@ENISA-VirtualBox:~$ aflogical-ose
Make sure android device is connected to USB

479 KB/s (28794 bytes in 0.058s)
        pkg: /data/local/tmp/AFLogical-OSE_1.5.2.apk
Success

Starting: Intent { cmp=com.viaforensics.android.aflogical_ose/com.viaforensics.a
ndroid.ForensicsActivity }

Press enter to pull /sdcard/forensics into ~/aflogical-data/
```

Figure 8 aflogical-ose command

2.4.4.5 On the device's screen it can be seen types of information for downloading from the device memory. Click on capture and wait until you "Data extraction completed" message appears.

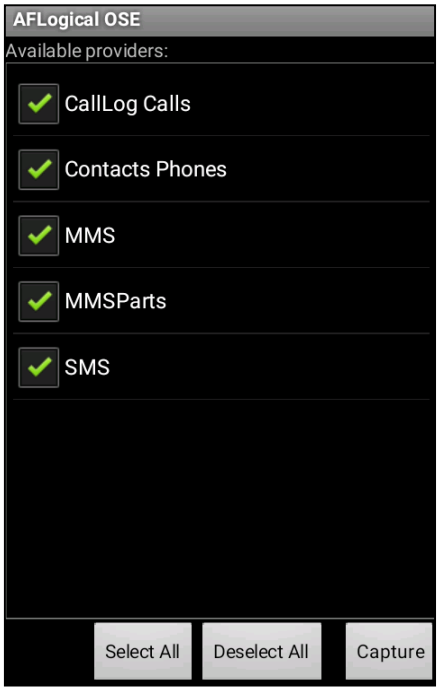

Figure 9

2.4.4.6 Now you need to get back into the terminal window and press enter to download data to the local hard disk.

```
Press enter to pull /sdcard/forensics into ~/aflogical-data/

pull: building file list...
pull: /sdcard/forensics/20150824.0942/Contacts Phones.csv -> /home/enisa/aflogic
al-data/20150824.0942/Contacts Phones.csv
pull: /sdcard/forensics/20150824.0942/CallLog Calls.csv -> /home/enisa/aflogical
-data/20150824.0942/CallLog Calls.csv
pull: /sdcard/forensics/20150824.0942/SMS.csv -> /home/enisa/aflogical-data/2015
0824.0942/SMS.csv
pull: /sdcard/forensics/20150824.0942/MMSParts.csv -> /home/enisa/aflogical-data
/20150824.0942/MMSParts.csv
pull: /sdcard/forensics/20150824.0942/MMS.csv -> /home/enisa/aflogical-data/2015
0824.0942/MMS.csv
pull: /sdcard/forensics/20150824.0942/info.xml -> /home/enisa/aflogical-data/201
50824.0942/info.xml
6 files pulled. 0 files skipped.
89 KB/s (58319 bytes in 0.639s)
```

Figure 10 aflogical-ose pull data from device

2.4.4.7 After that you have to find a folder called aflogical-data on your hard drive. In this folder you will find another folder named by the date and time of the extraction and when you will go deeper then you will find *.csv files with resources downloaded from the phone.

2.5 Task 4.2: File system extraction from Android devices

2.5.1 Introduction

In this task the students will use adb and dd tools to make a file system extraction from Android device. The trainer will give a short introduction into the usage of the Android AVD's and used specific commands.

2.5.2 Tools used

- AVD
- adb
- cat, dd, su, sudo

2.5.3 Details

Students have to prepare the Android Virtual Machine with the Android AVD tool. After that they'll have to fill some data into Android Virtual Machine and once that's done students can make logical extraction. If they are any problems with creation of Virtual Machine and / or populating it with sample data students can use AVD called *Android_VM_ENISA*.

2.5.4 Task walk-through

2.5.4.1 Firstly we connect the device through usb and enable usb debugging in the phone settings.

2.5.4.2 To identify the partition layout we connect to the device through adb shell and list partitions through /proc/partitions.

```
shell@shamu:/ $ su
root@shamu:/ # cat /proc/partitions
major minor  #blocks  name

 179        0   30535680 mmcblk0
 179        1     114688 mmcblk0p1
 179        2      16384 mmcblk0p2
 179        3        384 mmcblk0p3
 179        4         56 mmcblk0p4
 179        5         16 mmcblk0p5
 179        6         32 mmcblk0p6
 179        7       1024 mmcblk0p7
 179        8        256 mmcblk0p8
 179        9        512 mmcblk0p9
 179       10        500 mmcblk0p10
 179       11       4156 mmcblk0p11
 179       12        384 mmcblk0p12
 179       13       1024 mmcblk0p13
 179       14        256 mmcblk0p14
 179       15        512 mmcblk0p15
 179       16        500 mmcblk0p16
 179       17          4 mmcblk0p17
 179       18        512 mmcblk0p18
 179       19       1024 mmcblk0p19
 179       20       1024 mmcblk0p20
 179       21       1024 mmcblk0p21
 179       22       1024 mmcblk0p22
 179       23      16384 mmcblk0p23
 179       24      16384 mmcblk0p24
 179       25       2048 mmcblk0p25
 179       26      32768 mmcblk0p26
 179       27        256 mmcblk0p27
 179       28         32 mmcblk0p28
 179       29        128 mmcblk0p29
 179       30       8192 mmcblk0p30
 179       31       1024 mmcblk0p31
 259        0       2528 mmcblk0p32
 259        1          1 mmcblk0p33
 259        2          8 mmcblk0p34
 259        3      16400 mmcblk0p35
 259        4       9088 mmcblk0p36
 259        5      16384 mmcblk0p37
 259        6     262144 mmcblk0p38
 259        7      65536 mmcblk0p39
 259        8       1024 mmcblk0p40
 259        9    2097152 mmcblk0p41
 259       10   27807616 mmcblk0p42
 179       32       4096 mmcblk0rpmb
 254        0   27807616 dm-0
root@shamu:/ #
```

Figure 11 /proc/partitions

2.5.4.3 Alternatively to better understand the mount points we list partitions by name.

```
root@shamu:/ # ls -la /dev/block/platform/msm_sdcc.1/by-name/
lrwxrwxrwx root     root              1970-09-09 07:43 aboot -> /dev/block/mmcblk0p7
lrwxrwxrwx root     root              1970-09-09 07:43 abootBackup -> /dev/block/mmcblk0p13
lrwxrwxrwx root     root              1970-09-09 07:43 boot -> /dev/block/mmcblk0p37
lrwxrwxrwx root     root              1970-09-09 07:43 cache -> /dev/block/mmcblk0p38
lrwxrwxrwx root     root              1970-09-09 07:43 cid -> /dev/block/mmcblk0p29
lrwxrwxrwx root     root              1970-09-09 07:43 ddr -> /dev/block/mmcblk0p6
lrwxrwxrwx root     root              1970-09-09 07:43 frp -> /dev/block/mmcblk0p18
lrwxrwxrwx root     root              1970-09-09 07:43 keystore -> /dev/block/mmcblk0p24
lrwxrwxrwx root     root              1970-09-09 07:43 kpan -> /dev/block/mmcblk0p36
lrwxrwxrwx root     root              1970-09-09 07:43 logo -> /dev/block/mmcblk0p30
lrwxrwxrwx root     root              1970-09-09 07:43 logs -> /dev/block/mmcblk0p25
lrwxrwxrwx root     root              1970-09-09 07:43 mdm1dhob -> /dev/block/mmcblk0p28
lrwxrwxrwx root     root              1970-09-09 07:43 mdm1hob -> /dev/block/mmcblk0p27
lrwxrwxrwx root     root              1970-09-09 07:43 mdm1m9kefs1 -> /dev/block/mmcblk0p19
lrwxrwxrwx root     root              1970-09-09 07:43 mdm1m9kefs2 -> /dev/block/mmcblk0p20
lrwxrwxrwx root     root              1970-09-09 07:43 mdm1m9kefs3 -> /dev/block/mmcblk0p21
lrwxrwxrwx root     root              1970-09-09 07:43 mdm1m9kefsc -> /dev/block/mmcblk0p33
lrwxrwxrwx root     root              1970-09-09 07:43 metadata -> /dev/block/mmcblk0p2
lrwxrwxrwx root     root              1970-09-09 07:43 misc -> /dev/block/mmcblk0p31
lrwxrwxrwx root     root              1970-09-09 07:43 modem -> /dev/block/mmcblk0p1
lrwxrwxrwx root     root              1970-09-09 07:43 oem -> /dev/block/mmcblk0p39
lrwxrwxrwx root     root              1970-09-09 07:43 padA -> /dev/block/mmcblk0p11
lrwxrwxrwx root     root              1970-09-09 07:43 padB -> /dev/block/mmcblk0p22
lrwxrwxrwx root     root              1970-09-09 07:43 padC -> /dev/block/mmcblk0p40
lrwxrwxrwx root     root              1970-09-09 07:43 padD -> /dev/block/mmcblk0p32
lrwxrwxrwx root     root              1970-09-09 07:43 persist -> /dev/block/mmcblk0p26
lrwxrwxrwx root     root              1970-09-09 07:43 recovery -> /dev/block/mmcblk0p35
lrwxrwxrwx root     root              1970-09-09 07:43 rpm -> /dev/block/mmcblk0p8
lrwxrwxrwx root     root              1970-09-09 07:43 rpmBackup -> /dev/block/mmcblk0p14
lrwxrwxrwx root     root              1970-09-09 07:43 sbl1 -> /dev/block/mmcblk0p3
lrwxrwxrwx root     root              1970-09-09 07:43 sbl1bak -> /dev/block/mmcblk0p12
lrwxrwxrwx root     root              1970-09-09 07:43 sdi -> /dev/block/mmcblk0p4
lrwxrwxrwx root     root              1970-09-09 07:43 sec -> /dev/block/mmcblk0p5
lrwxrwxrwx root     root              1970-09-09 07:43 sp -> /dev/block/mmcblk0p23
lrwxrwxrwx root     root              1970-09-09 07:43 ssd -> /dev/block/mmcblk0p34
lrwxrwxrwx root     root              1970-09-09 07:43 system -> /dev/block/mmcblk0p41
lrwxrwxrwx root     root              1970-09-09 07:43 tz -> /dev/block/mmcblk0p10
lrwxrwxrwx root     root              1970-09-09 07:43 tzBackup -> /dev/block/mmcblk0p16
lrwxrwxrwx root     root              1970-09-09 07:43 userdata -> /dev/block/mmcblk0p42
lrwxrwxrwx root     root              1970-09-09 07:43 utags -> /dev/block/mmcblk0p9
lrwxrwxrwx root     root              1970-09-09 07:43 utagsBackup -> /dev/block/mmcblk0p15
lrwxrwxrwx root     root              1970-09-09 07:43 versions -> /dev/block/mmcblk0p17
```

Figure 12 Partitions By Name

2.5.4.4 Next, extract the system.img to the root of the sdcard.

```
root@shamu:/ # dd if=/dev/block/mmcblk0p41 of=/sdcard/system.img

4194304+0 records in
4194304+0 records out
2147483648 bytes transferred in 539.731 secs (3978803 bytes/sec)
```

Figure 13 Backup of system.img to file

2.5.4.5 Finally we pull the `system.img` locally to further investigate.

```
C:\Program Files (x86)\Android\android-sdk\platform-tools>adb pull /sdcard/system.img
4571 KB/s (2147483648 bytes in 458.745s)
```

Figure 14

2.6 Task 4.3: Manual file carving

2.6.1 Introduction

In this task, the students will use wxHexEditor tool to perform file carving from a file system imaged of an Android device. The trainer will give a short introduction into the usage of the wxHexEditor and tells something about file signatures.

2.6.2 Tools used

- wxHexEditor

2.6.3 Details

This exercise refers to Task 2 above. Now that students have dumped a partition from Android Virtual Machine, the goal is to find some JPG file in the partition image.

2.6.4 Task walk-through

A file signature is data used to identify or verify the content of a file. In particular, it may be a so called magic number which identifies the format of the file. Generally a short sequence of bytes (most magic numbers are 2–4 bytes long) is placed at the beginning of the file.

Manual file carving is the process of reconstructing files by scanning the raw image of the disk looking for file signatures and its contents, and reassembling them. This is usually done by examining the header (the first few bytes) and footer (the last few bytes) of a file.

2.6.4.1 Open wxHexEditor by typing in Linux Terminal command wxHexEditor and open one of partition file by clicking on File -> Open.

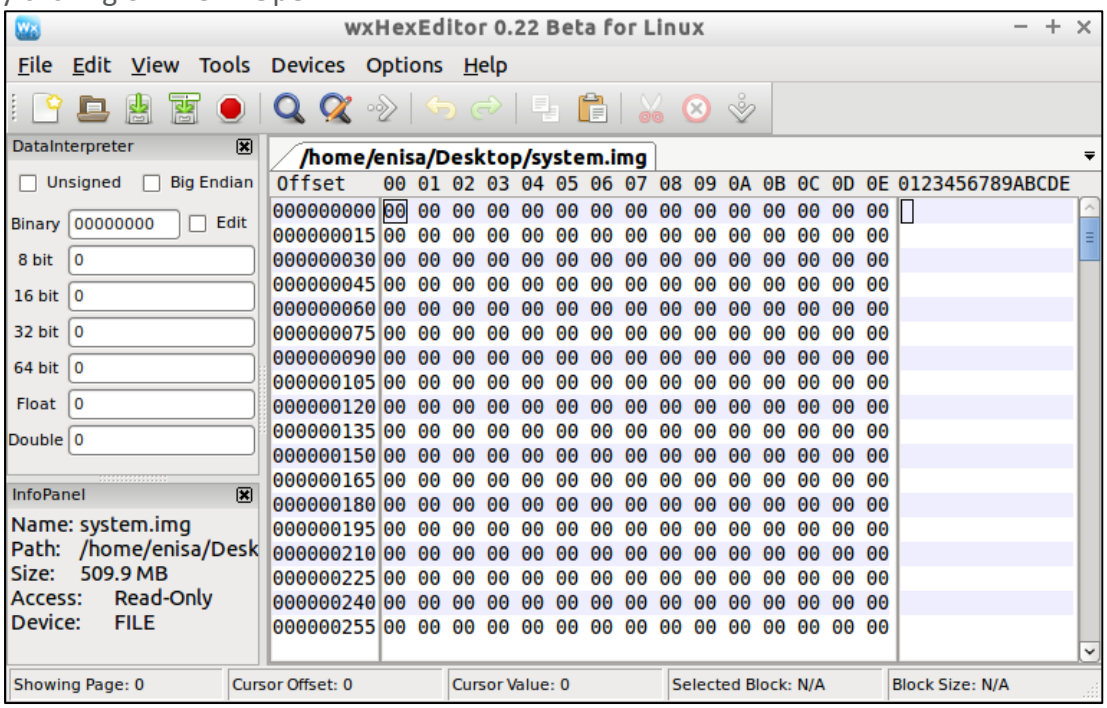

Figure 15

2.6.4.2 You need to find a header of a JPG file which is FF D8 FF (list of all known file signatures can be found on the Internet). It is important, that the header is a few bytes ahead of an ASCII string "JFIF". Use search tool ("Edit -> Find -> Find All") directly from wxHexEditor.

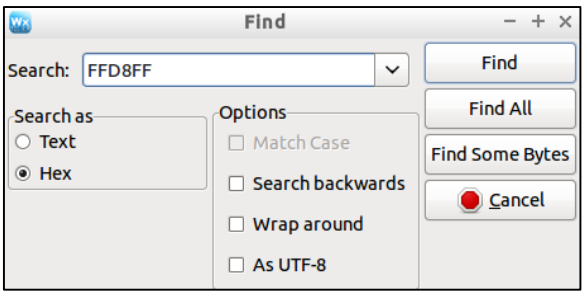

Figure 16

2.6.4.3 Save the offset address of a JPG file header.

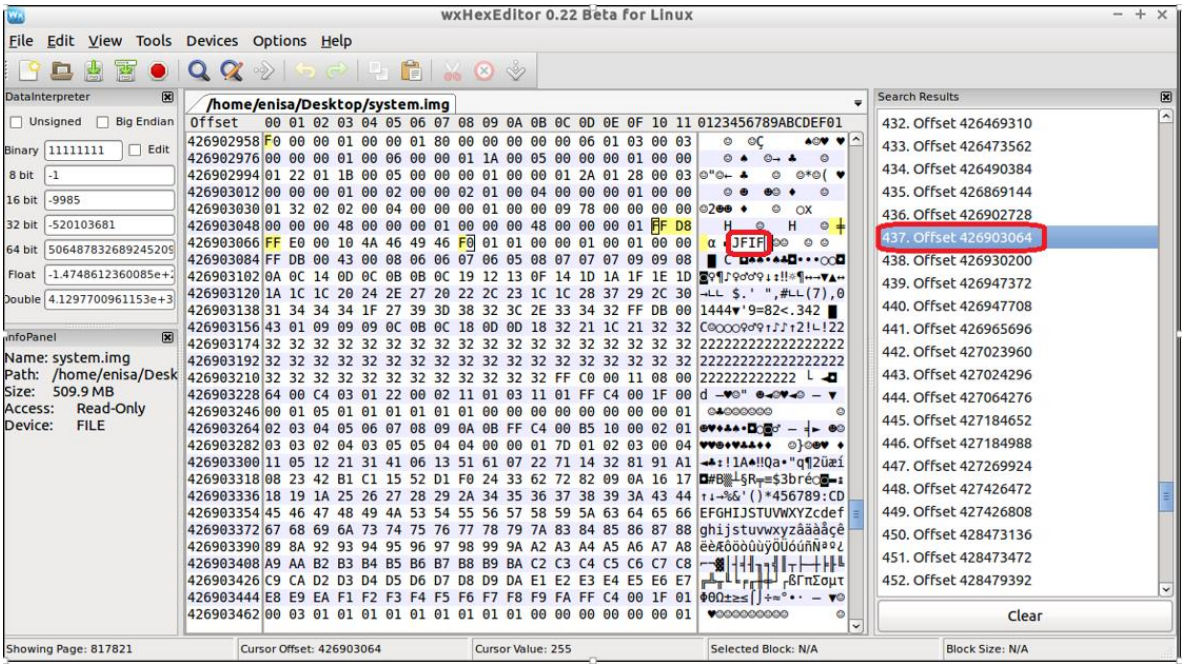

Figure 17

2.6.4.4 Now you need to find a hex value of FF D9 which is a JPEG files' footer. Try to locate the nearest occurrence after the header identified in the previous step.

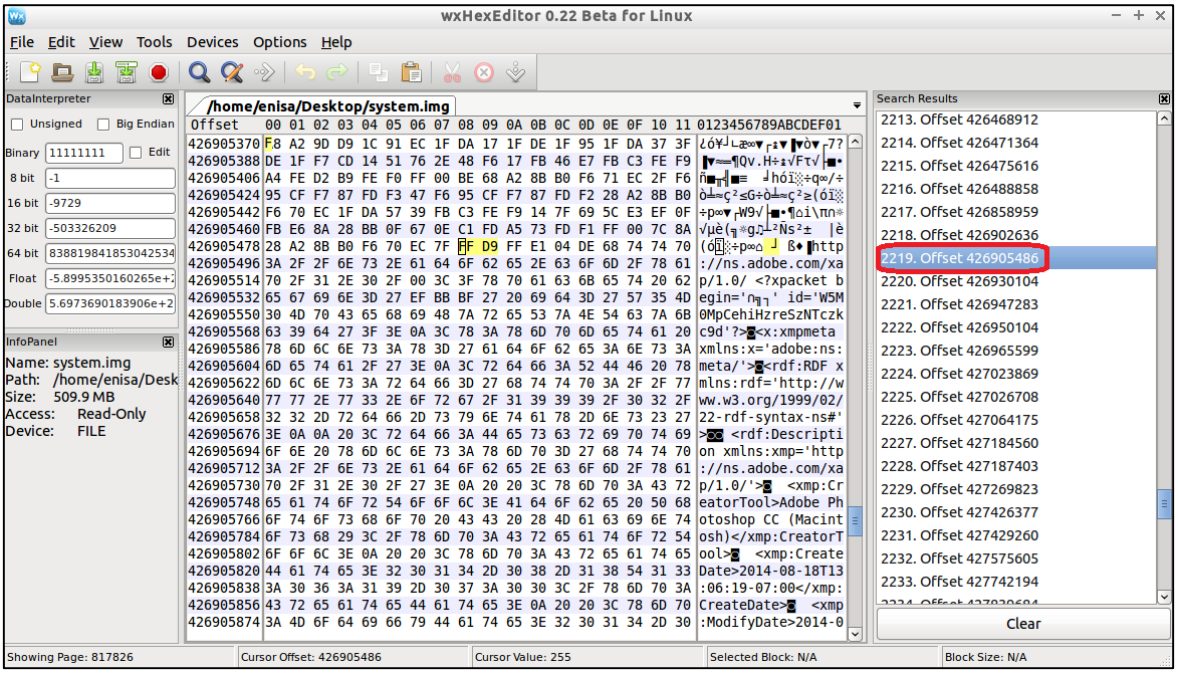

Figure 18

2.6.4.5 When you find the header and trailer of JPG file you have to mark hexadecimal code from header to footer (you need to know offset addresses).

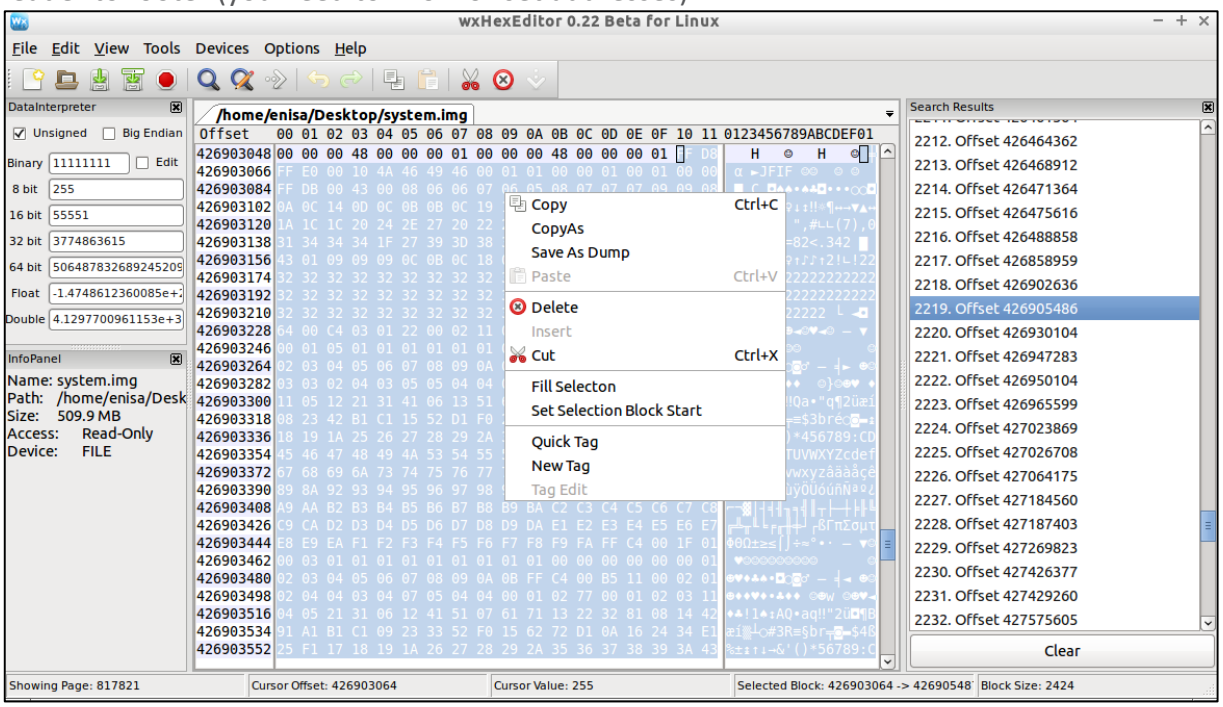

Figure 19

2.6.4.6 Now you choose "Save As Dump" and save this file as TEST.JPG. After that you will be able to see the picture.

It's worth to mention that file carving can be done automatically with commercial tools, such as Micro Systemation's XACT, or open-source ones, e.g. foremost.

2.7 Task 4.4: RAM memory dump from Android device

2.7.1 Introduction

In this task the students will use tools to make a RAM memory extraction from Android device. After that students will use Volatility and Autopsy tools to analyse RAM dump file.

2.7.2 Tools used

- Android SDK[10]
- Android NDK[11]
- LiME[12]
- Dwarfdump[13]
- Volatility[14]
- Autopsy[15]

[10] Android SDK, http://developer.android.com/sdk/index.html, last accessed on: 2015-09-14
[11] Android NDK, http://developer.android.com/tools/sdk/ndk/index.html, last accessed on: 2015-09-14
[12] LiME: Linux Memory Extractor, https://github.com/504ensicslabs/lime last accessed on: 2015-09-14
[13] Libdwarf and Dwarfdump, http://wiki.dwarfstd.org/index.php?title=Libdwarf_And_Dwarfdump, last accessed on: 2015-09-14
[14] Volatility: RAM dump analyser, https://code.google.com/p/volatility/wiki/, last accessed on: 2015-09-14
[15] Autopsy® is a digital forensics platform and graphical interface to The Sleuth Kit® and other digital forensics tools, http://www.sleuthkit.org/autopsy/, last accessed on: 2015-09-14

2.7.3 Details

This exercise will explain how to:

- Perform Android device memory forensics with Volatility,
- Set up Android build environment,
- Cross-compile of Android kernel,
- Use the Android Emulator,
- Acquire memory from Android devices with LiME module,
- Build Volatility profile for Android,
- Run Volatility commands against Android memory dumps.

2.7.4 Task walk-through

2.7.5 Dumping RAM memory

```
git clone https://android.googlesource.com/kernel/goldfish
```

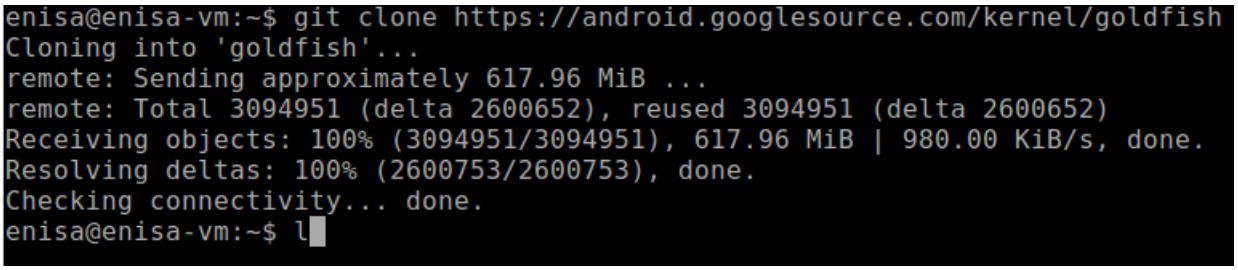

Figure 20 Download of kernel source code

2.7.5.1 The android kernel has different versions that are split into branches. You can check the branches by issuing git branch –a inside the kernel source folder.

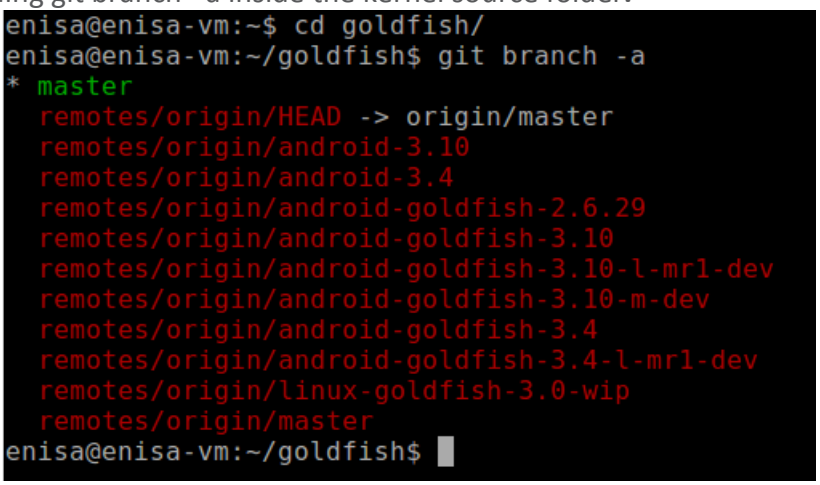

Figure 21 Kernel branches

2.7.5.2 For this exercise the android kernel version 2.6.29 is going to be used. To do this students are going to create a new branch named lime tracking the 2.6.29 kernel source. Issue the following command:

```
git branch --track lime remotes/origin/android-goldfish-2.6.29
```

```
git checkout lime
```

```
enisa@enisa-vm:~/goldfish$ git branch --track lime remotes/origin/android-goldfish-2.6.29
Branch lime set up to track remote branch android-goldfish-2.6.29 from origin.
enisa@enisa-vm:~/goldfish$ git branch -a
  lime
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/android-3.10
  remotes/origin/android-3.4
  remotes/origin/android-goldfish-2.6.29
  remotes/origin/android-goldfish-3.10
  remotes/origin/android-goldfish-3.10-l-mr1-dev
  remotes/origin/android-goldfish-3.10-m-dev
  remotes/origin/android-goldfish-3.4
  remotes/origin/android-goldfish-3.4-l-mr1-dev
  remotes/origin/linux-goldfish-3.0-wip
  remotes/origin/master
enisa@enisa-vm:~/goldfish$ git checkout lime
Checking out files: 100% (26821/26821), done.
Switched to branch 'lime'
Your branch is up-to-date with 'origin/android-goldfish-2.6.29'.
```

Figure 22 Create branch lime and checkout

2.7.5.3 In order to compile the kernel you will need the configuration file. Usually the configuration file for the kernel is included by the OEM in the source. Additionally if the kernel that is built in the device supports it, it can be extracted from it by pulling the /proc/config.gz file. In this case students will use the included configuration file in *arch/arm/configs/goldfish_armv7_defconfig* . First you will need to set the environment variables to compile the kernel as shown below:

```
enisa@enisa-vm:~/goldfish$ export ARCH=arm
enisa@enisa-vm:~/goldfish$ export SUBARCH=arm
enisa@enisa-vm:~/goldfish$ export CROSS_COMPILE=/usr/share/android-
ndk/toolchains/arm-linux-androideabi-4.6/prebuilt/linux-x86_64/bin/arm-
linux-androideabi-
```

2.7.5.4 Afterwards students will create the initial configuration file from the goldfish default configuration.

```
enisa@enisa-vm:~/goldfish$ make goldfish_armv7_defconfig
  HOSTCC  scripts/basic/fixdep
scripts/basic/fixdep.c: In function 'traps':
scripts/basic/fixdep.c:377:2: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
  if (*(int *)test != INT_CONF) {
  ^
scripts/basic/fixdep.c:379:4: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
    *(int *)test);
    ^
  HOSTCC  scripts/basic/docproc
  HOSTCC  scripts/basic/hash
  HOSTCC  scripts/kconfig/conf.o
scripts/kconfig/conf.c: In function 'conf_sym':
scripts/kconfig/conf.c:159:6: warning: variable 'type' set but not used [-Wunused-but-set-variable]
  int type;
      ^
scripts/kconfig/conf.c: In function 'conf_choice':
scripts/kconfig/conf.c:231:6: warning: variable 'type' set but not used [-Wunused-but-set-variable]
  int type;
      ^
scripts/kconfig/conf.c: In function 'conf_askvalue':
scripts/kconfig/conf.c:105:8: warning: ignoring return value of 'fgets', declared with attribute warn_unused_result [-Wunused-result]
   fgets(line, 128, stdin);
        ^
scripts/kconfig/conf.c: In function 'conf_choice':
scripts/kconfig/conf.c:307:9: warning: ignoring return value of 'fgets', declared with attribute warn_unused_result [-Wunused-result]
    fgets(line, 128, stdin);
         ^
  HOSTCC  scripts/kconfig/kxgettext.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/lex.zconf.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
arch/arm/configs/goldfish_armv7_defconfig:292:warning: override: FB_EARLYSUSPEND changes choice state
#
# configuration written to .config
#
enisa@enisa-vm:~/goldfish$
```

Figure 23 Initial config file creation

2.7.5.5 Next edit the configuration file to enable module loading. Open .config file and edit line 115 as shown below:

```
110 CONFIG_HAVE_KRETPROBES=y
111 CONFIG_HAVE_GENERIC_DMA_COHERENT=y
112 CONFIG_SLABINFO=y
113 CONFIG_RT_MUTEXES=y
114 CONFIG_BASE_SMALL=0
115 CONFIG_MODULES=y
116 CONFIG_MODULES_UNLOAD=y
117 CONFIG_MODULES_FORCE_UNLOAD=y
118 CONFIG_BLOCK=y
119 # CONFIG_LBD is not set
120 # CONFIG_BLK_DEV_IO_TRACE is not set
121 # CONFIG_BLK_DEV_BSG is not set
122 # CONFIG_BLK_DEV_INTEGRITY is not set
123
```

Figure 24 Configuration change

2.7.5.6 Compile the kernel issuing make command. When finished the compiled kernel should be in `arch/arm/boot/zImage`.

```
OBJCOPY arch/arm/boot/Image
Kernel: arch/arm/boot/Image is ready
AS      arch/arm/boot/compressed/head.o
GZIP    arch/arm/boot/compressed/piggy.gz
AS      arch/arm/boot/compressed/piggy.o
CC      arch/arm/boot/compressed/misc.o
LD      arch/arm/boot/compressed/vmlinux
OBJCOPY arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
Building modules, stage 2.
MODPOST 1 modules
CC      drivers/hid/hid-dummy.mod.o
LD [M]  drivers/hid/hid-dummy.ko
```

Figure 25 Kernel Compilation

2.7.5.7 Now you can start the emulator with the kernel you have just compiled issuing the following command.

```
enisa@enisa-vm:~/goldfish$ emulator -avd Nexus -kernel
arch/arm/boot/zImage
```

2.7.5.8 Next you need to download the lime module and compile it.

```
enisa@enisa-vm:~$ git clone https://github.com/504ensicsLabs/LiME
```

```
enisa@enisa-vm:~$ git clone https://github.com/504ensicsLabs/LiME
Cloning into 'LiME'...
remote: Counting objects: 82, done.
remote: Total 82 (delta 0), reused 0 (delta 0), pack-reused 82
Unpacking objects: 100% (82/82), done.
Checking connectivity... done.
enisa@enisa-vm:~$ cd LiME/src/
enisa@enisa-vm:~/LiME/src$
```

Figure 26 Download lime source

2.7.5.9 Edit the Makefile accordingly.

```
enisa@enisa-vm:~/LiME/src$ diff Makefile ~/Makefile
25a26,27
> KDIR_GOLDFISH := ~/goldfish
> CCPATH :=/usr/share/android-ndk/toolchains/arm-linux-androideabi-
4.6/prebuilt/linux-x86_64/bin/
33,35c35,38
<     $(MAKE) -C /lib/modules/$(KVER)/build M=$(PWD) modules
<     strip --strip-unneeded lime.ko
<     mv lime.ko lime-$(KVER).ko
---
>     $(MAKE) ARCH=arm CROSS_COMPILE=$(CCPATH)/arm-linux-androideabi- -C
$(KDIR_GOLDFISH) EXTRA_CFLAGS=-fno-pic M=$(PWD) modules
>     mv lime.ko lime-goldfish.ko
```

Then compile the module issuing make

```
enisa@enisa-vm:~/LiME/src$ make
make ARCH=arm CROSS_COMPILE=/usr/share/android-ndk/toolchains/arm-linux-androideabi-4.6/prebuilt/linux-x86_64/bin//arm-linux-androideabi- -C ~/goldfish EXTRA_CFLAGS=-fno-pic M=/home/enisa/L
iME/src modules
make[1]: Entering directory `/home/enisa/goldfish'
  CC [M]  /home/enisa/LiME/src/tcp.o
  CC [M]  /home/enisa/LiME/src/disk.o
  CC [M]  /home/enisa/LiME/src/main.o
  LD [M]  /home/enisa/LiME/src/lime.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/enisa/LiME/src/lime.mod.o
  LD [M]  /home/enisa/LiME/src/lime.ko
make[1]: Leaving directory `/home/enisa/goldfish'
mv lime.ko lime-goldfish.ko
```

Figure 27 Module compilation

2.7.5.10 Next push the compiled module to the running emulator and set the memory dump path. Afterwards you can pull the memory dump as show below:

```
enisa@enisa-vm:~/LiME/src$ adb push lime-goldfish.ko /sdcard/lime.ko
184 KB/s (10528 bytes in 0.055s)
enisa@enisa-vm:~/LiME/src$ adb shell
root@android:/ # insmod /sdcard/lime.ko "path=/sdcard/lime.dump format=lime"


root@android:/ #
root@android:/ #
root@android:/ # exit
enisa@enisa-vm:~/LiME/src$ adb pull /sdcard/lime.dump

2780 KB/s (359661600 bytes in 126.327s)
```

Figure 28 LiMe memory dump.

2.7.6 Examining memory dump with Volatility

2.7.6.1 Build a Volatility Profile

Volatility uses profiles to properly analyse RAM dump. For Android an already prepared profile called LinuxGolfish-2_6_29ARM should be used. If the students want to create their own profiles, they should refer to another exercise by ENISA: https://www.enisa.europa.eu/activities/cert/training/training-resources/documents/advanced-artifact-handling-handbook (Section 2.2.2.3, Task 1.2.3 Building a Volatility profile).

2.7.6.2 Examine the Memory Dump with Volatility

Since Android is based on Linux, students can use any of the Linux-related Volatility commands[16] to analyse the memory dump. Mostly used commands for Android are explained below. The descriptions are copied from Volatility project website:

- linux_pslist

This plugin prints the list of active processes starting from the init_task symbol and walking the task_struct->tasks linked list. It does not display the swapper process. If the DTB column is blank, the item is likely a kernel thread.

```
enisa@ENISA-VirtualBox:~$ volatility --profile=LinuxGolfish-2_6_29ARM -f ram.lime linux_pslist
Volatility Foundation Volatility Framework 2.3.1
Offset     Name                 Pid             Uid             Gid    DTB        Start Time
---------- -------------------- --------------- --------------- ------ ---------- ----------
0xdf812c00 init                 1               0               0      0x1fc48000 2015-08-24 12:13:55 UTC+0000
0xdf812800 kthreadd             2               0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf812400 ksoftirqd/0          3               0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf812000 events/0             4               0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf819c00 khelper              5               0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf819800 suspend              6               0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf819400 kblockd/0            7               0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf819000 cqueue               8               0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf85ac00 kseriod              9               0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf85a800 kmmcd                10              0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf85a400 pdflush              11              0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf85a000 pdflush              12              0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf9acc00 kswapd0              13              0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf9ac800 aio/0                14              0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf9ac000 mtdblockd            25              0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf9ac400 kstriped             26              0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf9ce000 hid_compat           27              0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf9cec00 rpciod/0             30              0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf9cd000 mmcqd                31              0               0      ---------- 2015-08-24 12:13:55 UTC+0000
0xdf9ce400 ueventd              32              0               0      0x1fc70000 2015-08-24 12:13:55 UTC+0000
0xdf9ce800 servicemanager       33              1000            1000   0x1fd58000 2015-08-24 12:13:56 UTC+0000
0xdf9cd400 vold                 34              0               0      0x1fdf0000 2015-08-24 12:13:56 UTC+0000
```

Figure 29 linux_pslist

- linux_proc_maps

This plugin prints details of process memory, including heaps, stacks, and shared libraries.

[16] Volatility: A command reference for Linux, https://code.google.com/p/volatility/wiki/LinuxCommandReference23, last accessed on: 2015-09-14

```
enisa@ENISA-VirtualBox:~$ volatility --profile=LinuxGolfish-2_6_29ARM -f ram.lime linux_proc_maps
Volatility Foundation Volatility Framework 2.3.1
Pid      Start              End                Flags       Pgoff Major  Minor  Inode      File Path
-------- ------------------ ------------------ ------ ---------- ------ ------ ---------- --------------------
------------------------------------------------------------
       1 0x0000000000008000 0x0000000000022000 r-x           0x0      0      1         19 /init

       1 0x0000000000022000 0x0000000000024000 rw-       0x19000      0      1         19 /init

       1 0x0000000000024000 0x0000000000034000 rw-           0x0      0      0          0 [heap]

       1 0x0000000040000000 0x0000000040001000 r--           0x0      0      0          0

       1 0x0000000040001000 0x0000000040009000 rw-           0x0      0     10         47 /dev/__properties__

       1 0x00000000bef6b000 0x00000000bef80000 rw-           0x0      0      0          0 [stack]

      32 0x0000000000008000 0x0000000000022000 r-x           0x0      0      1         19 /init

      32 0x0000000000022000 0x0000000000024000 rw-       0x19000      0      1         19 /init

      32 0x0000000000024000 0x0000000000033000 rw-           0x0      0      0          0 [heap]

      32 0x0000000040000000 0x0000000040008000 r--           0x0      0     10         47 /dev/__properties__
```

Figure 30 linux_proc_maps

- linux_arp

This plugin prints the ARP table.

```
enisa@ENISA-VirtualBox:~$ volatility --profile=LinuxGolfish-2_6_29ARM -f ram.lim
e linux_arp
Volatility Foundation Volatility Framework 2.3.1
[::                                    ] at 00:00:00:00:00:00    on lo
[10.0.2.3                              ] at 52:54:00:12:35:03    on eth0
[0.0.0.0                               ] at 00:00:00:00:00:00    on lo
[10.0.2.2                              ] at 52:54:00:12:35:02    on eth0
```

Figure 31 linux_arp

- linux_ifconfig

This plugin prints the active interface information, including IPs, interface name, MAC address, and whether the NIC is in promiscuous mode or not (sniffing).

```
enisa@ENISA-VirtualBox:~$ volatility --profile=LinuxGolfish-2_6_29ARM -f ram.lim
e linux_ifconfig
Volatility Foundation Volatility Framework 2.3.1
Interface        IP Address           MAC Address        Promiscous Mode
---------------- -------------------- ------------------ ---------------
lo               127.0.0.1            00:00:00:00:00:00  False
eth0             10.0.2.15            00:00:00:00:00:00  False
```

Figure 32 linux_ifconfig

- linux_route_cache

This plugin enumerates the data in the routing table cache. It can show you which systems a machine communicated with in the past.

```
enisa@ENISA-VirtualBox:~$ volatility --profile=LinuxGolfish-2_6_29ARM -f ram.lime linux_route_cache
Volatility Foundation Volatility Framework 2.3.1
Interface       Destination         Gateway
--------------- ------------------- -------
eth0            94.154.96.7         10.0.2.2
eth0            77.237.27.32        10.0.2.2
eth0            216.58.209.65       10.0.2.2
eth0            77.237.27.25        10.0.2.2
eth0            10.0.2.2            10.0.2.2
eth0            149.156.70.60       10.0.2.2
eth0            77.237.27.45        10.0.2.2
eth0            10.0.2.3            10.0.2.3
eth0            77.237.27.24        10.0.2.2
eth0            77.237.27.25        10.0.2.2
eth0            77.237.27.59        10.0.2.2
eth0            77.237.27.18        10.0.2.2
eth0            77.237.27.31        10.0.2.2
eth0            149.156.70.60       10.0.2.2
eth0            77.237.27.32        10.0.2.2
eth0            77.237.27.39        10.0.2.2
eth0            77.237.27.59        10.0.2.2
lo              10.0.2.15           10.0.2.15
eth0            8.8.8.8             10.0.2.2
```

Figure 33 linux_route_cache

- linux_mount

This plugins mimics of the output of /proc/mounts on a running Linux system. For each mountpoint it prints the flags, mounted source (drive, network share, etc) and the director it is mounted on.

```
enisa@ENISA-VirtualBox:~$ volatility --profile=LinuxGolfish-2_6_29ARM -f ram.lime linux_mount
Volatility Foundation Volatility Framework 2.3.1
none                    /dev/cpuctl                         cgroup      rw

none                    /acct                               cgroup      rw

/dev/block/mtdblock0    /system                             yaffs2      ro

sysfs                   /sys                                sysfs       rw

none                    /acct                               cgroup      rw

devpts                  /dev/pts                            devpts      rw

/dev/block/mtdblock0    /system                             yaffs2      ro

sysfs                   /sys                                sysfs       rw

none                    /dev/cpuctl                         cgroup      rw

/dev/block/mtdblock1    /data                               yaffs2      rw,nosuid,nodev

tmpfs                   /dev                                tmpfs       rw,nosuid

/dev/block/vold/179:0   /mnt/secure/asec/.android_secure    vfat        rw,nosuid,nodev,noexec
```

Figure 34 linux_mount

2.7.7 Using Autopsy

The Autopsy allows to analyse files extracted from an Android device. It supports physical dumps from most of Android devices (please note that in this exercise physical acquisition methods are not explained) as well as raw memory dump files.

2.7.7.1 To run Autopsy you need to start the Autopsy service as a root user.

```
enisa@ENISA-VirtualBox:~$ sudo autopsy

================================================================================

                    Autopsy Forensic Browser
                http://www.sleuthkit.org/autopsy/
                           ver 2.24

================================================================================
Evidence Locker: /var/lib/autopsy
Start Time: Mon Aug 24 14:33:24 2015
Remote Host: localhost
Local Port: 9999

Open an HTML browser on the remote host and paste this URL in it:

    http://localhost:9999/autopsy

Keep this process running and use <ctrl-c> to exit
```

Figure 35

2.7.7.2 When the service is started, open web browser and type this address: http://localhost:9999/autopsy.

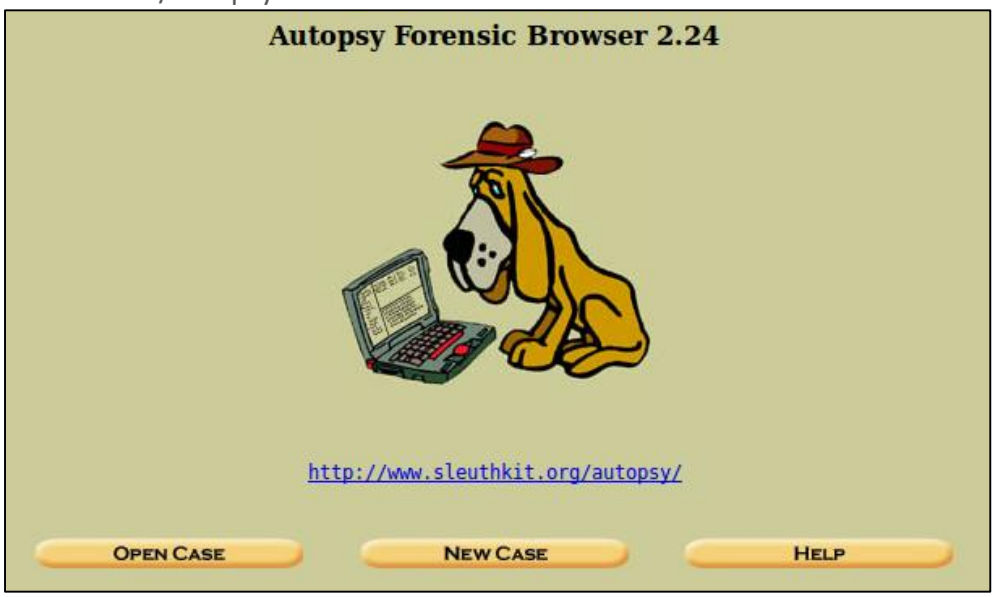

Figure 36

2.7.7.3 To create new case click on NEW CASE. To open existing one, click OPEN CASE. When you create a new case you have to fill in information such as "Case name" and "Investigator names". Description is optional.

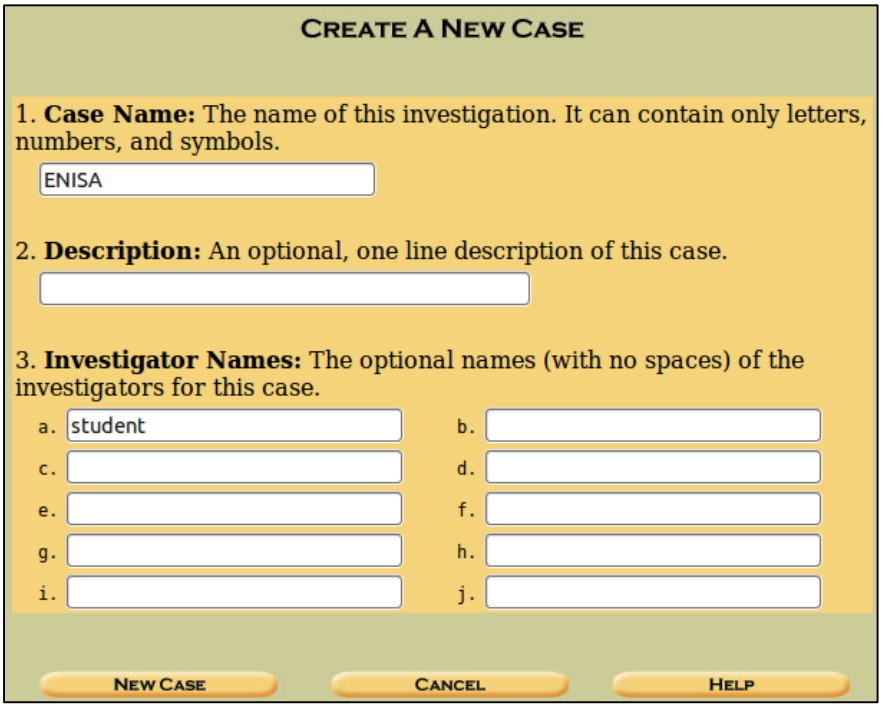

Figure 37

2.7.7.4 Next you'll be prompted to add a host i.e. device subject to investigation.

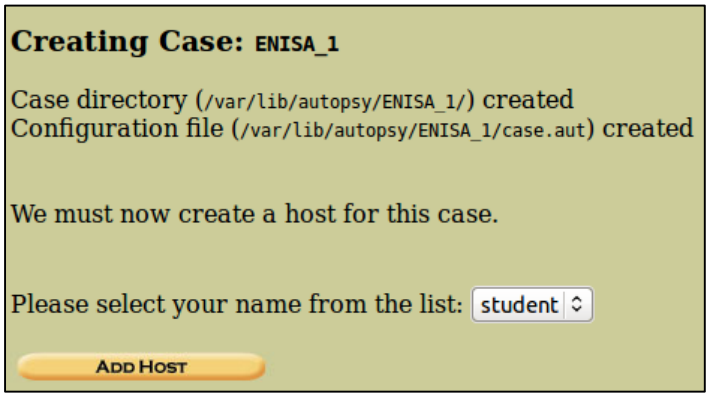

Figure 38

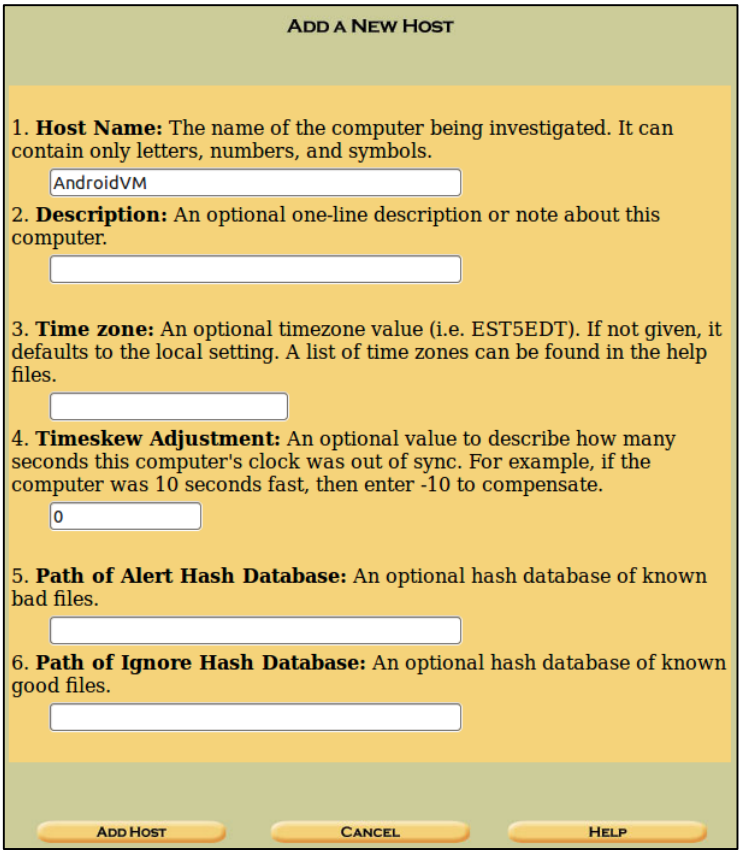

Figure 39

2.7.7.5 Next you'll be prompted to add an image of the host / device.

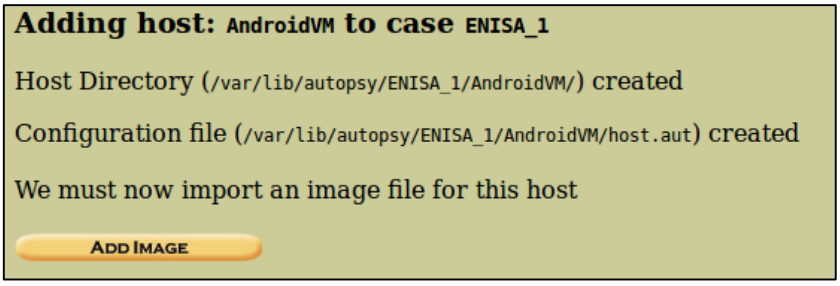

Figure 40

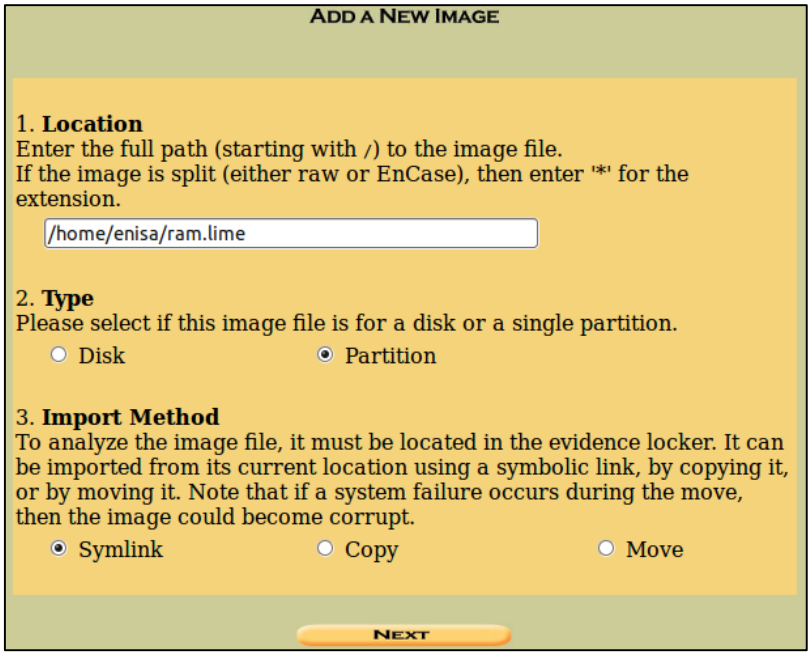

Figure 41

2.7.7.6 Next step is to choose proper file system of the added image. Choose "raw" for the RAM memory dump.

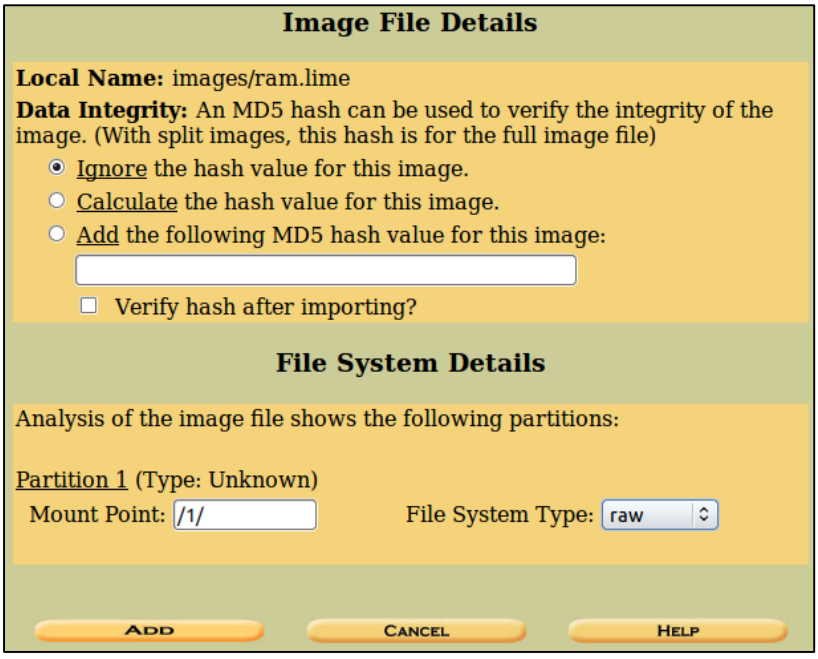

Figure 42

2.7.7.7 Once completed you'll be presented with the following screen which allows to run analysis, add another image file, close the file or – among other options – check image's integrity.

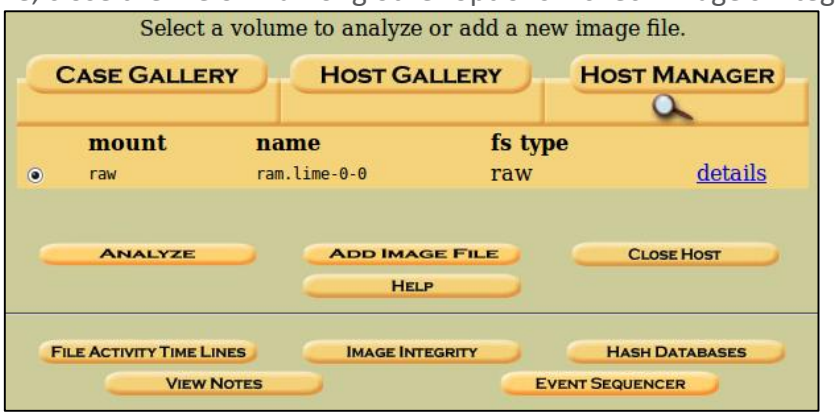

Figure 43

2.7.7.8 Since the image subject to analysis if a raw image, some functionalities may not be available. As an example, try and search for keyword "@enisa.europa.eu" which – in this case – was used for the e-mail address set up on the Android system. Try to locate e-mail account's password.

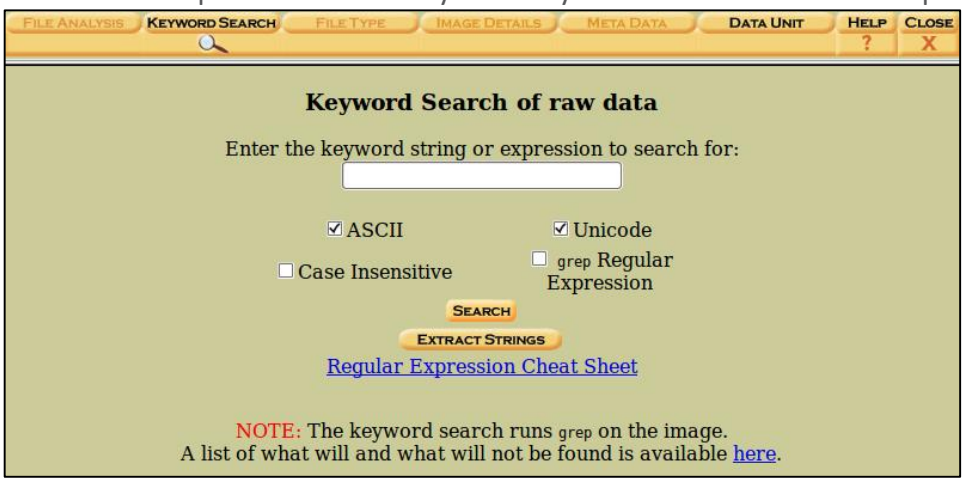

Figure 44

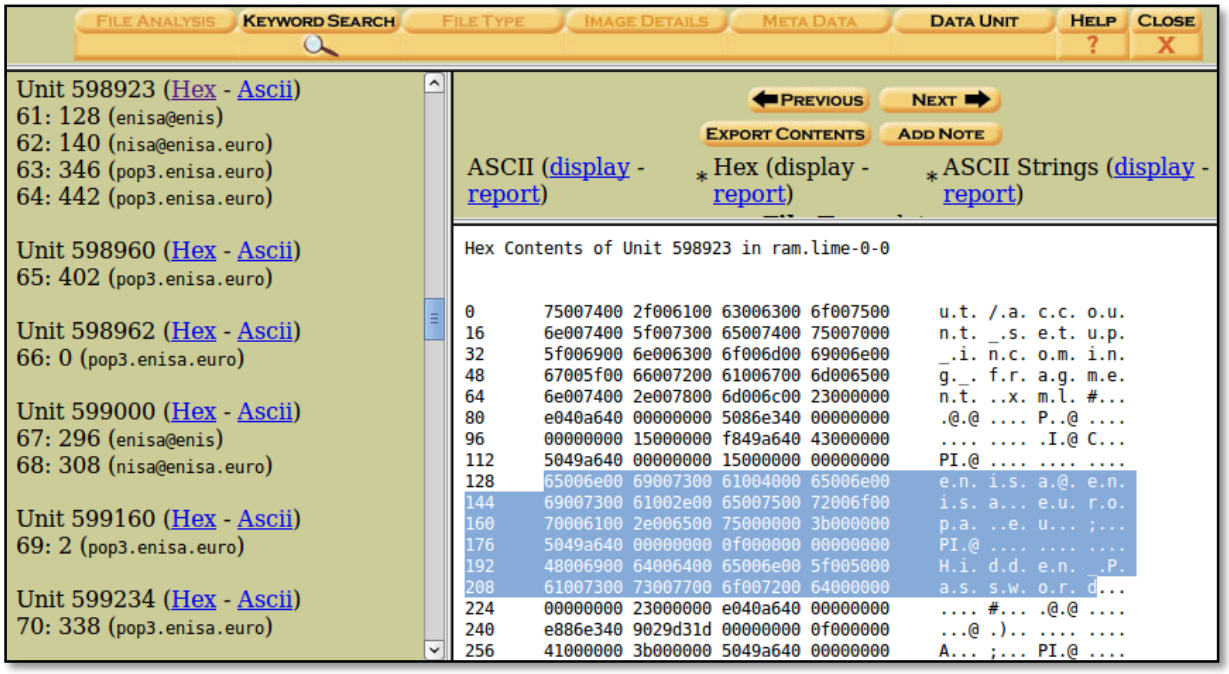

Figure 45

2.8 Task 4.5: iOS – iPhone Backup Analyser 2

2.8.1 Introduction

In this task, the students are given a set of files from iTunes backup system for analysis with iPhone Backup Analyser 2 (iPBA2). This software allows the user to browse through the content of an iPhone/iPad backup made by iTunes (or other software able to perform iOS devices' backup).

2.8.2 Details

As explained on iPBA2 project website, it parses the backup directory and shows decoded file system tree. Each file can be clicked to see its properties, such as:

- Real name and name in the backup directory
- File UNIX permissions
- Data hash (as calculated by iOS)
- User and group ID
- Modify time, access time, creation time
- File type (from magic numbers)

Any built-in viewer will allow to browse through known file formats e.g.:

- ASCII viewer
- PLIST structure browser
- SQLITE browser
- HEX viewer

2.8.3 Task walk-through

This section will explain possible ways to analysis of data stored in iOS backup.

2.8.3.1 Open iPBA2 and open folder ef2662a6b74953ed19d5aa3c25cfcd0019ed43ee.

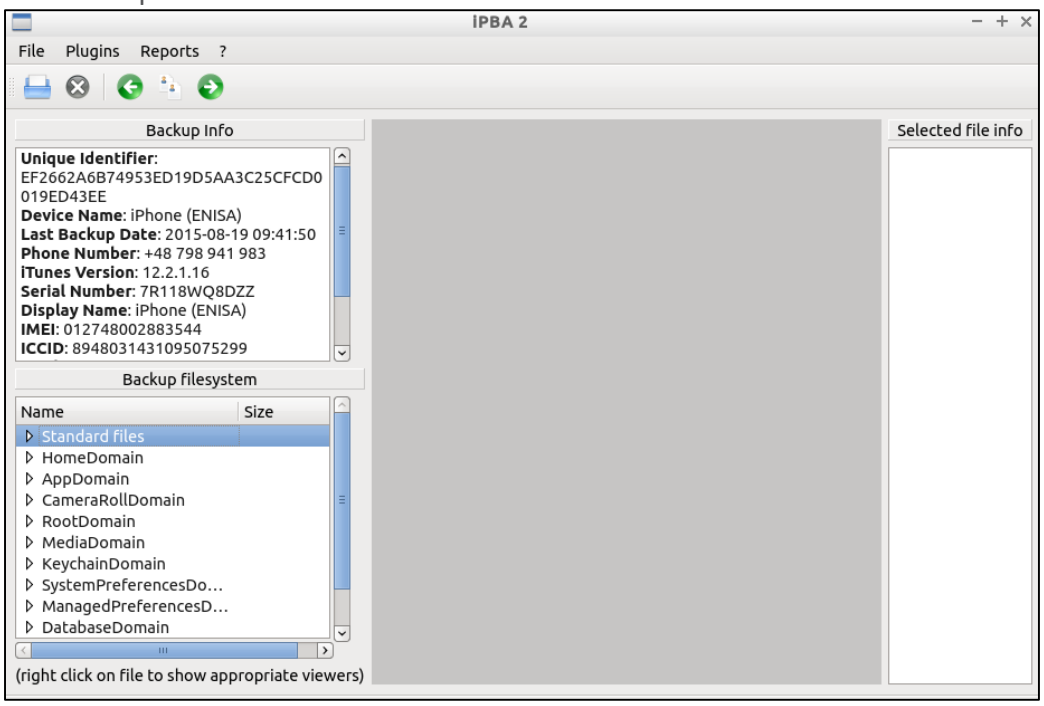

Figure 46

2.8.3.2 You can use predefined PLUGINS to view some data and create reports from predefined REPORTS tools.

2.8.3.3 Take a look at the following option: CameraRollDomain -> Media/DCIM/100APPLE.

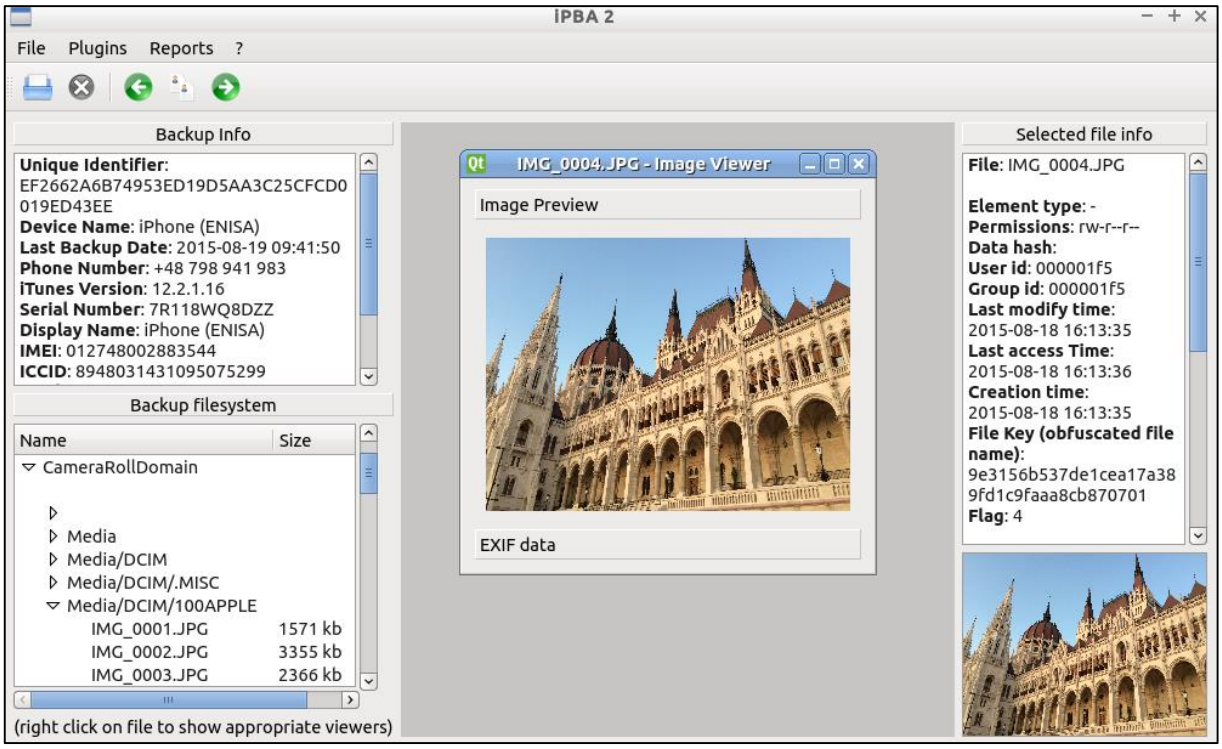

Figure 47

2.8.3.4 You can view or even export simple JPG file from backup. Please export few JPG files from backup.

2.8.3.5 Let's take a look of JPG files which you exported by another tool called exiftool. This is a software which allows to read EXIF data. Try to locate GPS co-ordinates.

```
enisa@ENISA-VirtualBox:~/Documents$ exiftool IMG_0001.JPG
ExifTool Version Number         : 9.46
File Name                       : IMG_0001.JPG
Directory                       : .
File Size                       : 1572 kB
File Modification Date/Time     : 2015:08:24 16:39:56+02:00
File Access Date/Time           : 2015:08:24 16:39:59+02:00
File Inode Change Date/Time     : 2015:08:24 16:39:56+02:00
File Permissions                : rw-rw-r--
File Type                       : JPEG
MIME Type                       : image/jpeg
GPS Altitude                    : 116.3 m Above Sea Level
GPS Date/Time                   : 2015:08:06 22:09:05Z
GPS Latitude                    : 47 deg 29' 55.17" N
GPS Longitude                   : 19 deg 3' 36.38" E
GPS Position                    : 47 deg 29' 55.17" N, 19 deg 3' 36.38" E
Image Size                      : 3264x2448
```

Figure 48

2.8.3.6 Now you are able to see GPS position where this image file was most presumably taken. Open any website that allows to search GPS position (e.g. https://maps.google.com) and the co-ordinates into search field. You need to change word "deg" to a degree symbol (i.e. °) with a combination of ALT + 248 on keyboard.

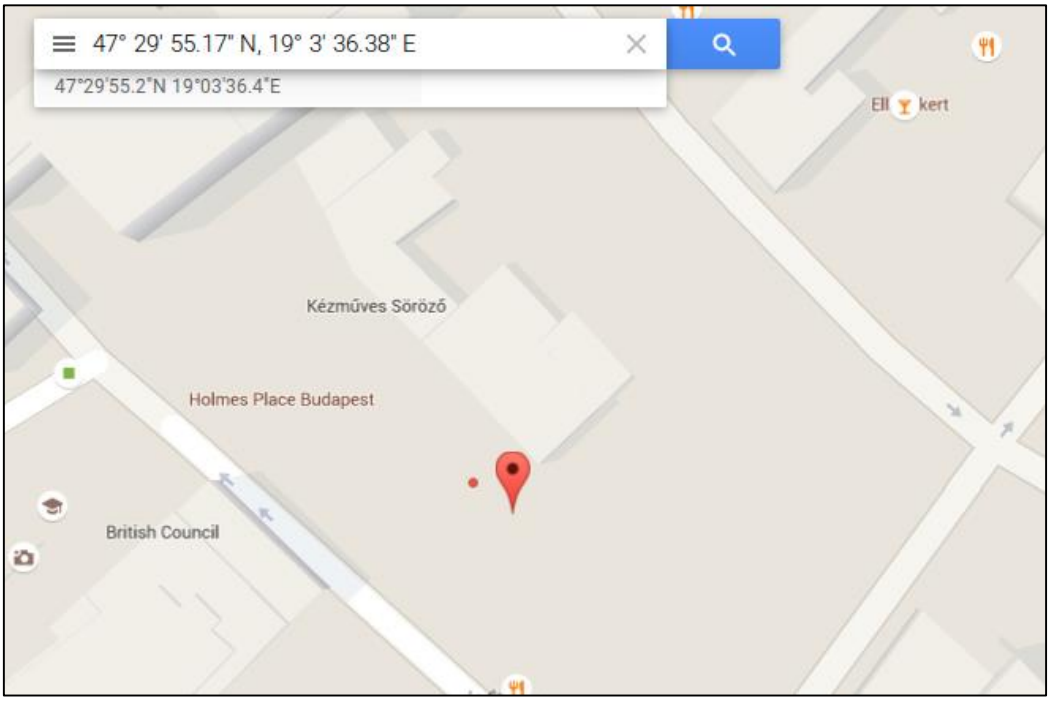

Figure 49

2.8.3.7 Now take a look into AppDomain. This is a list of installed applications. Locate folder Documents for application WhatsApp (net.wshatsapp.WhatsApp). Here you can find an SQLite databases used by WhatsApp instant messenger. Open database called ChatStorage.sqlite in SQLite viewer. Try to find ZWAMESSAGE table which contains messages history.

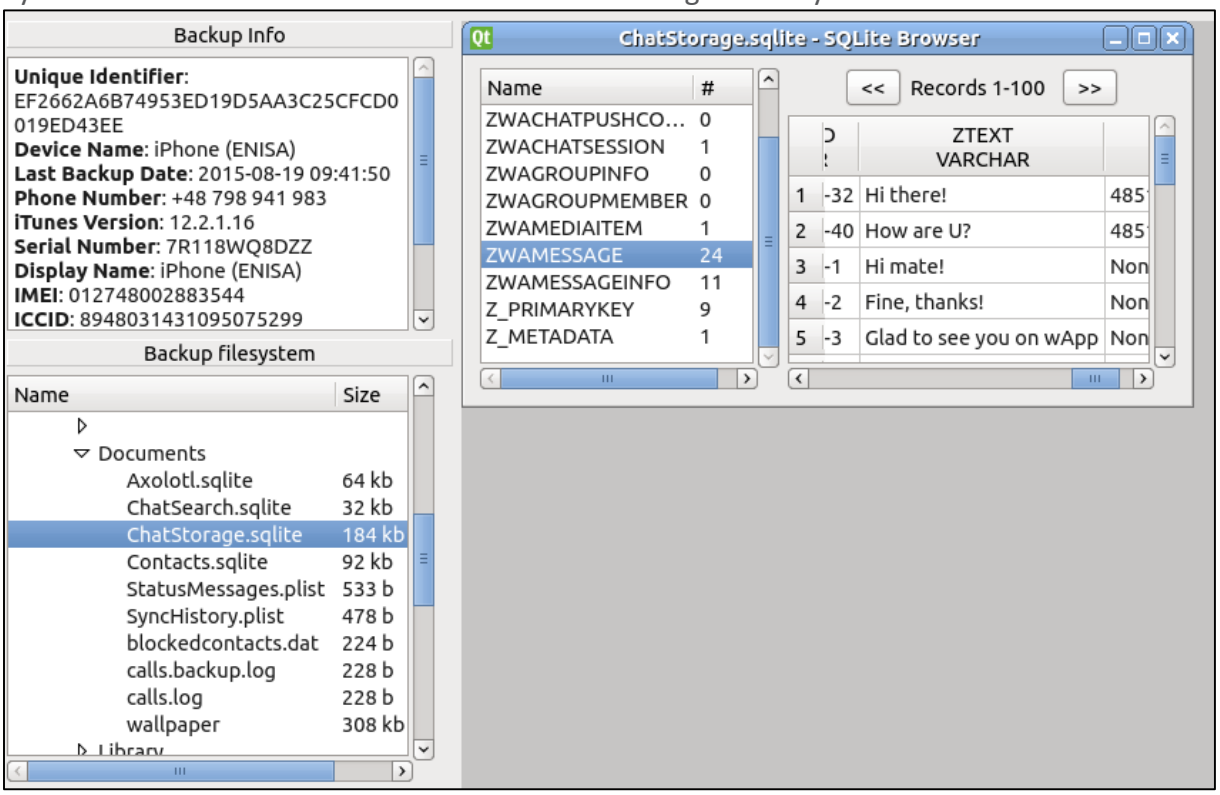

Figure 50

2.9 Task 4.6: Brute-forcing Android encryption mechanisms

2.9.1 Introduction

In this task, the students will try the process of cracking the PIN used to encrypt Android device (Ice Cream Sandwich and Jelly Bean) using brute force methods.

2.9.2 Details

Students will have to prepare a recovery partition for Android device. After that they will be able to download necessary files which will be used during cracking process.

2.9.3 Task walk-through

This section will show possible approaches to brute-force attacks against PIN-based encryption of Android devices. The task requires students to use a physical device, since AVD emulators won't support fastboot mode.

2.9.3.1 First you need to put the phone in recovery mode so that it can boot a custom recovery image. If the device already has a custom recovery image with root address installed, you can skip this step. There's and easy way to accomplish this with adb. If you have a device with adb enabled, simply connect it to PC (make sure you pass it through if running in a virtual machine), run a terminal and issue the following command:

```
enisa@ENISA-VirtualBox:~$ adb reboot bootloader
```

Figure 51

2.9.3.2 Next, boot the device from a rooted recovery image. For this purpose, a Clockwork Mod[17] image is used, but you can use any device-compatible recovery image with root and adb enabled. Please, note where you saved the recovery image. From a terminal, run fastboot and make sure you can communicate with the device.

```
enisa@ENISA-VirtualBox:~$ fastboot devices

???????????             fastboot
```

Figure 52

2.9.3.3 Now that you checked that you can communicate with the device over fastboot, boot it using the recovery image.

```
enisa@ENISA-VirtualBox:~$ fastboot boot ~/Downloads/recovery-clockwork-6.0.4.3-c
respo4g.img
< waiting for device >
downloading 'boot.img'... OKAY
booting... OKAY
```

Figure 53

2.9.3.4 Your device should be in a recovery mode now. Next, pull the needed necessary header and footer data so that you can brute-force the encryption PIN. Their location varies device by device so choose the steps for your particular device type.

```
enisa@ENISA-VirtualBox:~$ adb shell dd if=/dev/block/mmcblk0p2 of=tmp_header bs=
512 count=1
enisa@ENISA-VirtualBox:~$ adb pull tmp header ~/Desktop/tmp header
enisa@ENISA-VirtualBox:~$ adb shell dd if=/dev/block/mmcblk0p13 of=tmp footer
enisa@ENISA-VirtualBox:~$ adb pull tmp_footer ~/Desktop/tmp_footer
```

Figure 54

[17] ROM Manager: ROMs and Recovery Images, http://www.clockworkmod.com/rommanage, last accessed on: 2015-09-14

2.9.3.5 Now that you have everything you need you'll run the Android Brute Force Encryption cracking program against the header and footer files. By default, a 4-digit numeric passcodes will be used but you can change the number of digits at your will remembering that the longer the PIN, the more time is necessary to brute-force it.

```
enisa@ENISA-VirtualBox:~$ bruteforce_stdcrypto ~/Desktop/tmp_header ~/Desktop/tm
p_footer
Defaulting max PIN digits to 4
Footer File    : /home/enisa/Desktop/tmp_footer
Magic          : 0xD0B5B1C4
Major Version  : 1
Minor Version  : 0
Footer Size    : 104 bytes
Flags          : 0x00000000
Key Size       : 128 bits
Failed Decrypts: 0
Crypto Type    : aes-cbc-essiv:sha256
Encrypted Key  : 0xE51861649D0005F874AD6CCAB6DF2C61
Salt           : 0xA163525990AC7A053E1E372914999BE8
---------------
Trying to Bruteforce Password... please wait
Trying: 0000
Trying: 0001
Trying: 0002
Trying: 0003

Found PIN!: 1309
```

Figure 55

After a while you will be able to see PIN code.

2.10 Task 5.1: Analysing pcap data and proxy logs of Android.Trojan.SLocker.DZ

2.10.1 Introduction

In this task the students are given a set of files (PCAP, mitmproxy) created during observing activity by an Android device infected with the Android.Trojan.Slocker.DZ ransomware. The students will use Wireshark and the text editor of their choice to search for patterns indicating malicious behaviour and analyse and describe this.

2.10.2 Tools used

- Wireshark
- MITMProxy

2.10.3 Details

There are two files in the traces subdirectory of the exercise environment named:

F836F5C6267F13BF9F6109A6B8D79175.pcap and F836F5C6267F13BF9F6109A6B8D79175.log.

2.10.4 Task walk-through

This section will contain possible ways to analyse the given information and identify the answers to the requests in the table:

2.10.4.1 Open the PCAP file in Wireshark. After loading the PCAP file you will see the list of packets captured. Only a subset of these packets are part of the malware communication, to identify these we can use some of the tools Wireshark provides.

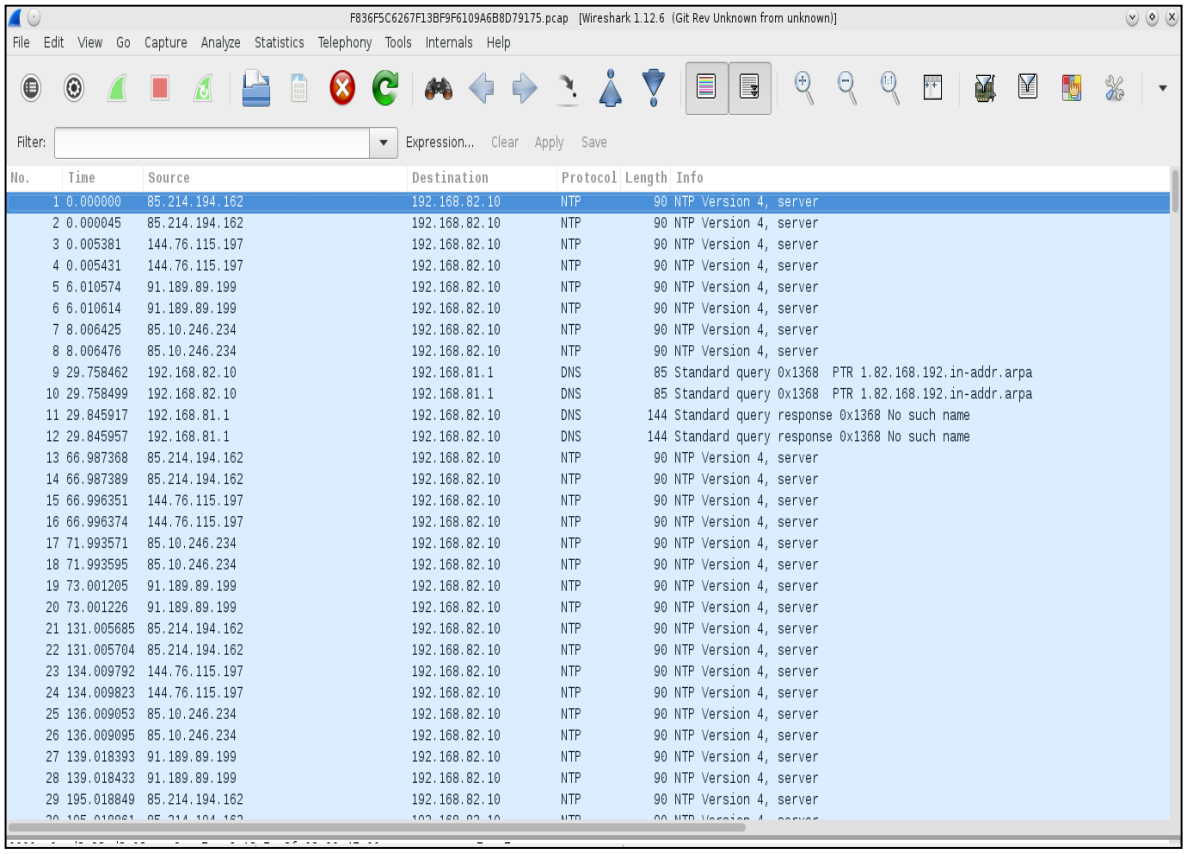

Figure 56

2.10.4.2 Open the list of conversations (Statistics → Conversations → Tab IPv4). In this list you will find all conversations contained in the network traffic dump accompanied by additional information regarding starting point in time, amount of data transferred and duration of the connection. This does not show the malicious traffic by itself but delivers an overview and some details regarding the information to be analysed.

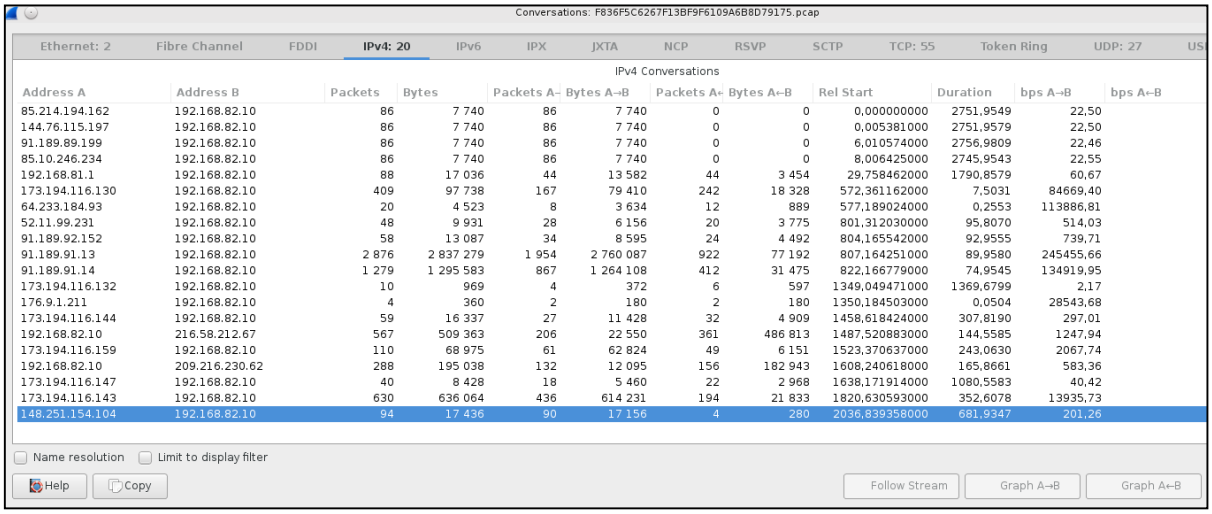

Figure 57

2.10.4.3 Open the list of endpoints (Statistics → Endpoints → Tab IPv4). In this list the geo-location of the conversation endpoints is added to the list.

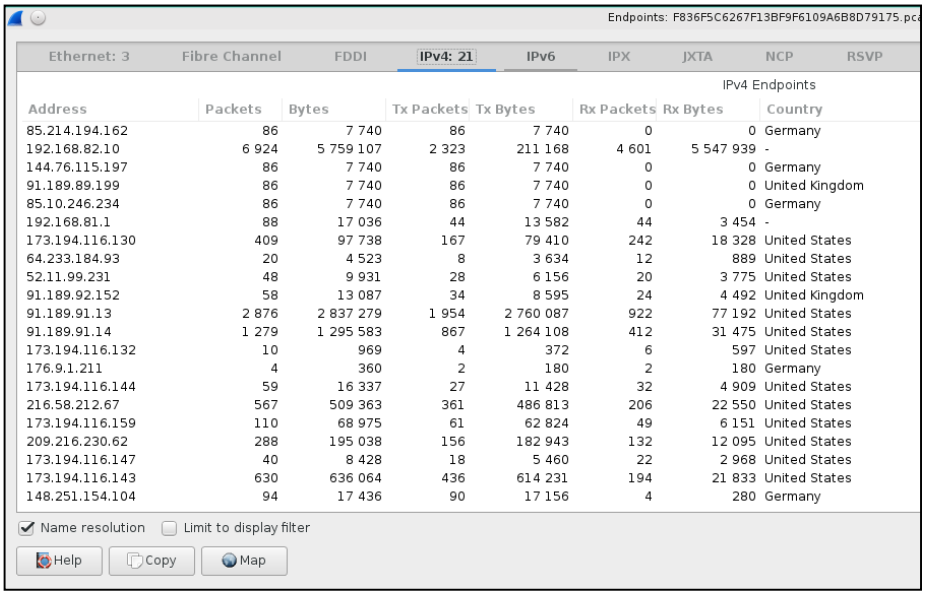

Figure 58

2.10.4.4 Open the Protocol Hierarchy (Statistics → Protocol Hierarchy). The previous approaches have given no clear indicator for malicious behaviour or even a hint for which connections should be inspected in more detail. Thus you have to try to dig deeper and find information regarding the protocols used in the dump. In this case you'll find interesting information regarding JSON data which has been transmitted in clear-text.

▾ Hypertext Transfer Protocol	4,85 %		336	2,28 %
Line-based text data	1,21 %		84	0,71 %
Media Type	0,42 %		29	0,43 %
JavaScript Object Notation	0,38 %		26	0,13 %
Malformed Packet	0,07 %		5	0,07 %

Figure 59

2.10.4.5 Apply a filter to the captured traffic (Right-click on the selected entry → Apply as Filter → Selected).

2.10.4.6 Use TCP stream analysis (Right-click → Follow TCP Stream). Using this feature provides a human-readable presentation of the HTTP traffic. Indicators of malicious traffic can be clearly identified, for example the deception phrase in the payload of the first server response.

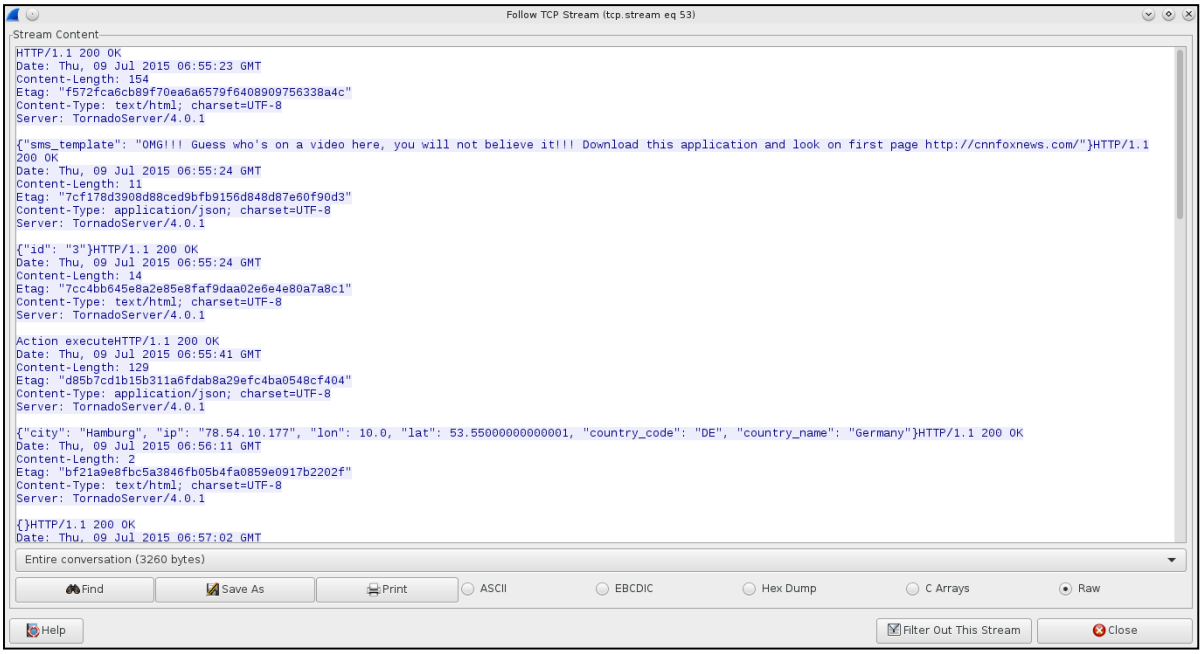

Figure 60

2.10.5 Task walk-through with mitmproxy logs

For the creation of the following screenshots Honeyproxy[18] was used. The project is based on MITMProxy and creates a web interface to inspect and analyse the traffic captured. Unfortunately it is not under active development as of the time of preparation of this exercise.

[18] HoneyProxy — a man-in-the-middle SSL proxy & traffic analyser, http://honeyproxy.org/, last accessed on: 2015-09-14

2.10.5.1 Overview of the web interface and the captured data.

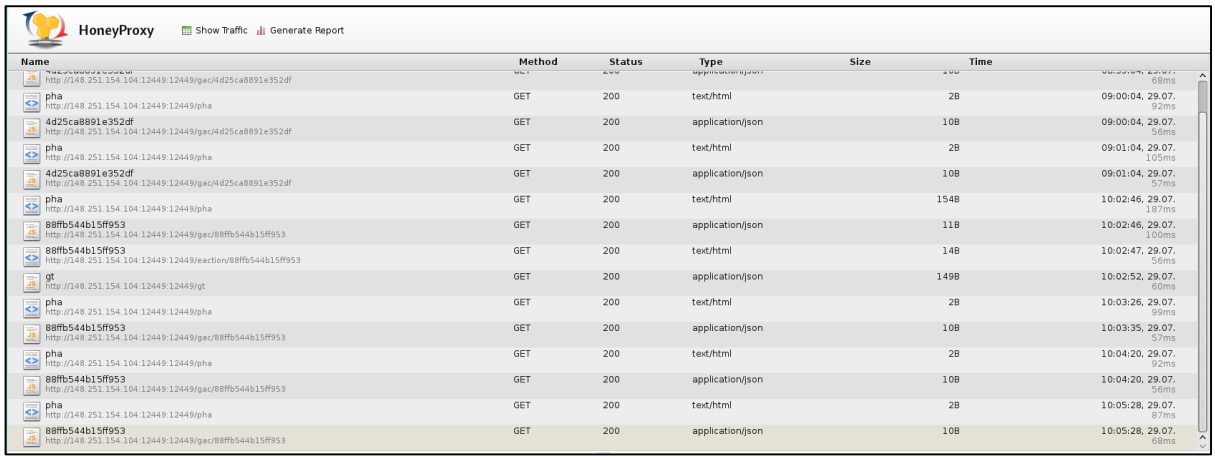

Figure 61

2.10.5.2 Close view of the malware requests.

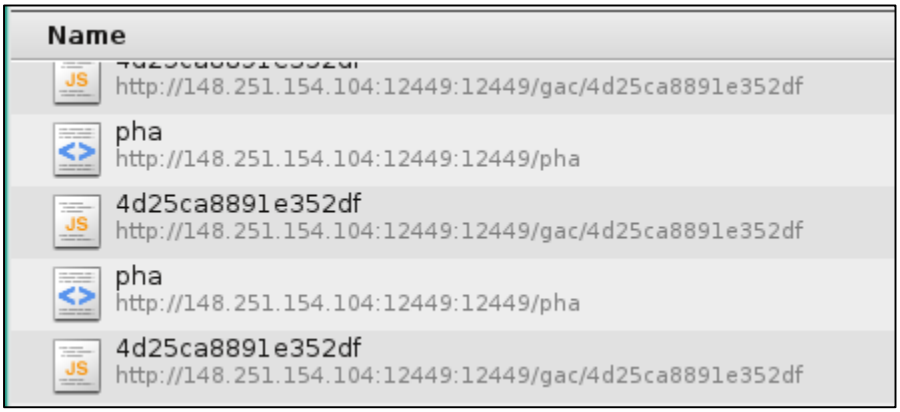

Figure 62

2.10.5.3 Request showing the message displayed to the victim.

Figure 63

2.10.5.4 Request showing the transmitted information.

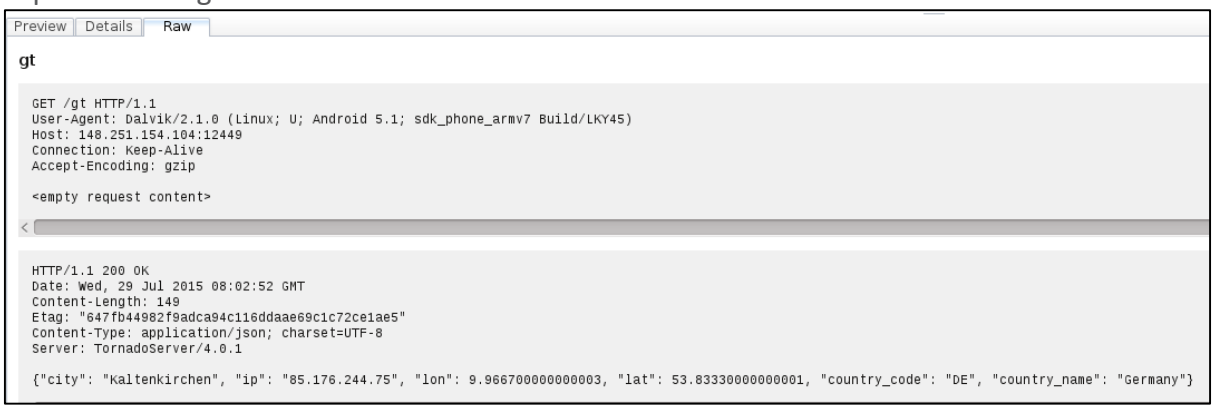

Figure 64

2.11 Task 5.2: Analysing pcap data and proxy logs of iOS.Oneclickfraud

2.11.1 Introduction

In this task the students are given a set of files (PCAP, mitmproxy) created during observing activity by an iOS device infected with the iOS.Oneclickfraud malware. The students will use Wireshark and the text editor of their choice to search for patterns indicating malicious behaviour and analyse and describe these.

2.11.2 Tools

- Wireshark

2.11.3 Details

There are two files in the traces subdirectory of the exercise environment named:

71972F763EB5EAEB87681D2615E9E68E.pcap and 71972F763EB5EAEB87681D2615E9E68E.log.

2.11.4 Test walk-through

The general approach to identify the malign traffic in the PCAP file is identical to Task 5.1. Following we will show the screenshots unique to Task 5.2. There is no walk-through with proxy logs as the server is not responding to the malware requests.

2.11.4.1 List of conversations.

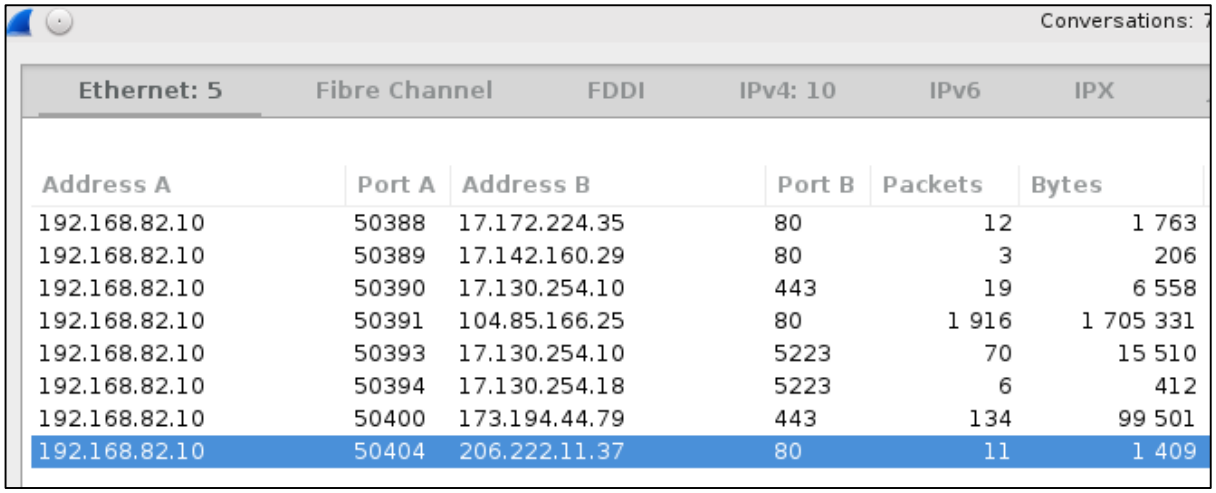

Figure 65

2.11.4.2 List of endpoints.

Endpoints: 71972F763EB5EAEB87681D2615E9E68E.pca

Ethernet: 6 | Fibre Channel | FDDI | IPv4: 11 | IPv6 | IPX | JXTA | NCP | RSVP

TCP Endpoints

Address	Port	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Country
192.168.82.10	50388	12	1 763	7	644	5	1 119	-
17.172.224.35	80	12	1 763	5	1 119	7	644	United States
192.168.82.10	50389	3	206	2	132	1	74	-
17.142.160.29	80	3	206	1	74	2	132	United States
192.168.82.10	50390	19	6 558	10	838	9	5 720	-
17.130.254.10	443	19	6 558	9	5 720	10	838	United States
192.168.82.10	50391	1 916	1 705 331	820	55 126	1 096	1 650 205	-
104.85.166.25	80	1 916	1 705 331	1 096	1 650 205	820	55 126	-
192.168.82.10	50393	70	15 510	36	6 558	34	8 952	-
17.130.254.10	5223	70	15 510	34	8 952	36	6 558	United States
192.168.82.10	50394	6	412	4	264	2	148	-
17.130.254.18	5223	6	412	2	148	4	264	United States
192.168.82.10	50400	134	99 501	59	8 148	75	91 353	-
173.194.44.79	443	134	99 501	75	91 353	59	8 148	United States
192.168.82.10	50404	11	1 409	6	607	5	802	-
206.222.11.37	80	11	1 409	5	802	6	607	United States

Figure 66

2.11.4.3 TCP Stream.

Follow TCP Stream (tcp.stream eq 7)

Stream Content

```
GET /app/init HTTP/1.1
Host: eroeroou.com
Accept: */*
Accept-Language: en-us
Connection: keep-alive
Accept-Encoding: gzip, deflate
User-Agent: EroEroMovie/1 CFNetwork/711.1.12 Darwin/14.0.0

HTTP/1.1 404 Not Found
Date: Wed, 08 Jul 2015 10:36:41 GMT
Server: Apache/2.2.15 (CentOS)
Content-Length: 284
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /app/init was not found on this server.</p>
<hr>
<address>Apache/2.2.15 (CentOS) Server at eroeroou.com Port 80</address>
</body></html>
```

Figure 67

2.12 Task 6.1: Analysing Android.Trojan.SLocker.DZ

2.12.1 Introduction

In this task the students will analyse an Android.Trojan.SLocker.DZ APK file. They will have to answer a couple of questions which will lead them to the identification of various characteristics of this trojan malware.

2.12.2 Tools

- AndroGuard
- apktool

2.12.3 Details

In the exercise directory students will find an APK file F836F5C6267F13BF9F6109A6B8D79175.apk. For the analysis of this file they can use the pre-installed AndroGuard, apktool and the text editor of their choice (there are several available on the system). In the next section students will find questions they have to answer during the analysis to identify the behaviour of the application.

2.12.4 Task walk-through

In this section, we will walk through a possible approach to analyse the malware and extract requested information.

2.12.4.1 Decode the APK file with the following command:

apktool d F836F5C6267F13BF9F6109A6B8D79175.apk -o F836F5C6267F13BF9F6109A6B8D79175

```
ENISA: apktool d F836F5C6267F13BF9F6109A6B8D79175.apk -o F836F5C6267F13BF9F6109A6B8D79175
I: Using Apktool 2.0.1 on F836F5C6267F13BF9F6109A6B8D79175.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/mirko/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
ENISA:
```

Figure 68

2.12.4.2 Search for permissions in the AndroidManifest:

grep permission F836F5C6267F13BF9F6109A6B8D79175/AndroidManifest.xml

```
ENISA: ls -l F836F5C6267F13BF9F6109A6B8D79175
total 24
-rw-r--r--  1 mirko users 2823 Jul 21 09:55 AndroidManifest.xml
-rw-r--r--  1 mirko users  310 Jul 21 09:55 apktool.yml
drwxr-xr-x  2 mirko users 4096 Jul 21 09:55 assets
drwxr-xr-x  3 mirko users 4096 Jul 21 09:55 original
drwxr-xr-x 11 mirko users 4096 Jul 21 09:55 res
drwxr-xr-x  4 mirko users 4096 Jul 21 09:55 smali
ENISA:
```

Figure 69

```
ENISA: grep permission F836F5C6267F13BF9F6109A6B8D79175/AndroidManifest.xml
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS"/>
ENISA:
```

Figure 70

2.12.4.3 Search for the package name in the AndroidManifest:

grep package F836F5C6267F13BF9F6109A6B8D79175/AndroidManifest.xml

```
ENISA: grep package F836F5C6267F13BF9F6109A6B8D79175/AndroidManifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.adobe.videoprayer" platformBuildVersionCode="19" platformBuildVersionName="4.4.2-1456859">
ENISA:
```

Figure 71

2.12.4.4 Search for the intents in the AndroidManifest.

```
ENISA: grep -A1 intent AndroidManifest.xml
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
--
            <intent-filter android:priority="2147483647">
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
            </intent-filter>
        </receiver>
--
            <intent-filter android:priority="2147483647">
                <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
            </intent-filter>
        </receiver>
```

Figure 72

2.12.4.5 Control the assets directory.

```
ENISA: ls -l F836F5C6267F13BF9F6109A6B8D79175/
total 24
-rw-r--r--  1 mirko users 2823 Jul 21 09:55 AndroidManifest.xml
-rw-r--r--  1 mirko users  310 Jul 21 09:55 apktool.yml
drwxr-xr-x  2 mirko users 4096 Jul 21 09:55 assets
drwxr-xr-x  3 mirko users 4096 Jul 21 09:55 original
drwxr-xr-x 11 mirko users 4096 Jul 21 09:55 res
drwxr-xr-x  4 mirko users 4096 Jul 21 09:55 smali
ENISA: ls -l F836F5C6267F13BF9F6109A6B8D79175/assets/
total 272
-rw-r--r-- 1 mirko users  73468 Jul 21 09:55 320x320_prism-logo.png
-rw-r--r-- 1 mirko users 142910 Jul 21 09:55 320x320_stamp and sign.png
-rw-r--r-- 1 mirko users  15798 Jul 21 09:55 accordion_closed.png
-rw-r--r-- 1 mirko users   1348 Jul 21 09:55 accordion.css
-rw-r--r-- 1 mirko users    430 Jul 21 09:55 accordion.js
-rw-r--r-- 1 mirko users  15814 Jul 21 09:55 accordion_open.png
-rw-r--r-- 1 mirko users   2997 Jul 21 09:55 tab1.html
-rw-r--r-- 1 mirko users   9497 Jul 21 09:55 tab2.html
-rw-r--r-- 1 mirko users   1918 Jul 21 09:55 tab4.html
```

Figure 73

2.12.4.6 View the contents of the HTML file:

w3m F836F5C6267F13BF9F6109A6B8D79175/assets/tab1.html

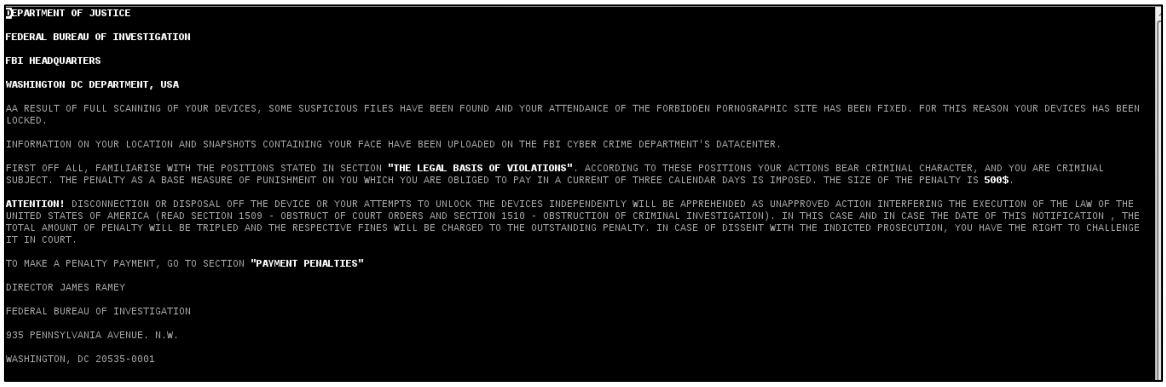

Figure 74

2.12.4.7 Search for IP addresses in the dataset:

*grep -Eor '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' F836F5C6267F13BF9F6109A6B8D79175/**

```
ENISA: grep -Eor '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' F836F5C6267F13BF9F6109A6B8D79175/*
F836F5C6267F13BF9F6109A6B8D79175/res/values/strings.xml:192.168.2.81
F836F5C6267F13BF9F6109A6B8D79175/smali/com/adobe/videoprayer/net/req/AbsRequest.smali:148.251.154.104
```

Figure 75

2.12.4.8 Inspect the strings.xml file:

```
less F836F5C6267F13BF9F6109A6B8D79175/res/values/strings.xml
```

Figure 76

2.13 Task 6.2: Analysing iOS.Oneclickfraud

2.13.1 Introduction

In this task the students will use class-dump-z to analyse iOS.Oneclickfraud. As in Task 6.1 they will have to answer some questions regarding the characteristics. The trainer will give a short introduction into the usage of the disassembler.

2.13.2 Tools

- class-dump-z

2.13.3 Details

In the exercise directory, students will find an iOS application file 71972F763EB5EAEB87681D2615E9E68E. For the analysis of this file they will have to use the pre-installed class-dump-z disassembler. In the next section they will find questions they have to answer during the analysis to identify the behaviour of the application.

2.13.4 Task walk-through

In this section a possible approach how to analyse the malware and extract the requested information will be shown.

2.13.4.1 Identify the file and unzip it:

file 71972F763EB5EAEB87681D2615E9E68E

unzip 71972F763EB5EAEB87681D2615E9E68E

```
ENISA: file 71972F763EB5EAEB87681D2615E9E68E
71972F763EB5EAEB87681D2615E9E68E: Zip archive data, at least v1.0 to extract Zip archive data, at least v1.0 to extract
ENISA: unzip 71972F763EB5EAEB87681D2615E9E68E
Archive:  71972F763EB5EAEB87681D2615E9E68E
   creating: Payload/
   creating: Payload/EroEroMovie.app/
   creating: Payload/EroEroMovie.app/_CodeSignature/
  inflating: Payload/EroEroMovie.app/_CodeSignature/CodeResources
  inflating: Payload/EroEroMovie.app/archived-expanded-entitlements.xcent
   creating: Payload/EroEroMovie.app/Base.lproj/
  inflating: Payload/EroEroMovie.app/Base.lproj/LaunchScreen.nib
  inflating: Payload/EroEroMovie.app/embedded.mobileprovision
  inflating: Payload/EroEroMovie.app/EroEroMovie
   creating: Payload/EroEroMovie.app/Frameworks/
  inflating: Payload/EroEroMovie.app/Frameworks/libswiftCore.dylib
  inflating: Payload/EroEroMovie.app/Frameworks/libswiftCoreGraphics.dylib
  inflating: Payload/EroEroMovie.app/Frameworks/libswiftCoreImage.dylib
  inflating: Payload/EroEroMovie.app/Frameworks/libswiftDarwin.dylib
  inflating: Payload/EroEroMovie.app/Frameworks/libswiftDispatch.dylib
  inflating: Payload/EroEroMovie.app/Frameworks/libswiftFoundation.dylib
  inflating: Payload/EroEroMovie.app/Frameworks/libswiftObjectiveC.dylib
  inflating: Payload/EroEroMovie.app/Frameworks/libswiftSecurity.dylib
  inflating: Payload/EroEroMovie.app/Frameworks/libswiftUIKit.dylib
  inflating: Payload/EroEroMovie.app/Info.plist
  inflating: Payload/EroEroMovie.app/PkgInfo
   creating: SwiftSupport/
  inflating: SwiftSupport/libswiftCore.dylib
  inflating: SwiftSupport/libswiftCoreGraphics.dylib
  inflating: SwiftSupport/libswiftCoreImage.dylib
  inflating: SwiftSupport/libswiftDarwin.dylib
  inflating: SwiftSupport/libswiftDispatch.dylib
  inflating: SwiftSupport/libswiftFoundation.dylib
  inflating: SwiftSupport/libswiftObjectiveC.dylib
  inflating: SwiftSupport/libswiftSecurity.dylib
  inflating: SwiftSupport/libswiftUIKit.dylib
ENISA:
```

Figure 77

2.13.4.2 Use strings to gather information:

strings -a Payload/EroEroMovie.app/embedded.mobileprovision

```
ENISA: strings -a Payload/EroEroMovie.app/embedded.mobileprovision |grep -i -A 1 certification
Apple Certification Authority1
Apple Root CA0
--
Apple Certification Authority1-0+
$Apple iPhone Certification Authority0
!p(E`\
--
Apple Certification Authority1-0+
$Apple iPhone Certification Authority0
080521020415Z
--
Reliance on this certificate by any party assumes acceptance of the then applicable standard terms and conditions of use, certificate policy and certification practice statements.0D
=0;09
--
Apple Certification Authority1-0+
$Apple iPhone Certification Authority
150514110826Z0#
```

Figure 78

```
ENISA: strings -a Payload/EroEroMovie.app/embedded.mobileprovision |grep -i TeamName -A 1
        <key>TeamName</key>
        <string>Neon Way Co.,Ltd</string>
ENISA:
```

Figure 79

```
ENISA: strings -a Payload/EroEroMovie.app/embedded.mobileprovision |grep -i date
        <key>CreationDate</key>
        <date>2015-05-14T11:08:26Z</date>
        <key>ExpirationDate</key>
        <date>2016-05-13T11:08:26Z</date>
ENISA:
```

Figure 80

3. References

- Build System Overview
  http://developer.android.com/sdk/installing/studio-build.html
- Android applications permissions
  http://developer.android.com/preview/features/runtime-permissions.html
- Universal Binaries and 32-bit/64-bit PowerPC Binaries
  https://developer.apple.com/library/mac/documentation/DeveloperTools/Conceptual/MachORuntime/index.html#//apple_ref/c/tag/fat_header
- File signatures table
  http://www.garykessler.net/library/file_sigs.html
- Android SDK
  http://developer.android.com/sdk/index.html
- Android NDK
  http://developer.android.com/tools/sdk/ndk/index.html
- LiME: Linux Memory Extractor
  https://github.com/504ensicslabs/lime
- Libdwarf and Dwarfdump
  http://wiki.dwarfstd.org/index.php?title=Libdwarf_And_Dwarfdump
- Build a Volatility Profile
  https://code.google.com/p/volatility/wiki/AndroidMemoryForensics#Build_a_Volatility_Profile
- Volatility: RAM dump analyser
  https://code.google.com/p/volatility/wiki/
- Autopsy® is a digital forensics platform and graphical interface to The Sleuth Kit® and other digital forensics tools
  http://www.sleuthkit.org/autopsy/
- LiME: Linux Memory Extractor
  https://github.com/504ensicslabs/lime
- Volatility: A command reference for Linux
  https://code.google.com/p/volatility/wiki/LinuxCommandReference23
- ROM Manager: ROMs and Recovery Images
  http://www.clockworkmod.com/rommanage
- HoneyProxy: a man-in-the-middle SSL proxy & traffic analyser
  http://honeyproxy.org/

ENISA

European Union Agency for Network
and Information Security
Science and Technology Park of Crete (ITE)
Vassilika Vouton, 700 13, Heraklion, Greece

Athens Office

1 Vass. Sofias & Meg. Alexandrou
Marousi 151 24, Athens, Greece

PO Box 1309, 710 01 Heraklion, Greece
Tel: +30 28 14 40 9710
info@enisa.europa.eu
www.enisa.europa.eu