Mobile Malware

Topic

- Mobile Malware Overview
- Mobile Malware Behavior
- Static Analysis on Mobile malware
- Dynamic Analysis on Mobile malware
- Mobile Malware Mitigation

Mobile Malware Overview

Overview of Mobile Devices

- Mobile *computers*:
  - Mainly smartphones, tablets
  - Sensors: GPS, camera, accelerometer, etc.
  - Computation: powerful CPUs (≥ 1 GHz, multi-core)
  - Communication: cellular/4G, Wi-Fi, near field communication (NFC), etc.
- Many connect to cellular networks: *billing system*
- Cisco: 7 billion mobile devices will have been sold by 2012 [1]

- Global sales of smartphones to end users totaled 349 million units in the first quarter of 2016 (gartner Q1, 2016)
- Driven by 4G connectivity promotion plans from communications service providers (CSPs) in many markets worldwide

Worldwide Smartphone Sales to End Users by Vendor in 1Q16 (Thousands of Units)

Company	1Q16 Units	1Q16 Market Share (%)	1Q15 Units	1Q15 Market Share (%)
Samsung	81,186.9	23.2	81,122.8	24.1
Apple	51,629.5	14.8	60,177.2	17.9
Huawei	28,861.0	8.3	18,111.1	5.4
Oppo	16,112.6	4.6	6,585.1	2.0
Xiaomi	15,048.0	4.3	14,740.2	4.4
Others	156,413.4	44.8	155,561.4	46.3
Total	**349,251.4**	**100.0**	**336,297.8**	**100.0**

Source: Gartner (May 2016)

- Two major Mobile OS are Android and iOS
- 78.8% are dominated by Androids and this can be a reasoned why most mobile malware are attacking Android base mobile phone

Worldwide Smartphone Sales to End Users by Operating System in 1Q16 (Thousands of Units)

Operating System	1Q16 Units	1Q16 Market Share (%)	1Q15 Units	1Q15 Market Share (%)
Android	293,771.2	84.1	254,941.9	78.8
iOS	51,629.5	14.8	60,177.2	17.9
Windows	2,399.7	0.7	8,270.8	2.5
Blackberry	659.9	0.2	1,325.4	0.4
Others	791.1	0.2	1,582.5	0.5
Total	**349,251.4**	**100.0**	**336,297.8**	**100.0**

Source: Gartner (May2016)

Uniqueness of Mobile…

Mobile devices are shared more often

Personal phones and tablets shared with family

Enterprise tablet shared with co-workers

Social norms of mobile apps vs. file systems

Mobile devices have multiple personas

Work tool

Entertainment device

Personal organization

Security profile per persona?

Mobile devices are diverse

OS immaturity for enterprise mgmt

BYOD dictates multiple OSs

Vendor / carrier control dictates multiple OS versions

Mobile devices are used in more locations

A single location could offer public, private, and cell connections

Anywhere, anytime

Increasing reliance on enterprise WiFi

Mobile devices prioritize the user

Conflicts with user experience not tolerated

OS architecture puts the user in control

Difficult to enforce policy, app lists

Android Security Architecture

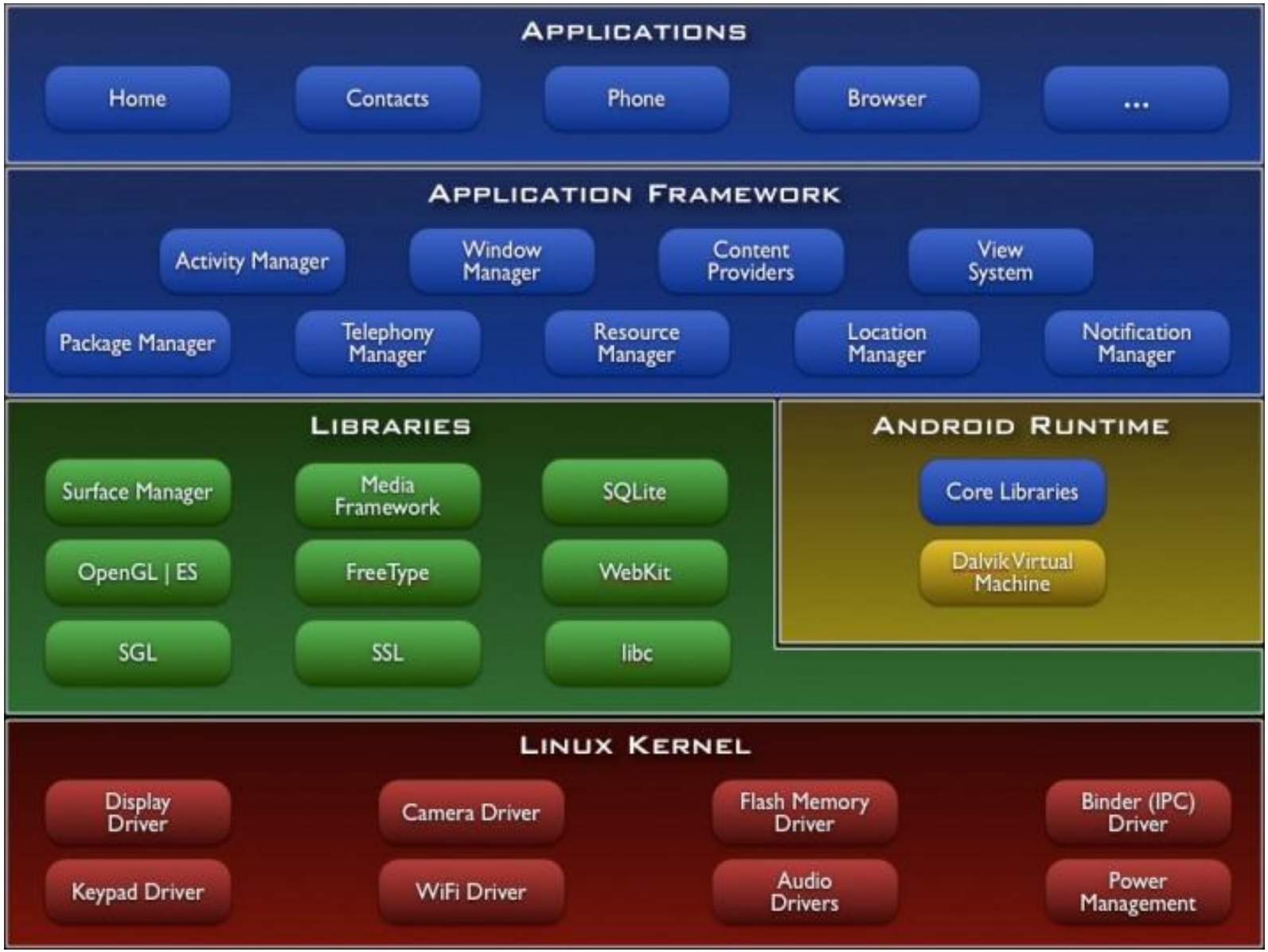
Security goals

- Protect user data
- Protect system resources (hardware, software)
- Provide application isolation

Foundations of Android Security

Application Isolation and Permission Requirement

- Mandatory application sandbox for all applications
- Secure inter-process communication
- System-built and user-defined permissions
- Application signing


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Android software stack

- Each component assumes that the components below are properly secured.
- All code above the Linux Kernel is restricted by the Application Sandbox
- Linux kernel is responsible sandboxing application
  - “mutually distrusting principals”
  - Default access to only its own data
- The app Sandbox apps can talk to other apps only via Intents (message) , IPC, and ContentProviders
- To escape sandbox, permissions is needed

Security at the Linux kernel

- A user-based permissions model
- Process isolation: Each application has its sandbox based on separation of processes: to protect user resources from each another; each runs in its own Linux process to secure Inter-Process communication (IPC)

Ex:

- Prevents user A from reading user B's files
- Ensures that user A does not access user B's CPU, memory resources
- Ensures that user A does not access user B's devices (e.g. telephony, GPS, Bluetooth)

Application Sandbox

- The Android system assigns a unique user ID (UID) to each Android application and runs it as that user in a separate process.
- When launching a new *Activity*, the new process isn't going to run as the launcher but with its own identity with the permission specified by the developer.
- The developer of that application has ensured that it will not do anything the phone's user didn't intend. Any program can ask Activity Manager to launch almost any other application, which runs with that application's UID.
- Ex. application A is not allowed to do something malicious like to read application B's data or dial the phone without permission.
- All libraries, application runtime, and all applications run within the Application Sandbox in the kernel.

Permissions and Encryption

- **Permissions**

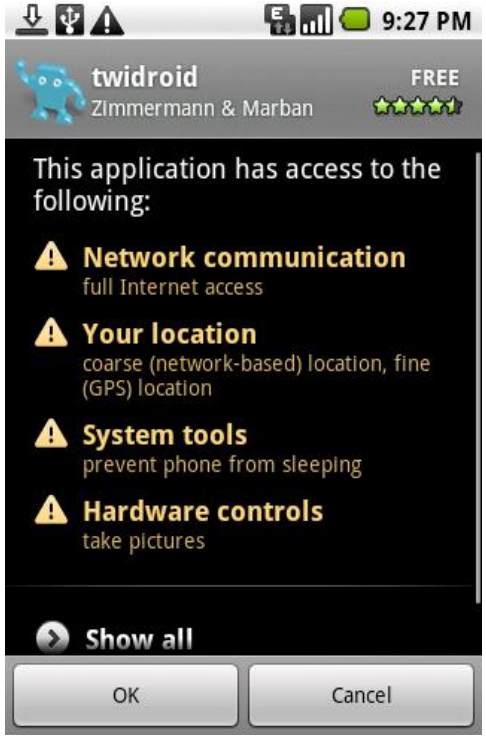
  In Android, each application runs as its own user. Unless the developer explicitly exposes files to other applications, files created by one application cannot be read or altered by another application.

- **Password Protection**

  Android can require a user-supplied password prior to providing access to a device. In addition to preventing unauthorized use of the device, this password protects the cryptographic key for full file system encryption.

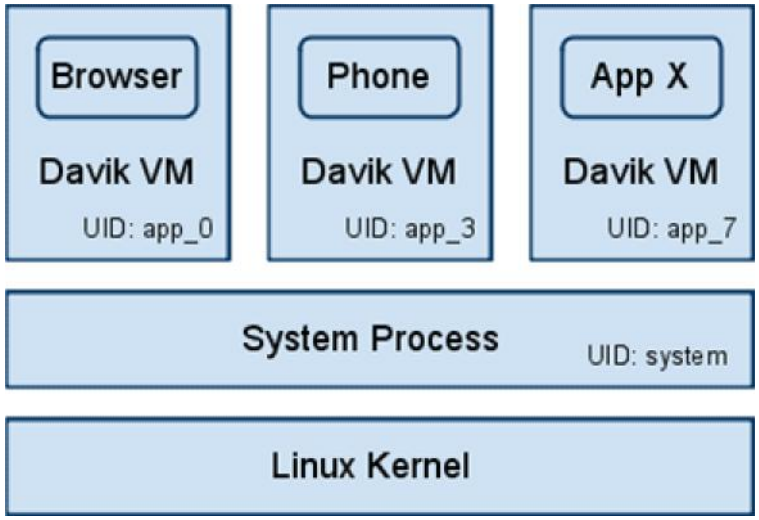
Permissions

- Whitelist model
  1. Allow minimal access by default
  2. Allow for user accepted access to resources
- Ask users less questions
- Make questions more understandable
- 194 permissions
  - More ⇒ granularity
  - Less ⇒ understandability

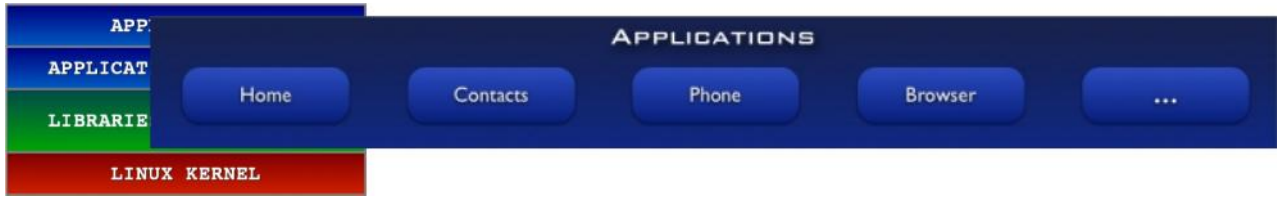


Application Sandbox

- Each application runs within its own UID and VM
- Default privilege separation model
- Instant security features
  - Resource sharing
    - CPU, Memory
  - Data protection
    - FS permissions
  - Authenticated IPC
    - Unix domain sockets
- Place access controls close to the resource, not in the VM



Android S/W Stack - Application



- Android provides a set of core applications:
  - ✓ Email Client
  - ✓ SMS Program
  - ✓ Calendar
  - ✓ Maps
  - ✓ Browser
  - ✓ Contacts
  - ✓ Etc

- All applications are written using the Java language.

Android S/W Stack – App Framework

- Enabling and simplifying the reuse of components
  - ✓ Developers have full access to the same framework APIs used by the core applications.
  - ✓ Users are allowed to replace components.

Android S/W Stack – App Framework (Cont)

- Features

Feature	Role
View System	Used to build an application, including lists, grids, text boxes, buttons, and embedded web browser
Content Provider	Enabling applications to access data from other applications or to share their own data
Resource Manager	Providing access to non-code resources (localized strings, graphics, and layout files)
Notification Manager	Enabling all applications to display customer alerts in the status bar
Activity Manager	Managing the lifecycle of applications and providing a common navigation backstack

Android S/W Stack - Libraries

- Including a set of C/C++ libraries used by components of the Android system
- Exposed to developers through the Android application framework

Android S/W Stack - Runtime

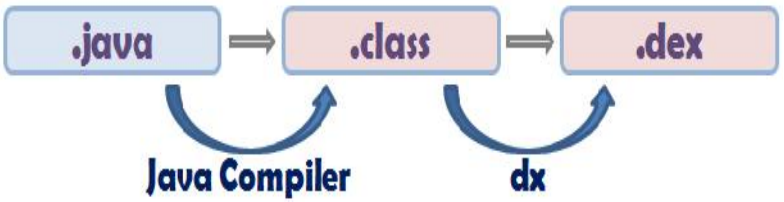
- Core Libraries
  - ✓Providing most of the functionality available in the core libraries of the Java language
  - ✓APIs
    - ➢ Data Structures
    - ➢ Utilities
    - ➢ File Access
    - ➢ Network Access
    - ➢ Graphics
    - ➢ Etc

Android S/W Stack – Runtime (Cont)

- Dalvik Virtual Machine
  - ✓Providing environment on which every Android application runs
    - ➢ Each Android application runs in its own process, with its own instance of the Dalvik VM.
    - ➢ Dalvik has been written such that a device can run multiple VMs efficiently.
  - ✓Register-based virtual machine

Android S/W Stack – Runtime (Cont)

- Dalvik Virtual Machine (Cont)
  - ✓Executing the Dalvik Executable (.dex) format
    - ➢ .dex format is optimized for minimal memory footprint.
    - ➢ Compilation



  - ✓Relying on the Linux Kernel for:
    - ➢ Threading
    - ➢ Low-level memory management

Android S/W Stack – Linux Kernel

- Relying on Linux Kernel 2.6 for core system services
  - ✓ Memory and Process Management
  - ✓ Network Stack
  - ✓ Driver Model
  - ✓ Security
- Providing an abstraction layer between the H/W and the rest of the S/W sta ck

Mobile Threats and Attacks

- Mobile devices make attractive targets:
  - People store much personal info on them: email, calendars, contacts, pictures, etc.
  - Sensitive organizational info too…
  - Can fit in pockets, easily lost/stolen
  - Built-in billing system: SMS/MMS (mobile operator), in-app purchases (credit card), etc.
    - Many new devices have near field communications (NFC), used for contactless payments, etc.
    - Your device becomes your credit card
  - Location privacy issues
- NFC-based billing system vulnerabilities

Device Malware

- iOS malware: very little
- Major increase in Android malware from 2010 to 2015 Android malware growth keeps increasing ($$$)
- Main categories:
  - Trojans
  - Monitoring apps/spyware
  - Adware
  - Botnets

Mobile Malware Behavior

Overview on Malware Behavior

- Since the first discovery of android malware known as Fake player in 2010; malware for android has growth to an alarming rate.
- The malicious activities executed by the malware can be varies, for instance Fake player is a trojan that hide behind a legitimate Movie player application and can sent SMS messages automatically to premium rate number without the user knowledge [8].
- Geimini, Pjapss and HippoSMS are other example of android malware that capture or send SMS without the user knowledge.
- Some other android malware are more sinister, it can exploit the android vulnerabilities and gain an administrator or root access, Droiddream, DroidKungfu and BaseBridge are an example of android malware that come with privilege escalation binary for exploiting android vulnerabilities

- Stealing device and user credential information.
  - Most of the android malware are collecting information regarding International Mobile Equipment Indentity (IMEI) number, International Mobile Subscriber Identity (IMSI) number, GPS Location, Phone number, SDK version and Installed package.
  - Android malware also captured user activities such as SMS send or received and call out or in duration.

- Communicate with Command and Control (C&C) server.
  - Android malware have the intention to communicate to a C&C server similar to Botnet. Communication can be made through HTTP or SMS.
  - This connection are made for sending captured information from the device and also for updating the malware package or receiving any other malicious information for instance SMS premium number or any other malicious URL.

- Sending premium rate SMS and spam.
  - Android malware has the ability to automatically send SMS without the user knowledge.
  - This can be used as a spam or gathering illegal income by charging user when the SMS is send to the premium number

- Search engine Optimization.
  - Android malware can have an automatic script to visit ad, for increasing the number of visitor to a website or for the purpose of generating a charge per click without having actual interest in the target of the ad's link. Even though, it seem like it does not cause any harm, this activity can increase mobile data consumption especially to the user who are subscribing for the mobile broadband it will cause them extra charges.

- Updating and download package.
  - Android malware can used the communication made to the C&C server to update the existing package into a more malicious intention or even download a new package and installed it automatically in the user devices.

- Draining resources.
  - Android malware application can executed a malicious payload that can utilize all the disk storage or memory (RAM) quotas and hogging the CPU.

Static Analysis in Mobile Malware

Android App.

- Android application developers develop application in Java and control their operation using Java Libraries designed by Google.
- The Java application *.class file is converted into *.dex file using DX tool. *.dex is the executable format for android applications.
- The Java application program is compiled using the Java Compiler (javac) and the *.class file is generated which is given to the Dalvik VM.
- It generates *.dex file which is executed.

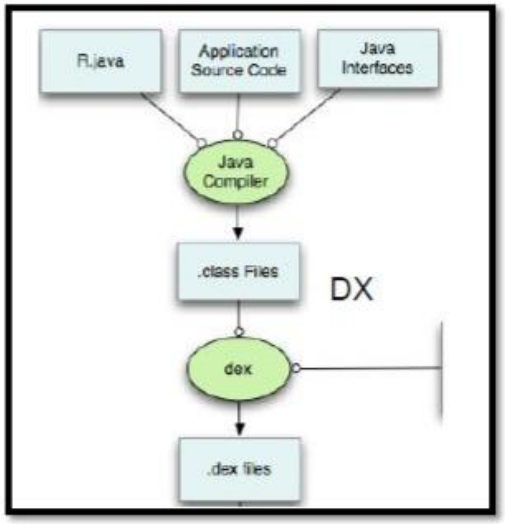


Figure 3: Flowchart on Dex Process Flow

- Android Package File (APK) consists of the following files:
  - .dex file
  - Res - Resources which APK requires.
  - Android Manifest xml file
  - META-INF directory which contains the certificate, the list of resources and manifest file.
- All these files and folders are bundled together to form the apk file.

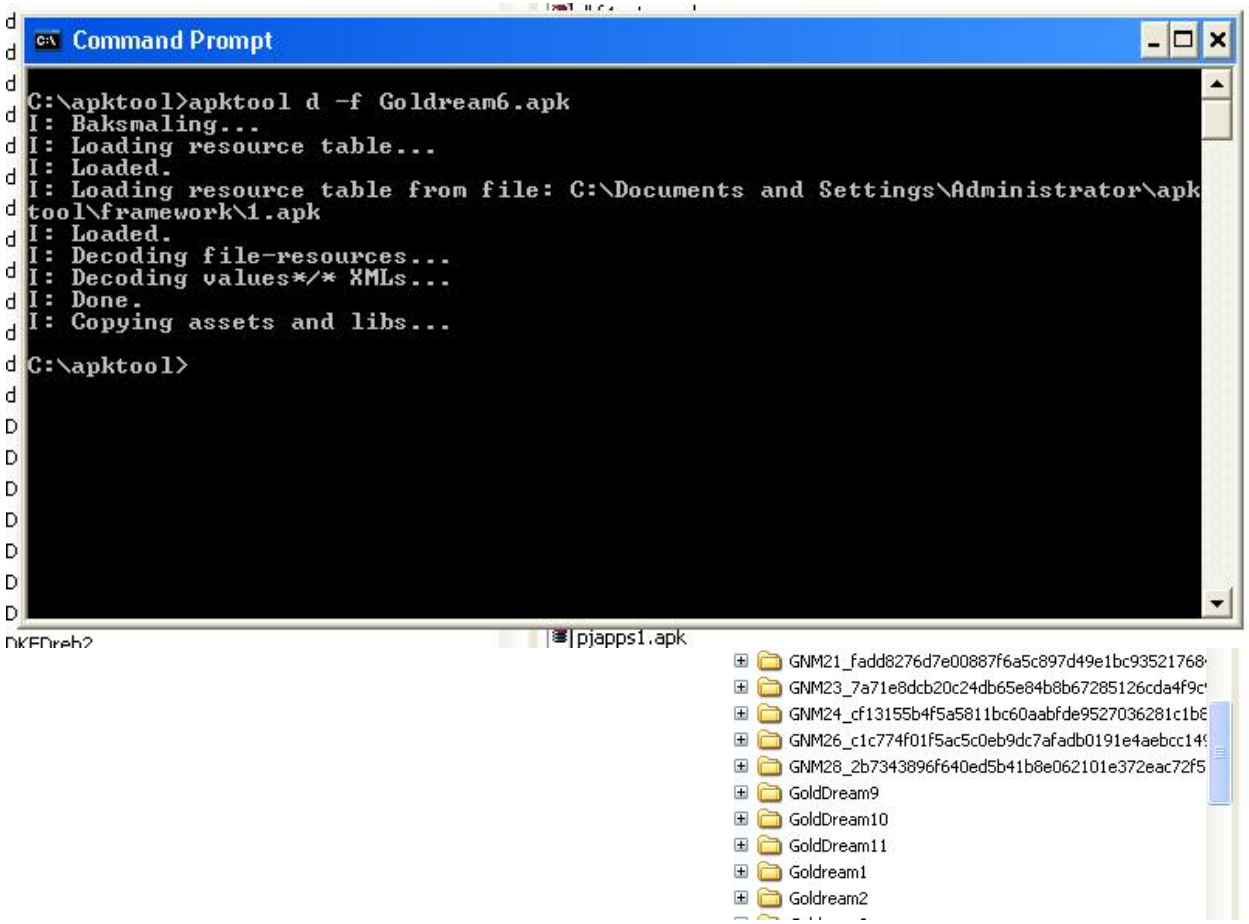
Potential Information to detect Malware Behavior

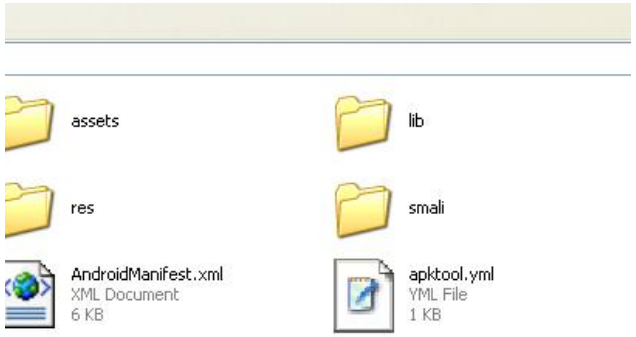
- Manifest (permission & Package name)
- API

Static analysis

- Disassembler is used to convert the code into hexadecimal format for better analysis during dynamic analysis.
  - **Apktool**
    - decode the malicious code to its original code
    - Able to modify the code and recompile using Apktool [16].
    - uses Baksmali for disassembling and Smali for assembling the code.
    - This dex format is used by Dalvik VM. Baksmali and Smali are the Icelandic names for “Disassembler” and “Assembler”.
- Unpacker is a tool used to unzip the compressed contents of the file or folder which is required in the uncompressed format for analysis.
  - **7-Zip**
    - helps to unzip the apk file of the application.
    - has high compression ratio, strong encryption and 2-10% more compression capability compared with WinZip, PKZip etc
  - **Dex2Jar**
    - Dex2Jar is a tool, which is used to convert the dex code into *.jar Java file.
  - **JD-GUI**
    - used to view those *.jar files.
    - can load all the packages embedded in the jar file and lists the *.java code.

apktool d –f [file.apk]

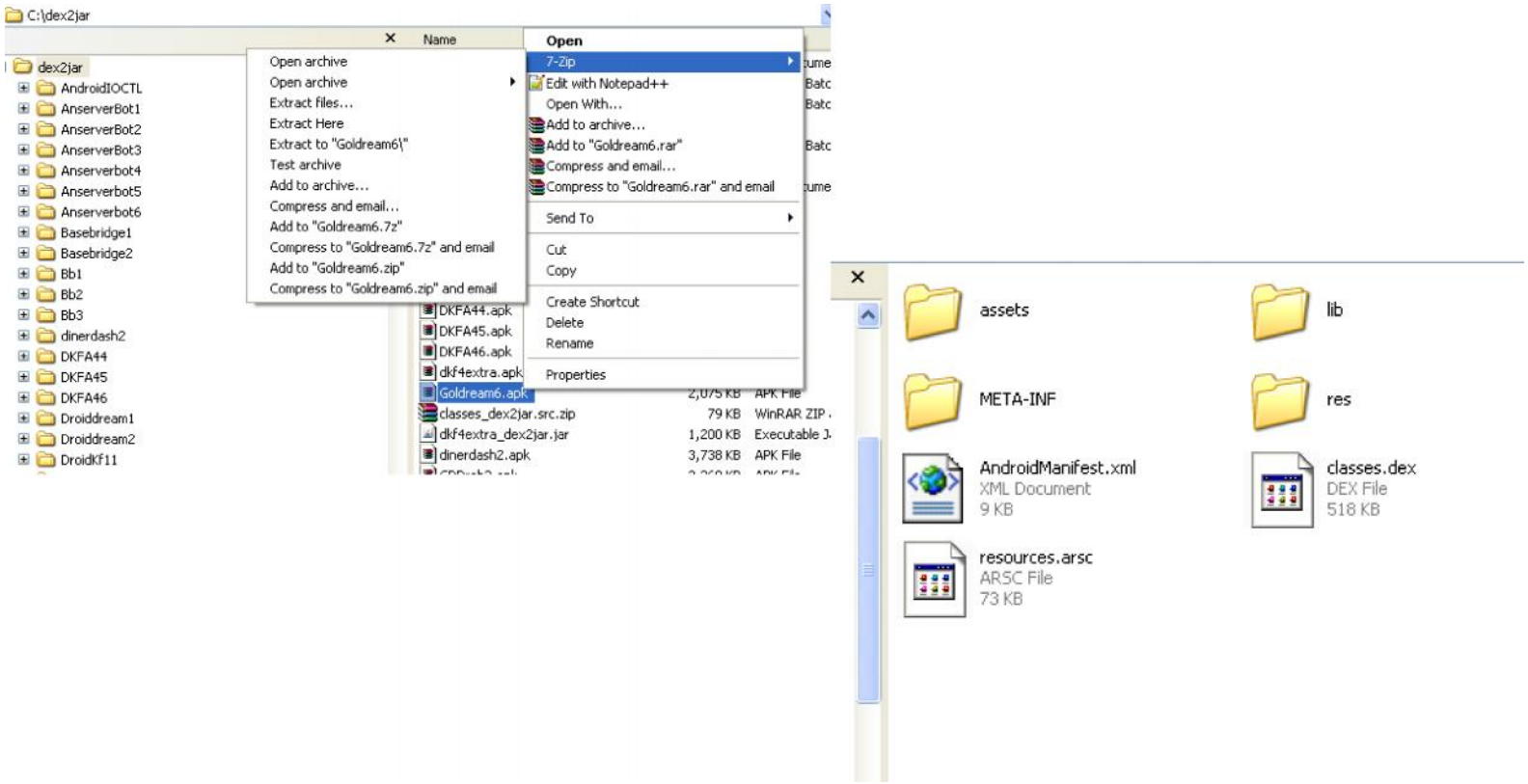




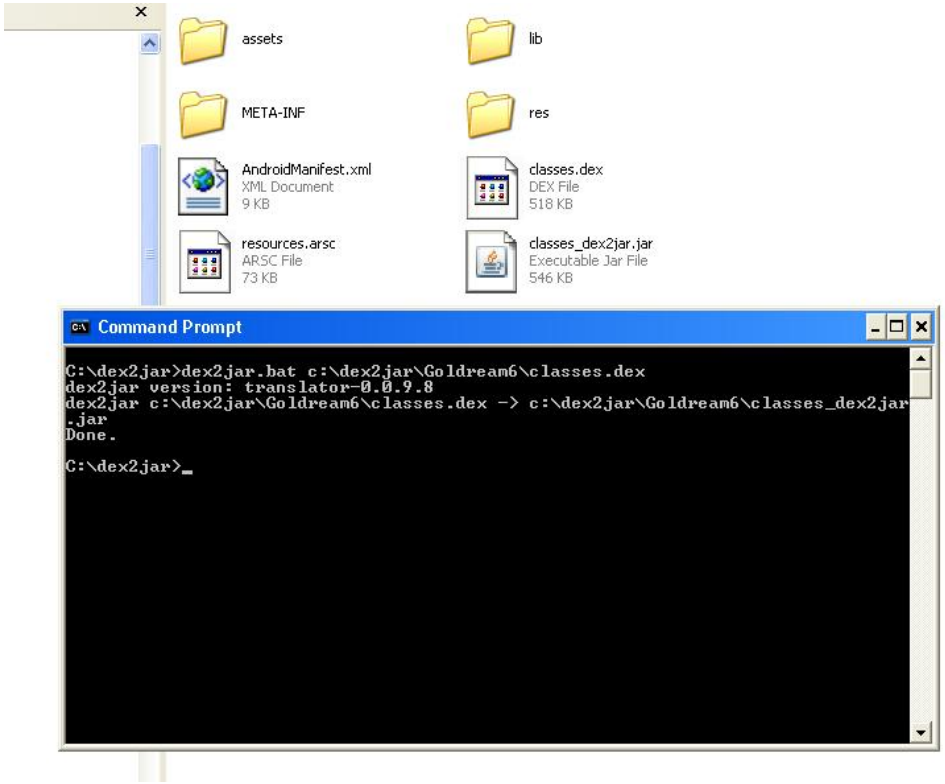
Manifest File

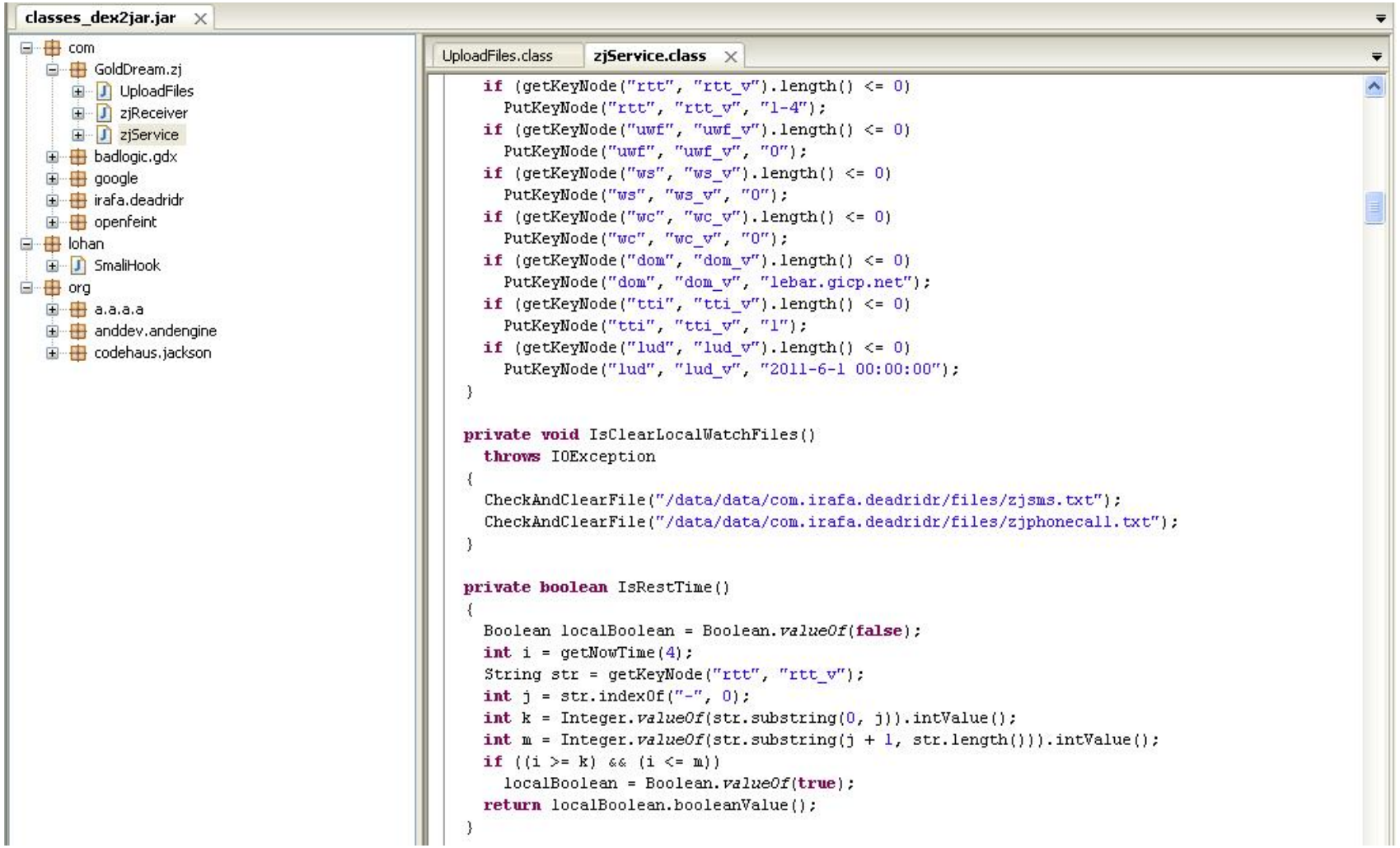
```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="19" android:versionName="1.6.3" android:installLocation="preferExternal" package=
"com.irafa.deadridr"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <application android:label="@string/app_name" android:icon="@drawable/icon">
        <activity android:theme="@android:style/Theme.NoTitleBar.Fullscreen" android:label="@string/app_name"
        android:name=".DeadRider" android:screenOrientation="landscape" android:configChanges="keyboardHidden">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <receiver android:name="com.GoldDream.zj.zjReceiver">
                <intent-filter>
                    <action android:name="android.intent.action.BOOT_COMPLETED" />
                    <action android:name="android.provider.Telephony.SMS_RECEIVED" />
                    <action android:name="android.intent.action.PHONE_STATE" />
                    <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
                </intent-filter>
            </receiver>
            <service android:label="Market" android:name="com.GoldDream.zj.zjService" android:exported="true"
            android:process="" />
        </activity>
        <activity android:theme="@android:style/Theme.NoTitleBar.Fullscreen" android:name=".LevelSelectActivity"
        android:screenOrientation="landscape" android:configChanges="keyboardHidden" />
        <activity android:theme="@android:style/Theme.NoTitleBar.Fullscreen" android:name=".Preferences"
        android:screenOrientation="landscape" android:configChanges="keyboardHidden" />
        <activity android:theme="@android:style/Theme.NoTitleBar.Fullscreen" android:name=".ControllPreferences"
        android:screenOrientation="landscape" android:configChanges="keyboardHidden" />
        <activity android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen" android:name=".PauseMenu"
        android:multiprocess="false" android:screenOrientation="landscape" android:configChanges="keyboardHidden"
        android:noHistory="true" />
        <activity android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen" android:name=".FailMenu"
        android:multiprocess="false" android:screenOrientation="landscape" android:configChanges="keyboardHidden"
```

7 zip


C:\dex2jar
dex2jar
AndroidIOCTL
AnserverBot1
AnserverBot2
AnserverBot3
Anserverbot4
Anserverbot5
Anserverbot6
Basebridge1
Basebridge2
Bb1
Bb2
Bb3
dinerdash2
DKFA44
DKFA45
DKFA46
Droiddream1
Droiddream2
DroidKf11
Name
Open archive
Open archive
Extract files...
Extract Here
Extract to "Goldream6\"
Test archive
Add to archive...
Compress and email...
Add to "Goldream6.7z"
Compress to "Goldream6.7z" and email
Add to "Goldream6.zip"
Compress to "Goldream6.zip" and email
Open
7-Zip
Edit with Notepad++
Open With...
Add to archive...
Add to "Goldream6.rar"
Compress and email...
Compress to "Goldream6.rar" and email
Send To
Cut
Copy
Create Shortcut
Delete
Rename
Properties
DKFA44.apk
DKFA45.apk
DKFA46.apk
dkf4extra.apk
Goldream6.apk
classes_dex2jar.src.zip 79 KB WinRAR ZIP
dkf4extra_dex2jar.jar 1,200 KB Executable J
dinerdash2.apk 3,738 KB APK File
assets
lib
META-INF
res
AndroidManifest.xml
XML Document
9 KB
classes.dex
DEX File
518 KB
resources.arsc
ARSC File
73 KB

Dex2jar.bat [path to classes.dex]




classes_dex2jar.jar
com
GoldDream.zj
UploadFiles
zjReceiver
zjService
badlogic.gdx
google
irafa.deadridr
openfeint
lohan
SmaliHook
org
a.a.a.a
anddev.andengine
codehaus.jackson
UploadFiles.class
zjService.class
if (getKeyNode("rtt", "rtt_v").length() <= 0)
PutKeyNode("rtt", "rtt_v", "1-4");
if (getKeyNode("uwf", "uwf_v").length() <= 0)
PutKeyNode("uwf", "uwf_v", "0");
if (getKeyNode("ws", "ws_v").length() <= 0)
PutKeyNode("ws", "ws_v", "0");
if (getKeyNode("wc", "wc_v").length() <= 0)
PutKeyNode("wc", "wc_v", "0");
if (getKeyNode("dom", "dom_v").length() <= 0)
PutKeyNode("dom", "dom_v", "lebar.gicp.net");
if (getKeyNode("tti", "tti_v").length() <= 0)
PutKeyNode("tti", "tti_v", "1");
if (getKeyNode("lud", "lud_v").length() <= 0)
PutKeyNode("lud", "lud_v", "2011-6-1 00:00:00");
}
private void IsClearLocalWatchFiles()
throws IOException
{
CheckAndClearFile("/data/data/com.irafa.deadridr/files/zjsms.txt");
CheckAndClearFile("/data/data/com.irafa.deadridr/files/zjphonecall.txt");
}
private boolean IsRestTime()
{
Boolean localBoolean = Boolean.valueOf(false);
int i = getNowTime(4);
String str = getKeyNode("rtt", "rtt_v");
int j = str.indexOf("-", 0);
int k = Integer.valueOf(str.substring(0, j)).intValue();
int m = Integer.valueOf(str.substring(j + 1, str.length())).intValue();
if ((i >= k) && (i <= m))
localBoolean = Boolean.valueOf(true);
return localBoolean.booleanValue();
}

Androguard –Automate all the previous Process

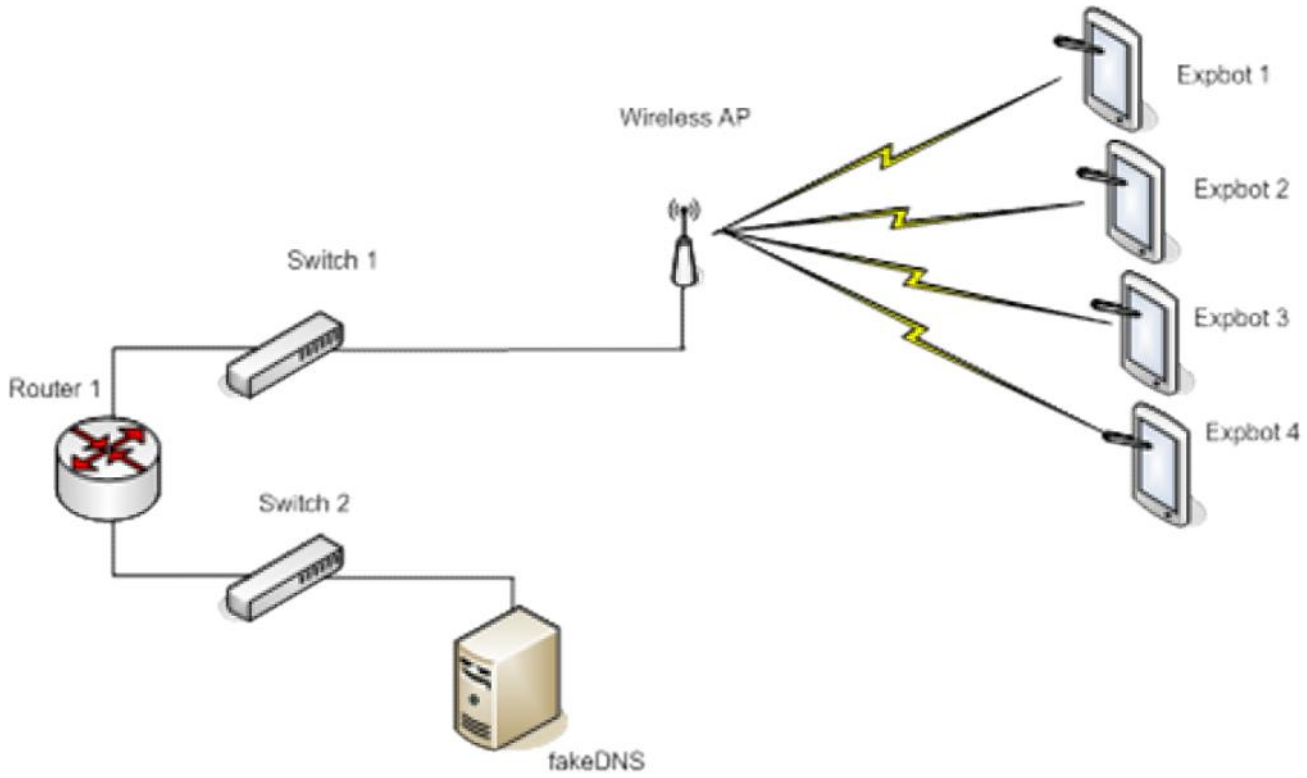

```
In [1]: a, d, dx = AnalyzeAPK("./apks/porn
./apks/pornoplayer.apk      ./apks/pornoplayer.apk.txt  ./apks/pornoplayer2.apk

In [1]: a, d, dx = AnalyzeAPK("./apks/pornoplayer.apk
./apks/pornoplayer.apk      ./apks/pornoplayer.apk.txt

In [1]: a, d, dx = AnalyzeAPK("./apks/pornoplayer.apk")

In [2]: a, d, dx
Out[2]:
(<androguard.core.bytecodes.apk.APK instance at 0x35aa6c8>,
 <androguard.core.bytecodes.dvm.DalvikVMFormat at 0x36bda90>,
 <androguard.core.analysis.analysis.uVMAnalysis instance at 0x36db878>)

In [3]: save_session([a, d, dx], "w00t.ag")

In [4]:
Do you really want to exit ([y]/n)? y

desnos@t0t0:~/androguard$ ./androlyze.py -s
Androlyze version 1.5
In [1]: a, d, dx = load_session("w00t.ag")

In [2]: a, d, dx
Out[2]:
(<androguard.core.bytecodes.apk.APK instance at 0x22ef638>,
 <androguard.core.bytecodes.dvm.DalvikVMFormat at 0x23af190>,
 <androguard.core.analysis.analysis.uVMAnalysis instance at 0x2472638>)
```

Dynamic Analysis

Data to be collected

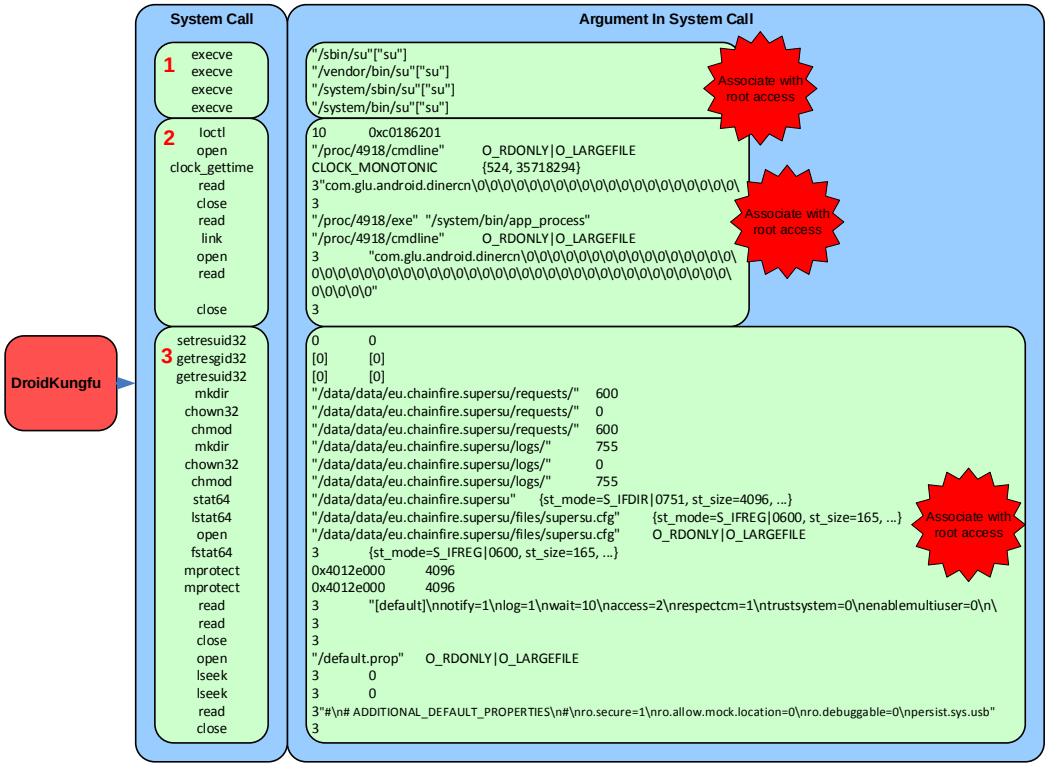
- Network Traffic
- Memory Usage
- CPU Usage
- System Call

Isolated environment

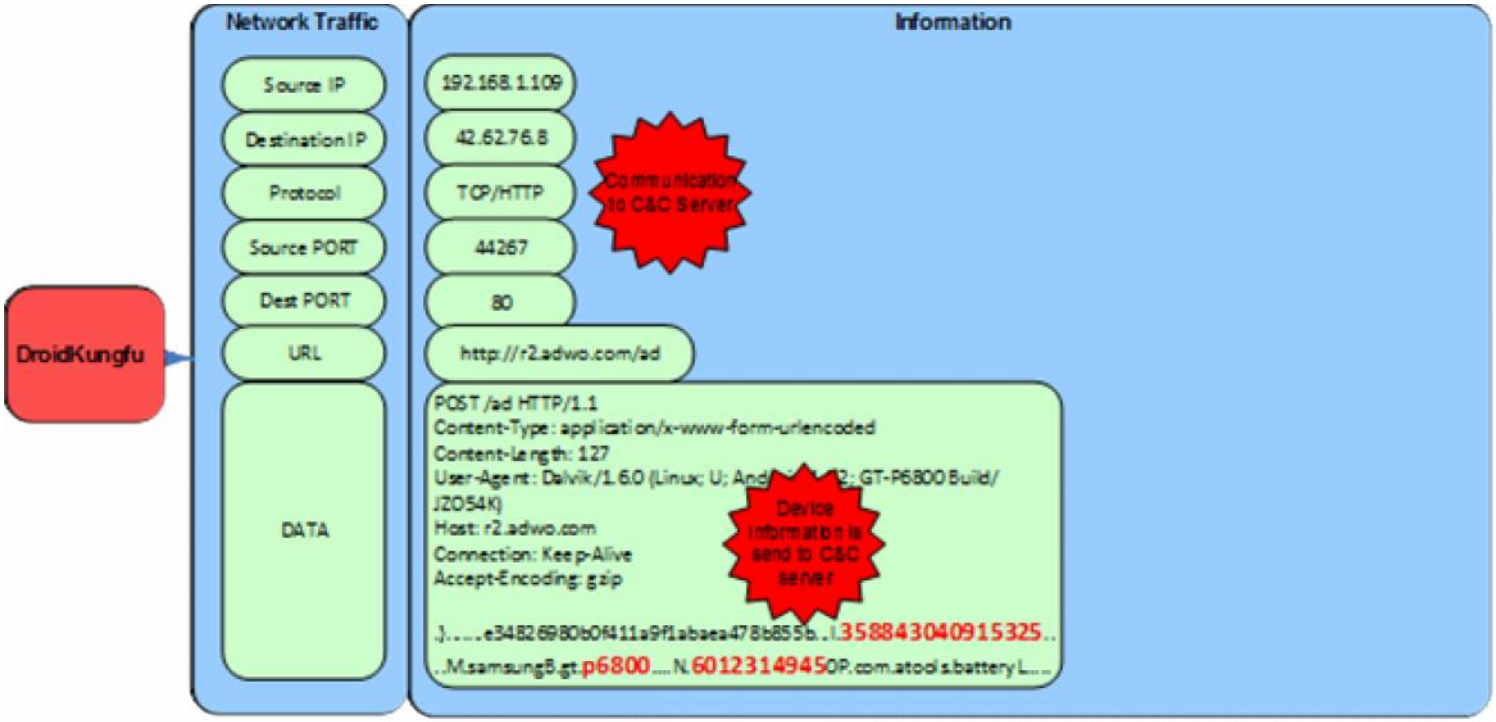


Dynamic Analysis Tool

- Virtual Machine
- Tcpdump
- Strace
  - **strace** is a diagnostic, debugging and instructional userspace utility for Linux.
  - It is used to monitor interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state.
  - The operation of strace is made possible by the kernel feature known as ptrace



Tcpdump through wireshark



- Procrank
  - procrank show a quick summary of process memory utilization.
  - By default, it shows Vss, Rss, Pss and Uss, and sorts by Vss.
  - procrank source is included in system/extras/procrank, and the binary is located in /system/xbin on an android device.
    - Vss = virtual set size
    - Rss = resident set size
    - Pss = proportional set size
    - Uss = unique set size
  - In general, the two numbers to watch are the Pss and Uss

```
# procrank
  PID      Vss      Rss      Pss      Uss  cmdline
 1217   36848K   35648K   17983K   13956K  system_server
 1276   32200K   32200K   14048K   10116K  android.process.acore
 1189   26920K   26920K    9293K    5500K  zygote
 1321   20328K   20328K    4743K    2344K  android.process.media
 1356   20360K   20360K    4621K    2148K  com.android.email
 1303   20184K   20184K    4381K    1724K  com.android.settings
 1271   19888K   19888K    4297K    1764K  com.android.inputmethod.latin
 1332   19560K   19560K    3993K    1620K  com.android.alarmclock
 1187    5068K    5068K    2119K    1476K  /system/bin/mediaserver
 1384     436K     436K     248K     236K  procrank
    1     212K     212K     200K     200K  /init
  753     572K     572K     171K     136K  /system/bin/rild
  748     340K     340K     163K     152K  /system/bin/sh
  751     388K     388K     156K     140K  /system/bin/vold
 1215     148K     148K     136K     136K  /sbin/adbd
  757     352K     352K     117K      92K  /system/bin/dbus-daemon
  760     404K     404K     104K      80K  /system/bin/keystore
  759     312K     312K     102K      88K  /system/bin/installd
  749     288K     288K      96K      84K  /system/bin/servicemanager
  752     244K     244K      71K      60K  /system/bin/debuggerd
```

Summary

- Mobile Malware Overview
- Mobile Malware Behavior
- Static Analysis on Mobile malware
- Dynamic Analysis on Mobile malware
- Mobile Malware Mitigation