Dissecting a RAT. Analysis of the AndroRAT. — Stratosphere Laboratory

Published: 2021-03-31 · Archived: 2026-04-02 11:53:51 UTC

This blog post was authored by Kamila Babayeva (@_kamifai_) and Sebastian Garcia (@eldracote).

The RAT analysis research is part of the Civilsphere Project (https://www.civilsphereproject.org/), which aims to protect the civil society at risk by understanding how the attacks work and how we can stop them. Check the webpage for more information.

This is the fourth blog of a series analyzing the network traffic of Android RATs from our Android Mischief Dataset [more information here], a dataset of network traffic from Android phones infected with Remote Access Trojans (RAT). In this blog post we provide the analysis of the network traffic of the RAT05-AndroRAT [download here]. The previous blogs analyzed Android Tester RAT, DroidJack RAT, and SpyMax RAT.

RAT Details and Execution Setup

The goal of each of our RAT experiments is to configure and execute the RAT software and to do every possible action while capturing all traffic and storing all logs. These RAT captures are functional and used as in real attacks.

The AndroRAT RAT is a software package that contains the controller software and builder software to create an APK. We executed the builder on a Windows 7 Virtualbox virtual machine with Ubuntu 20.04 as a host. The Android Application Package (APK) built by the RAT builder was installed in an Android virtual emulator called Genymotion with Android version 8.

While performing different actions on the RAT controller (e.g. upload a file, get GPS location, monitor files, etc.), we captured the network traffic on the Android virtual emulator. The network traffic from the phone was captured using Emergency VPN.

The details about the network traffic capture are:

- The controller IP address: 147.32.83.234
- The phone IP address: 10.8.0.137
- UTC time of the infection in the capture: 2020-09-10 15:18:00 UTC

Initial Communication and Infection

Once the APK was installed on the phone, it directly tries to establish a TCP connection with the command and control (C&C) server. To connect, the phone uses the IP address and the port of the controller specified in the APK. In our case, the IP address of the controller is 147.32.83.234 and the port is 1337/TCP. The controller IP

147.32.83.234 is the IP address of a Windows 7 virtual machine in our lab computer, meaning that the IP address is not connected to any known indicator of compromise (IoC). Figure 1 shows the initial communication from the phone to the C&C.

Figure 1. A 3-way handshake to establish the first connection between the phone and the C&C.

After establishing the first connection, the phone sends its first packet with some parameters, such as SIM card operator, phone number, SIM card serial number, IMEI, etc. Figure 2 displays the packet data in a structured way. It can be seen that the data is sent in plaintext and the character 't' is used as the delimiters to separate parameters name and values. From the packet structure in Figure 2,it can also be defined that APK uses the Java Hashtable class to store and send parameters.

ÿÿ¬ísr**java.util.Hashtable**»%!Jä¸F
loadFactorI thresholdxp?@
w

t Operator t Android
t SimOperator t Android
t SimSerial t 8931027000000000007
t SimCountry t us
t PhoneNumber t 15555218135
t Country t us
t IMEI t 000000000000000

x

Figure 2. The first data packet sent by the phone and an analysis of its structure. The data is sent in the plain text and the character 't' is used as a field delimiter.

After the initial connection by the phone, the command and control server shows the phone in its interface. Figure 3 displays the C&C interface with the initialization parameters that were sent by the phone in the first packet.

Figure 3. The C&C interface panel displays the parameters of the phone after the infection.

Figure 4. Panel in the C&C interface used to send commands to the phone.

0000 00 00 00 06 00 00 00 06 01 00 00 00 00 00 00 00 ....
0010 79 00 00 01 67 y...g

Figure 5. Data of the first packet sent by the C&C when the attacker enters into the panel to control the phone.

0000 00 00 00 06 00 00 00 06 01 00 00 00 00 00 00 00 ....
0010 15 00 00 02 6e ....n

Figure 6. Data of the second packet sent by the C&C when the attacker enters into the panel to control the phone.

Since the structure of these packets is not clear, we tried to understand what these commands mean by reverse engineering the APK that was used to infect the victim's phone. The analysis shows that each C&C command is

mapped to a single character that represents this command. The mapping is shown in Figure 7.

Figure 7. The mapping of each C&C commands (in capital letters) into a single character defined by a number. Found by reverse engineering the APK used to infect the victim.

Figure 8. Java code from the APK for the function *dataHeaderGenerator*. This function generates the header for the C&C and phone packets.

Figure 9. Java code from the malicious APK for the function *parse*. This function unwraps the C&C command.

Figure 10. Analysis of the packet structure of the C&C command '*Advanced Information*' sent to the phone.

Figure 10 shows an analysis diagram of the meaning of a packet sent to the phone with the command 'Advanced Information'. This packet has a data length of 6, therefore everything after the field *byteChannel* (00 79 00 00 01 67) has a length of 6 bytes. The bytes 00 79, which are used to represent C&C command, mean 121 in decimal representation. According to the mapping in Figure 7, the value 121 responds to the command 'Advanced Information'. Figure 11 shows how. The variable P_INST is 100, and the command GET_ADV_INFORMATIONS is P_INST + 21 = 100 + 21 = 121

Figure 11. Mapping value of the C&C command 'GET_ADV_INFORMATIONS'. The value of this command is 121 in decimal which is 00 79 in hexadecimal.

Figure 12. Analysis of the packet structure of the C&C command 'Preferences' sent to the phone.

For this packet, the data length is 6, therefore everything after the field *byteChannel* (00 15 00 00 02 6e) has a length of 6 bytes. The bytes 00 15, that are used for defining C&C command, mean 21 in decimal representation. According to the mapping in Figure 7, it is the command 'Preferences'. Figure 13 shows how this command is computed.

Figure 13. Mapping of the C&C command GET_PREFERENCE from Figure 7. GET_PREFERNCES is 21 in decimal, and 00 15 in hexadecimal.

Figure 14. Summary of the packet structure of the C&C commands.

Victim Phone Packet Structure

The phone answers to the C&C command 'getPreferences' and the command 'Advanced informations' with its own packets. The structure of the packets sent from the phone is different from the C&C command packet structure shown in Figure 14. Figure 15 shows the analysis of APK function 'send' that sends the packet from the phone with a specific structure. The packet structure the function uses is the following:

Name Length
header 15 bytes
data no more than 2033 bytes

The header in that structure uses a substructure:

Name Length
byteTotalLength 4 bytes
byteLocalLength 4 bytes
byteMoreF 1 byte
bytePointeurData 2 bytes
byteChannel 4 bytes

As for the data in the packet, if its length exceeds the limit of 2033 bytes, the data will be fragmented into more packets. Each packet will have a separate 15 bytes long header and will be fragmented with a length of 2033 bytes or less.

> ***Figure 15.*** Java code from the APK for the command 'send'. This function sends the packet from the phone according to the specific structure.

Using this structure we can now interpret the packets sent by the phone. Figure 16 shows the phone answer to the C&C command 'get Preferences' and the structure of the packet. The phone sends the 15 byte long header followed by the data. The data in Figure 16 includes the preferred parameters for phonNumberCall, phoneNumberSMS, keywordSMS.

> ***Figure 16.*** Packet sent from the phone as an answer to the C&C command 'get Preferences'. The packet data and its structure is shown.

The phone sends data about the battery status, phone info, and wifi information to answer the C&C command 'Advanced Information'. The phone uses the same structure of 15 byte long header and the data. Figure 17 shows a raw answer of the phone to the C&C command 'Advanced Information'.

```
JJg¬ísr Packet.AdvancedInformationPacketqB®IandroidSdkIbatteryHealthIbatteryLevelIbatteryPluggedZbatteryPresen
```

Figure 17. Raw answer as ASCII text sent from the phone to the C&C command 'Advanced Information'. The packet data and its structure is shown. It is not easy to find the field separators.

The summary of the structure of the packets sent from the phone is:

> ***Figure 18.*** The structure of the packet sent from the phone.

> ***Figure 19.*** Packet data and structure for the C&C command 'Toast' with the argument 'hello'.

The C&C command sent has the value 00 6d in hexadecimal or 109 in decimal representation. We can confirm that this mapping responds to the command 'Toast' (Figure 20). It is important to notice that the C&C command is mapped to the single character, but its argument 'hello' (68 65 6c 6c 6f) is not mapped to anything.

```
public static final short DO_TOAST = ((short) (P_INST + 9));
```

Figure 20. The mapping of the C&C command ‘Toast’. The value of this command is 109 which is 00 6d in hexadecimal.

‘Toast hello’ was successfully performed on the phone. The phone in return did not send any confirmation of the successful operation. Only for the C&C commands that require the phone to send information (e.g. file, call, sms), the phone sends the packet with the confirmation of receiving the command. Afterwards, it sends the required data.

As an example, we took the C&C command ‘Directory List’. The communication gos as follows:

1. The C&C sends the command ‘Directory List’ with the directory as an argument. (Figure 21)
2. The phone sends the confirmation of the command being received. (Figure 22)
3. The phone send the required data, i.e. file list in the directory. (Figure 23)

Figure 21. The packet data and its structure of the C&C command ‘Directory List’. The command aims to get the list of files in the specified directory (in our case directory ‘/’).

Figure 22. The phone sends the confirmation about the received command ‘Directory List’. The packet data and its structure is shown.

```
3ñ¬ísrjava.util.ArrayListxÒÇaIsizexpwsrutils.MyFileëÀÞ¹÷ÓZhiddenZisDirZisFileJ        lastModifJlengthZrZwLli
```

Figure 23. The phones send the list of files in a specified directory from the C&C command ‘Directory List’.

Long Connections

If we use the Wireshark tool to analyze all the traffic, we can open the menu “Conversations”, then “Statistics”, then “TCP”. There were several connections between the C&C (147.32.83.234) and the phone (10.8.0.37) as shown in Figure 24. The longest connection established between the C&C and the phone is 2611.3454 seconds long (approximately 44 minutes). This indicates that the connections between the phone and the controller are kept for a long period of time in order to answer fast.

Figure 24. All the connections in the traffic between the phone and the C&C.

Figure 25 displays all the TCP connections in the phone sorted by the highest connection duration. It is important to notice that there are even longer *normal* connections with durations of 3576.9112 seconds (approximately 57 minutes). This is the connection from the phone during normal operation to the IP address 157.240.30.11 which belongs to Facebook services.

Figure 25. Top connections from the phone from Wireshark -> Statistics -> Conversations -> TCP.

Conclusion

In this blog we have analyzed the network traffic from a phone infected with AndroRAT. We were able to decode its connection. The androRAT does not seem to be complex in its communication protocol and it is not sophisticated in its work.

To summarize, the details found in the network traffic of this RAT are:

- The phone connects directly to the IP address and ports specified in APK (default port and custom port).
- There is only one long connection, i.e. more than 40 minutes, between the phone and the controller over the port 1337/TCP.
- There is no heartbeat between the controller and the phone.
- The data is sent in the plain text.
- The C&C uses mapping to present the C&C command as a single character.
- Packets sent from the phone have a structure of **{byteTotalLength}{byteLocalLength}{byteMoreF}{bytePointeurData}{byteChannel}{data}**
- Packets sent from the C&C have a structure of **{byteTotalLength}{byteLocalLength}{byteMoreF}{bytePointeurData}{byteChannel}{C&C command}{targetChannel}{arguments}.**

Biographies
