Dissecting a RAT. Analysis of DroidJack v4.4 RAT network traffic.

Kamila Babayeva February 3, 2021

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 second 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 RAT02-DroidJack v4.4 [download here].

RAT Details and Execution Setup

The goal of each of our RAT experiments is to use the software ourselves and to execute every possible action while capturing all the traffic and storing all the logs. So these RAT captures are functional and were used in real attacks.

The DroidJack v.4.4 RAT is a software package that contains the controller software and builder software to build an APK. It was executed on a Windows 7 virtual machine with Ubuntu 20.04 as a host. The Android Application Package (APK) built by the RAT builder was installed in the 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 details about the network traffic capture are:

- The controller IP address: 147.32.83.253
- The phone IP address: 10.8.0.57
- UTC time of the infection in the capture: 2020-08-01 14:10:43 UTC

Initial Communication and Infection

Once the APK was installed in 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 147.32.83.253 and the port is 1337/TCP. Also, DroidJack uses the port 1334/TCP as a default port and the phone connects to it later too. The controller IP 147.32.83.253 is the IP address of Windows 7 virtual machine in our lab computer, meaning that the IP address is not connected to any indicator of compromise (IoC).

54650	2020-08-01 14:10:43	10.8.0.57	41881	147.32.83.253	1337	TCP	60	41881 → 1337 [SYN] Seq=0 Win=65535 Len=0 MSS=1361 SACK_PERM=1 TSval=271622 TSecr=0
54651	2020-08-01 14:10:43	147.32.83.253	1337	10.8.0.57	41881	TCP	40	1337 → 41881 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
54652	2020-08-01 14:10:44	10.8.0.57	41883	147.32.83.253	1337	TCP	60	41883 → 1337 [SYN] Seq=0 Win=65535 Len=0 MSS=1361 SACK_PERM=1 TSval=271726 TSecr=0
54653	2020-08-01 14:10:44	147.32.83.253	1337	10.8.0.57	41883	TCP	40	1337 → 41883 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
54654	2020-08-01 14:10:45	10.8.0.57	41885	147.32.83.253	1337	TCP	60	41885 → 1337 [SYN] Seq=0 Win=65535 Len=0 MSS=1361 SACK_PERM=1 TSval=271831 TSecr=0
54655	2020-08-01 14:10:45	147.32.83.253	1337	10.8.0.57	41885	TCP	40	1337 → 41885 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
54656	2020-08-01 14:10:46	10.8.0.57	41887	147.32.83.253	1337	TCP	60	41887 → 1337 [SYN] Seq=0 Win=65535 Len=0 MSS=1361 SACK_PERM=1 TSval=271935 TSecr=0
54657	2020-08-01 14:10:46	147.32.83.253	1337	10.8.0.57	41887	TCP	40	1337 → 41887 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
54661	2020-08-01 14:10:47	10.8.0.57	41889	147.32.83.253	1337	TCP	60	41889 → 1337 [SYN] Seq=0 Win=65535 Len=0 MSS=1361 SACK_PERM=1 TSval=272039 TSecr=0
54662	2020-08-01 14:10:47	147.32.83.253	1337	10.8.0.57	41889	TCP	40	1337 → 41889 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
54663	2020-08-01 14:10:48	10.8.0.57	41891	147.32.83.253	1337	TCP	60	41891 → 1337 [SYN] Seq=0 Win=65535 Len=0 MSS=1361 SACK_PERM=1 TSval=272142 TSecr=0
54664	2020-08-01 14:10:48	147.32.83.253	1337	10.8.0.57	41891	TCP	40	1337 → 41891 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
54665	2020-08-01 14:10:49	10.8.0.57	41893	147.32.83.253	1337	TCP	60	41893 → 1337 [SYN] Seq=0 Win=65535 Len=0 MSS=1361 SACK_PERM=1 TSval=272247 TSecr=0
54666	2020-08-01 14:10:49	147.32.83.253	1337	10.8.0.57	41893	TCP	52	1337 → 41893 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
54667	2020-08-01 14:10:49	10.8.0.57	41893	147.32.83.253	1337	TCP	40	41893 → 1337 [ACK] Seq=1 Ack=1 Win=88064 Len=0
54668	2020-08-01 14:10:49	147.32.83.253	1337	10.8.0.57	41893	TCP	46	1337 → 41893 [PSH, ACK] Seq=1 Ack=1 Win=262656 Len=6
54669	2020-08-01 14:10:49	10.8.0.57	41893	147.32.83.253	1337	TCP	40	41893 → 1337 [ACK] Seq=1 Ack=7 Win=88064 Len=0
54670	2020-08-01 14:10:49	10.8.0.57	41893	147.32.83.253	1337	TCP	104	41893 → 1337 [PSH, ACK] Seq=1 Ack=7 Win=88064 Len=64
54671	2020-08-01 14:10:49	147.32.83.253	1337	10.8.0.57	41893	TCP	40	1337 → 41893 [ACK] Seq=7 Ack=65 Win=262400 Len=0

Figure 1. A 3-way handshake started by the phone to establish TCP connection with the C&C controller.

In Figure 1 we can see that the connection was established, but the C&C server was resetting it several times. After a while a successful 3-way handshake was performed and the connection was established, the C&C sends the next packet with following data:

```
    00000000  00 00 00 02 0b 02                                  ......
```

Figure 2. Data sent by the C&C after establishing the first TCP connection with the phone.

The phone replies with some initialization parameters such as its phone model, Android version, and other parameters in plain text.

```
00000000  00 00 00 3c 03 4e 6f 6b  69 61 20 36 2e 31 23 4e   ...<.Nok ia 6.1#N
00000010  6f 6b 69 61 23 31 30 23  75 6e 6b 6e 6f 77 6e 23   okia#10# unknown#
00000020  4e 6f 74 20 52 65 67 69  73 74 65 72 65 64 23 64   Not Regi stered#d
00000030  72 6f 69 64 6a 61 63 6b  2d 61 70 70 23 30 20 f3   roidjack -app#0 .
```

Figure 3. Data sent by the phone with initialization parameters.

Communication over 1337/TCP

After establishing the communication over port 1337/TCP, there is a sequence of three NULL (00) bytes in the data of both packets, as shown in Figure 2 and Figure 3. This sequence is followed by the hexadecimal number 0x3C, which represents the **packet length** in its decimal form, and after that the phone sends the delimiter byte 0x03. The amount for the packet length does not include bytes for the NULL sequence and the byte for the packet length. The following is an example of the bytes in hexadecimal as seen from the packet sent by the phone in the Figure 3:

```
              data length   delimiter
00000000 00 00 00 3c 03 4e 6f 6b  69 61 20 36 2e 31 23 4e   ...<.Nok ia 6.1#N
00000010  6f 6b 69 61 23 31 30 23  75 6e 6b 6e 6f 77 6e 23   okia#10# unknown#
00000020  4e 6f 74 20 52 65 67 69  73 74 65 72 65 64 23 64    Not Regi stered#d
00000030  72 6f 69 64 6a 61 63 6b  2d 61 70 70 23 30 20 f3   roidjack -app#0 .
```

Figure 4. Bytes sent from the phone to the C&C controller in one packet, including how we found the format.

In Figure 4, the actual length of the packet is 64. The byte 0x3C is 60 in a decimal format, which is exactly the length of the packet without the byte for packet length 0x3C (1 byte) and the sequence of NULL characters (3 bytes).

In the small packets of length 1 or 2, like in Figure 2 or in the heartbeat in Figure 6, there are no delimiters. Thus only packets with data of more than 2 bytes sent from the C&C and the phone over 1337/TCP has the following format:

```
{00 00 00}{data length}{delimiter}{data in plain text}
```

Figure 5. The format of packets sent from the C&C and the phone as part of the custom protocol used by the RAT.

After sending phone parameters, the phone is waiting for the command from the controller. While waiting for the command, the phone and the C&C maintain a heartbeat, which in this case is a couple of packets in both directions inside the same connection. They exchange packets every 8 seconds.

```
    0000000B  00 00 00 01 0d                                  .....
0000004A  00 00 00 01 0d                                  .....
    00000010  00 00 00 01 0d                                  .....
0000004F  00 00 00 01 0d                                  .....
    00000015  00 00 00 01 0d                                  .....
00000054  00 00 00 01 0d                                  .....
    0000001A  00 00 00 01 0d                                  .....
00000059  00 00 00 01 0d                                  .....
    0000001F  00 00 00 01 0d                                  .....
0000005E  00 00 00 01 0d                                  .....
```

Figure 6. The heartbeat between the C&C and the phone.

After some time, when it is requested by the botmaster, the C&C server sends a packet with the command to the phone. The command is ‘File Voyager’, which aims to search through the file system of the phone. In the C&C software, the command ‘File Voyager ’ looks like this:

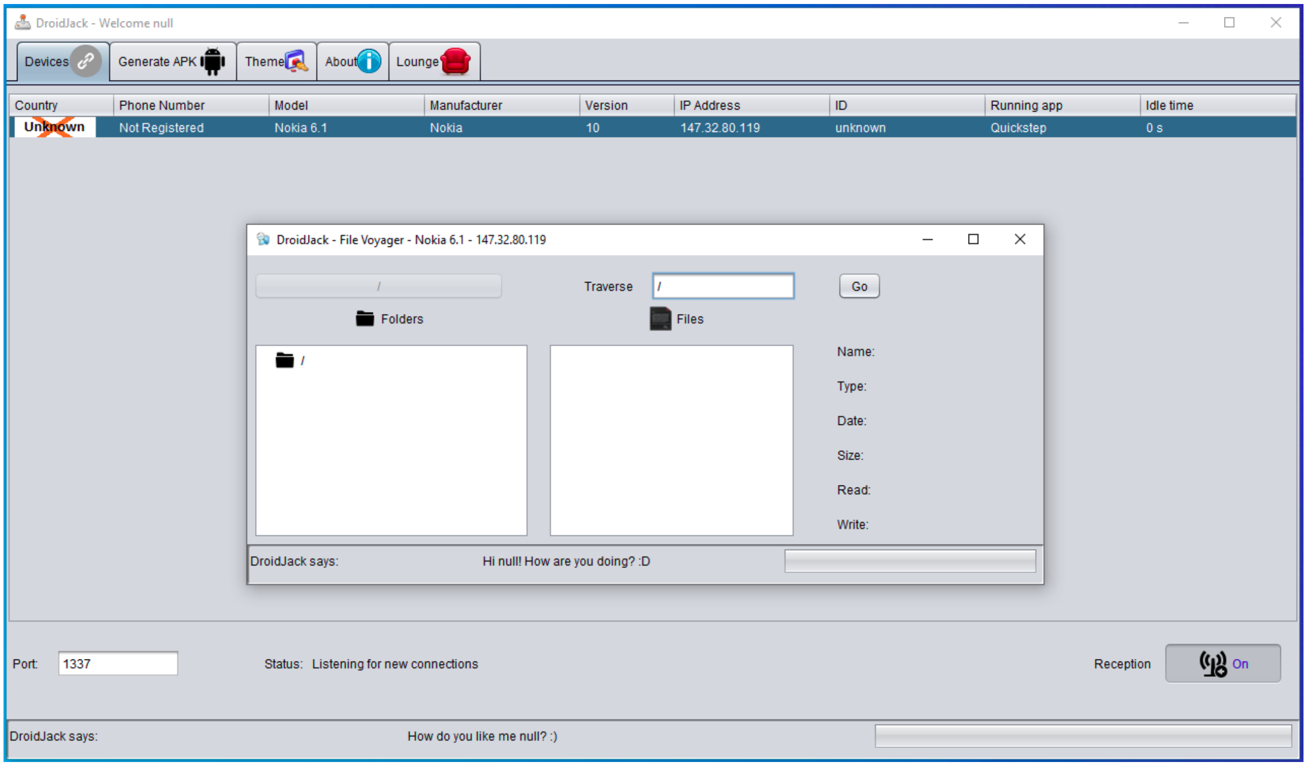

Figure 7. The command ‘File Voyager’ in DroidJack v4.4 C&C software.

Figure 8 shows an example of this order “File Voyager”, that is sent unencrypted.

```
    00000024  00 00 00 01 0d                                    .....
    00000029  00 00 00 1a 03 32 30 23  66 61 6c 73 65 23 2f 7e   .....20# false#/~
    00000039  23 30 31 39 34 30 37 34  35 36 36 37 23 b0         #0194074 5667#.
```

Figure 8. Command ‘File Voyager’ sent from the C&C after the heartbeat.

The commands from the C&C server to the phone seem to be predefined with a specific number. From Figure 8, number 20 might define the command ‘File Voyager’ and it is followed by some extra parameters (false#/~#0194074 5667#.). The character ‘#’ might be a separator between parameters. As a reply to the C&C command, the phone sends back:

```
00000063  00 00 00 15 03 6b 72 79  6f 6e 65 74 20 2d 20 6b   .....kry onet - k
00000073  65 65 70 3a 61 6c 69 76  e5                        eep:aliv .
```

Figure 9. The phone’s reply on the command ‘File Voyager’ sent by the C&C.

Communication over 1334/TCP

The reply of the phone to the C&C in Figure 9 is an acknowledgement for the received command. The actual phone reply with data is sent in a different connection. For each new command received from the C&C, the phone establishes a new TCP connection over port 1334/TCP, sends the data and closes the connection. Figure 10 shows a new connection over 1334/TCP to reply on the command in Figure 8.

tcp.stream eq 85

No.	Source	SrcPort	Destination	DstPort	Protocol	Info
54799	10.8.0.57	37842	147.32.83.253	1334	TCP	37842 → 1334 [SYN] Seq=0 Win=65535 Len=0 MSS=1
54800	147.32.83.253	1334	10.8.0.57	37842	TCP	1334 → 37842 [SYN, ACK] Seq=0 Ack=1 Win=65535
54801	10.8.0.57	37842	147.32.83.253	1334	TCP	37842 → 1334 [ACK] Seq=1 Ack=1 Win=88064 Len=0
54802	10.8.0.57	37842	147.32.83.253	1334	TCP	37842 → 1334 [PSH, ACK] Seq=1 Ack=1 Win=88064
54803	10.8.0.57	37842	147.32.83.253	1334	TCP	37842 → 1334 [FIN, ACK] Seq=5 Ack=1 Win=88064
54804	147.32.83.253	1334	10.8.0.57	37842	TCP	1334 → 37842 [ACK] Seq=1 Ack=6 Win=262656 Len=
54805	147.32.83.253	1334	10.8.0.57	37842	TCP	1334 → 37842 [FIN, ACK] Seq=1 Ack=6 Win=262656
54806	10.8.0.57	37842	147.32.83.253	1334	TCP	37842 → 1334 [ACK] Seq=6 Ack=2 Win=88064 Len=0

```
› Frame 54802: 44 bytes on wire (352 bits), 44 bytes captured (352 bits)
  Raw packet data
› Internet Protocol Version 4, Src: 10.8.0.57, Dst: 147.32.83.253
› Transmission Control Protocol, Src Port: 37842, Dst Port: 1334, Seq: 1, Ack: 1, Len: 4
⌄ Data (4 bytes)
    Data: 6e756c6c
    [Length: 4]
```

```
0000  45 00 00 2c 75 12 40 00  40 06 d4 5b 0a 08 00 39   E··,u·@· @··[···9
0010  93 20 53 fd 93 d2 05 36  67 7d 4a b6 b4 af bc 9c   · S····6 g}J·····
0020  50 18 00 ac 26 54 00 00  6e 75 6c 6c               P···&T·· null
```

Figure 10. The phone replies to the command sent by the C&C in port 1337/TCP (shown in Figure 8) with data over another connection on port 1334/TCP.

The packets in the connection 1334/TCP do not have any format as in Figure 5, the data is sent in the plain text:

```
00000000  6e 75 6c 6c                                      null
```

Figure 11. Packet sent from the phone to the controller over 1334/TCP.

Communication over 1337/UDP

Even though there is a heartbeat over port 1337/TCP, the phone sends UDP packets to the C&C over port 1337 every 20 seconds.

udp.port ==1337

No.	Time	Source	SrcPort	Destination	DstPort	Protocol	Info	Length
54753	16:11:02	10.8.0.57	41299	147.32.83.253	1337	UDP	41299 → 1337 Len=34	62
54771	16:11:23	10.8.0.57	44048	147.32.83.253	1337	UDP	44048 → 1337 Len=34	62
54781	16:11:43	10.8.0.57	38401	147.32.83.253	1337	UDP	38401 → 1337 Len=34	62
54815	16:12:03	10.8.0.57	45927	147.32.83.253	1337	UDP	45927 → 1337 Len=34	62
54826	16:12:23	10.8.0.57	40713	147.32.83.253	1337	UDP	40713 → 1337 Len=34	62
54837	16:12:43	10.8.0.57	40365	147.32.83.253	1337	UDP	40365 → 1337 Len=34	62
54881	16:13:03	10.8.0.57	48133	147.32.83.253	1337	UDP	48133 → 1337 Len=34	62
54954	16:13:23	10.8.0.57	38992	147.32.83.253	1337	UDP	38992 → 1337 Len=34	62
54987	16:13:43	10.8.0.57	43793	147.32.83.253	1337	UDP	43793 → 1337 Len=34	62
55003	16:14:03	10.8.0.57	42748	147.32.83.253	1337	UDP	42748 → 1337 Len=34	62
55090	16:14:23	10.8.0.57	43126	147.32.83.253	1337	UDP	43126 → 1337 Len=34	62

```
› Frame 54881: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
  Raw packet data
› Internet Protocol Version 4, Src: 10.8.0.57, Dst: 147.32.83.253
› User Datagram Protocol, Src Port: 48133, Dst Port: 1337
⌄ Data (34 bytes)
    Data: 5544504d5f464f524547524f554e443a756e6b6e6f776e2e…
    [Length: 34]
```

```
0000  45 00 00 3e cb 3c 40 00  40 11 7e 14 0a 08 00 39   E··>·<@· @·~····9
0010  93 20 53 fd bc 05 05 39  00 2a 2b 26 55 44 50 4d   · S····9 ·*+&UDPM
0020  5f 46 4f 52 45 47 52 4f  55 4e 44 3a 75 6e 6b 6e   _FOREGRO UND:unkn
0030  6f 77 6e 2e 2c 51 75 69  63 6b 73 74 65 70         own.,Qui ckstep
```

Figure 12. UDP packets from the phone to the C&C server sent every 20 seconds over port 1337/UDP.

The data inside UDP packets is in the plain text:

```
00000000  55 44 50 4d 5f 46 4f 52  45 47 52 4f 55 4e 44 3a   UDPM_FOR EGROUND:
00000010  75 6e 6b 6e 6f 77 6e 2e  2c 51 75 69 63 6b 73 74   unknown. ,Quickst
00000020  65 70                                              ep
```

Figure 13. Example data inside the UDP packets on port 1337/UDP sent from the phone to the controller.

Long Connections

If we open the Conversations -> statistics -> TCP menu in Wireshark, as shown in Figure 14, a lot of connections between the phone and the controller are over port 1334/TCP (new C&C - new connection) and only a few are over 1337/TCP. The connections over 1337/TCP are usually long, e.g. 1548.2056 seconds (approximately 40 minutes) or 1413.3981 seconds (approximately 31 minutes). This indicates that the connections between the phone and the controller are kept for long periods of time in order to answer fast.

Ethernet | IPv4 · 50 | IPv6 | TCP · 214 | UDP · 304

Address A	Port A	Address B	Port B	Packets	Duration
10.8.0.57	42059	147.32.83.253	1337	780	1548.2056
10.8.0.57	41893	147.32.83.253	1337	730	1413.3981
10.8.0.57	41883	147.32.83.253	1337	2	0.0008
10.8.0.57	41887	147.32.83.253	1337	2	0.0008
10.8.0.57	41881	147.32.83.253	1337	2	0.0007
10.8.0.57	41885	147.32.83.253	1337	2	0.0007
10.8.0.57	41889	147.32.83.253	1337	2	0.0006
10.8.0.57	41891	147.32.83.253	1337	2	0.0005
10.8.0.57	38038	147.32.83.253	1334	33	30.0918
10.8.0.57	37932	147.32.83.253	1334	8	1.5981
10.8.0.57	38010	147.32.83.253	1334	8	1.5777
10.8.0.57	37874	147.32.83.253	1334	8	0.8858
10.8.0.57	38092	147.32.83.253	1334	8	0.8760
10.8.0.57	37928	147.32.83.253	1334	11	0.7056
10.8.0.57	37852	147.32.83.253	1334	59	0.5474
10.8.0.57	37890	147.32.83.253	1334	35	0.5463
10.8.0.57	37892	147.32.83.253	1334	35	0.4804
10.8.0.57	37954	147.32.83.253	1334	26	0.4384
10.8.0.57	38084	147.32.83.253	1334	10	0.4013
10.8.0.57	37914	147.32.83.253	1334	59	0.3839
10.8.0.57	37930	147.32.83.253	1334	26	0.3087
10.8.0.57	38090	147.32.83.253	1334	8	0.2916
10.8.0.57	37886	147.32.83.253	1334	35	0.2512
10.8.0.57	37952	147.32.83.253	1334	27	0.2326
10.8.0.57	38008	147.32.83.253	1334	51	0.2196
10.8.0.57	37920	147.32.83.253	1334	35	0.2008
10.8.0.57	37868	147.32.83.253	1334	8	0.1814
10.8.0.57	37946	147.32.83.253	1334	26	0.1523
10.8.0.57	38056	147.32.83.253	1334	10	0.0954
10.8.0.57	37850	147.32.83.253	1334	8	0.0641
10.8.0.57	38040	147.32.83.253	1334	8	0.0407
10.8.0.57	38096	147.32.83.253	1334	8	0.0260

Figure 14. Top connections between the phone and the controller from Wireshark -> Statistics -> Conversations -> TCP. It can be noted the long duration of the main connections.

Detecting C&C using Slips

Slips is a Python Intrusion Detection and Prevention system that uses machine learning to detect malicious behaviours in the network traffic of the devices. Slips is an open-source tool and can be installed from here.

After Slips is run on the DroidJack v4.4 packet capture, Slips creates a profile per each IP that appeared in the traffic. Each profile contains flows sent from this IP. Each flow is described with a specific letter which description can be found here. Considering that, Slips detects the C&C channel over 1334/TCP. The behavioral model of the connection between the phone and C&C is in Figure 15, and Slips' machine learning module called LSTM detecting C&C channel is shown in Figure 16.

```
147.32.83.253:1334:tcp  11+R,r.Y,r+B+s+H+y+Y,Y+r+y+r,a,r+y,s,H,H
                        ,H,y+s,R.R,r,a,a,R,r+I+r,R.R,R*R,A,A,r+r
                        ,A,s,r+s,B,
```

Figure 15. Behavioral model of the connection between the phone and C&C over 1334/TCP.

```
Evidence
outTuple:147.32.83.253:1334:tcp:C&C channels detection 1,50,LSTM C&C channels detection, score: 0.7664518
```

Figure 16. Alert from slips that it detects a C&C channel over port 1334/TCP using a machine learning LSTM neural network. The LSTM uses the letters shown in Figure 15.

Slips did not detect periodic connection over 1337/UDP because the LSTM module focuses on TCP. But from the behavioral model of the connections over 1337/UDP shown in Figure 17, we can conclude that the model is periodic and most of connections are of a small size.

```
147.32.83.253:1337:udp  11,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a
                        ,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a
                        ,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a
                        ,a,a,a,a,a,a,a,a,a,r+r,a,a,r,r,a,a,a,a,a
                        ,a,a,a,a,a,a,a,a,a,a,a,a,r,r,r,a,a,a,a,a
                        ,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a
                        ,a,a,r.r,A,a,a,r,r,r,a,a,a,a,a,a,a,a,a,a
                        ,a,a,
```

Figure 17. Behavioral model created by Slips for the connection between phone and C&C over 1337/UDP.

Conclusion

In this blog, we have analyzed the network traffic from a phone infected with DroidJack v4.4 RAT. We were able to decode its connection and found the distinctive features as long duration or heartbeat. The DroidJack v4.4 RAT 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).

- Some connections over port 1337/TCP between the phone and the controller are long, i.e. more than 30 minutes.
- There is a heartbeat between the controller and the phone over 1337/TCP.
- Packets sent from the phone and the C&C over port 1337/TCP have a form of **{00 00 00}{data length}{delimiter}{data in plain text}**.
- A new connection over 1334/TCP is established when a new command is received from the C&C.
- The phone sends UDP packets to the C&C every 20 seconds.
- Packets sent from the phone to the C&C over 1334/TCP and 1337/UDP are in plain text.

Biographies

Kamila Babayeva

Kamila Babayeva is a 20 years old and third-year bachelor student in the Computer Science and Electrical Engineering program at the Czech Technical University in Prague. She is a researcher in the Civilsphere project, a project dedicated to protecting civil organizations and individuals from targeted attacks. Her research focuses on helping people and protecting their digital rights by developing free software based on machine learning. Initially, she worked as a junior Malware Reverser. Currently, Kamila leads the development of the Stratosphere Linux Intrusion Prevent System (Slips), which is used to protect the civil society in the Civilsphere lab.

Sebastian Garcia

Sebastian Garcia is a malware researcher and security teacher with experience in applied machine learning on network traffic. He founded the Stratosphere Lab, aiming to do impactful security research to help others using machine learning. He believes that free software and machine learning tools can help better protect users from abuse of our digital rights. He researches on machine learning for security, honeypots, malware traffic detection, social networks security detection, distributed scanning (dnmap), keystroke dynamics, fake news, Bluetooth analysis, privacy protection, intruder detection, and microphone detection with SDR (Salamandra). He co-founded the MatesLab hackspace in Argentina and co-founded the Independent Fund for Women in Tech. @eldracote. https://www.researchgate.net/profile/Sebastian_Garcia6