Network Protocols: Analysing a PCAP file

Trevor Wachaga

Contents

Introduction

The following report details the reverse engineering process of a network protocol, whereby its structure and underlying logic has been uncovered.

A PCAP file that contained captured network traffic was analysed using Wireshark, and lots of information in terms of hosts, protocols, and conversation history was found and discussed.

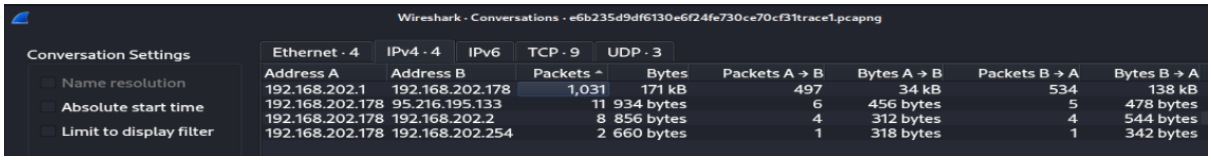
Through using filters, Wireshark interface tools and other software tools e.g. CyberChef, the network was able to be analysed thoroughly, and this report details the step-by-step walkthrough and any key findings.

Overall Information Flow

Hosts, Protocols and Ports

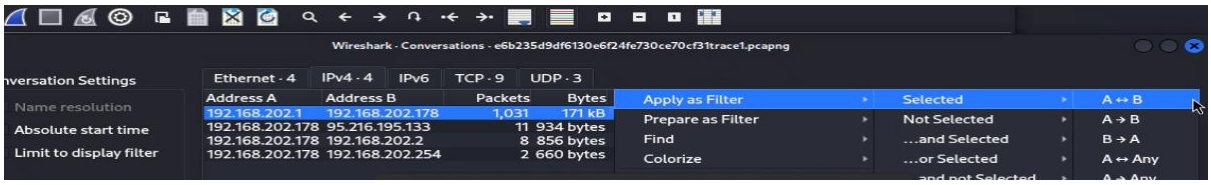
The first steps taken when analysing the PCAP file was looking for host information and conversation history. Furthermore, viewing the section called 'Conversations' under 'Statistics' was done first.

The screenshot below shows the overview of the network conversations:

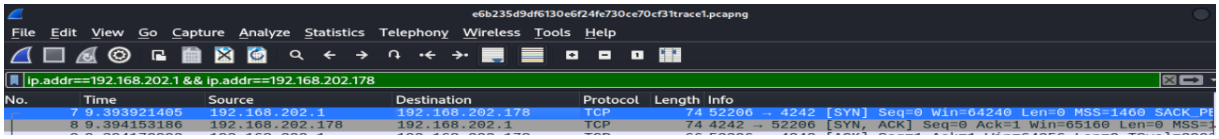


The conversations between 192.168.202.1 and 192.168.202.178 had the most packets sent (1031), and this was where the main interaction between the client and server occurred. Furthermore, from the data it could be inferred that 192.168.202.1 was the client as 192.168.202.178 had sent the most data (138kB) during their interactions, whereas 192.168.202.1 only sent 34kB. Additionally, 192.168.202.178 had conversations with other hosts, inferring that it was the server for 192.168.202.1.

To confirm this, a filter was applied so more information regarding the two IP addresses only was displayed.



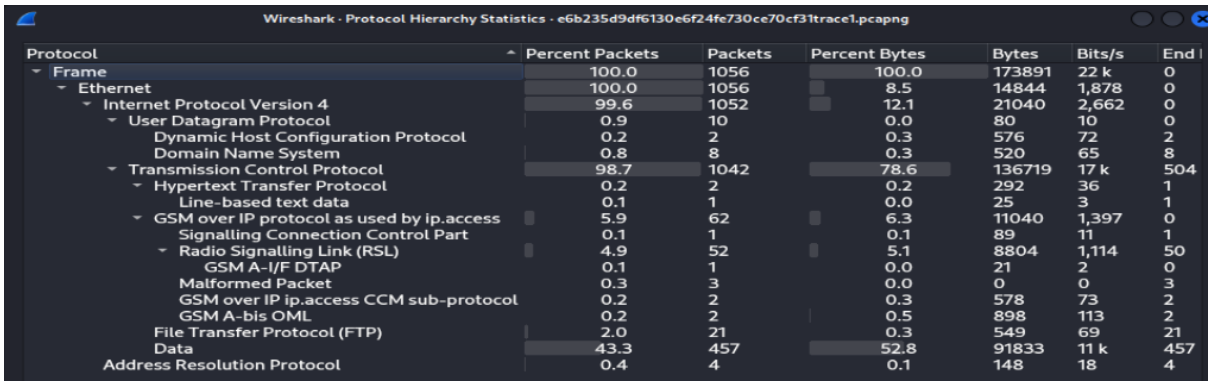
Furthermore, below shows that 192.168.202.1 initiated the connection as it sent a packet with the SYN flag set and then 192.169.202.178 sent an SYN-ACK flag in response acknowledging the request. These control flags are important when analysing packets as it let you know immediately what they are doing, and it confirmed which IP address was the client.



This confirmed that:

192.168.202.1 = Client
192.168.202.178 = Server

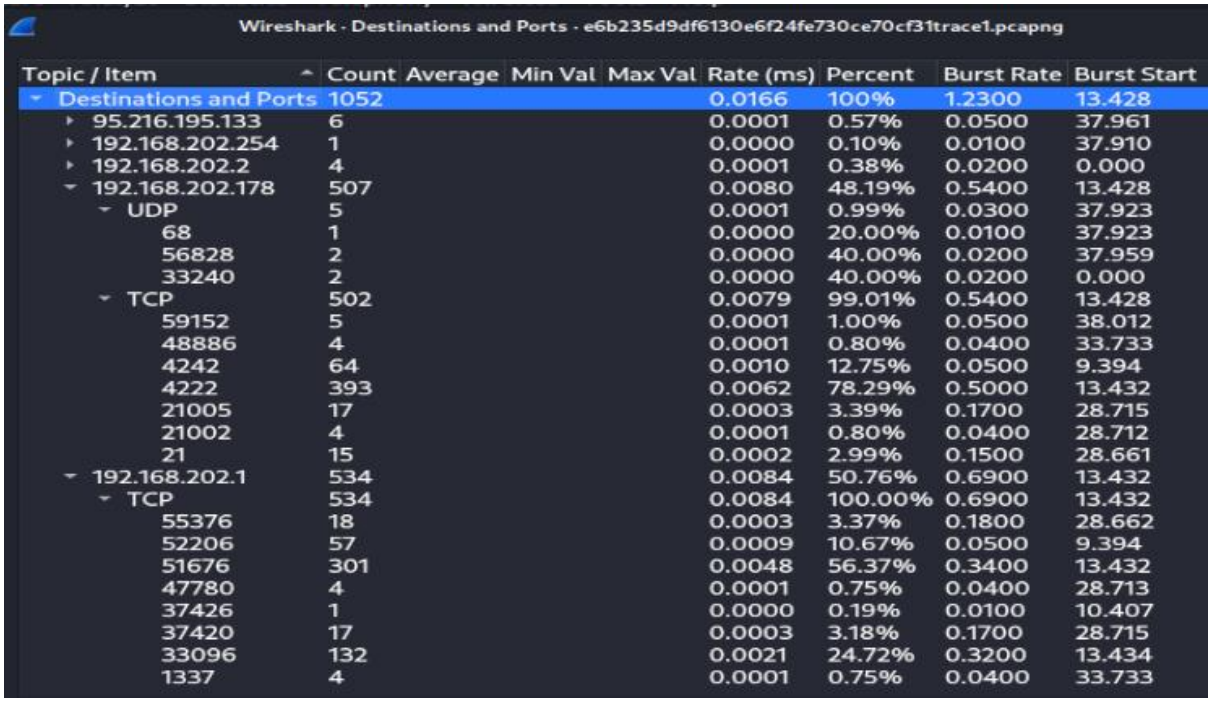
Next was figuring out that protocols used within the entirety of the PCAP file, so viewing ‘Protocol Hierarchy’ under statistics was done.



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End
Frame	100.0	1056	100.0	173891	22 k	0
Ethernet	100.0	1056	8.5	14844	1,878	0
Internet Protocol Version 4	99.6	1052	12.1	21040	2,662	0
User Datagram Protocol	0.9	10	0.0	80	10	0
Dynamic Host Configuration Protocol	0.2	2	0.3	576	72	2
Domain Name System	0.8	8	0.3	520	65	8
Transmission Control Protocol	98.7	1042	78.6	136719	17 k	504
Hypertext Transfer Protocol	0.2	2	0.2	292	36	1
Line-based text data	0.1	1	0.0	25	3	1
GSM over IP protocol as used by ip.access	5.9	62	6.3	11040	1,397	0
Signalling Connection Control Part	0.1	1	0.1	89	11	1
Radio Signalling Link (RSL)	4.9	52	5.1	8804	1,114	50
GSM A-I/F DTAP	0.1	1	0.0	21	2	0
Malformed Packet	0.3	3	0.0	0	0	3
GSM over IP ip.access CCM sub-protocol	0.2	2	0.3	578	73	2
GSM A-bis OML	0.2	2	0.5	898	113	2
File Transfer Protocol (FTP)	2.0	21	0.3	549	69	21
Data	43.3	457	52.8	91833	11 k	457
Address Resolution Protocol	0.4	4	0.1	148	18	4

The image above shows the protocols, and their percentage/packets used. There were many, but the main networking protocols were: DHCP, FTP, DNS, HTTP, ARP, IP, UDP, TCP and ETH. Moreover, TCP was used the most as it was the main protocol used for communication between the client and server as it “ensures the reliable transmission of data between devices on a network” (Lutkevich, 2021).

Another element that was necessary looking at was the ports used. Again, using statistics, under ‘IPv4 statistics’, ‘Destinations and Ports’ was looked at and below shows the ports used by the client and server.



Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
Destinations and Ports	1052				0.0166	100%	1.2300	13.428
95.216.195.133	6				0.0001	0.57%	0.0500	37.961
192.168.202.254	1				0.0000	0.10%	0.0100	37.910
192.168.202.2	4				0.0001	0.38%	0.0200	0.000
192.168.202.178	507				0.0080	48.19%	0.5400	13.428
UDP	5				0.0001	0.99%	0.0300	37.923
68	1				0.0000	20.00%	0.0100	37.923
56828	2				0.0000	40.00%	0.0200	37.959
33240	2				0.0000	40.00%	0.0200	0.000
TCP	502				0.0079	99.01%	0.5400	13.428
59152	5				0.0001	1.00%	0.0500	38.012
48886	4				0.0001	0.80%	0.0400	33.733
4242	64				0.0010	12.75%	0.0500	9.394
4222	393				0.0062	78.29%	0.5000	13.432
21005	17				0.0003	3.39%	0.1700	28.715
21002	4				0.0001	0.80%	0.0400	28.712
21	15				0.0002	2.99%	0.1500	28.661
192.168.202.1	534				0.0084	50.76%	0.6900	13.432
TCP	534				0.0084	100.00%	0.6900	13.432
55376	18				0.0003	3.37%	0.1800	28.662
52206	57				0.0009	10.67%	0.0500	9.394
51676	301				0.0048	56.37%	0.3400	13.432
47780	4				0.0001	0.75%	0.0400	28.713
37426	1				0.0000	0.19%	0.0100	10.407
37420	17				0.0003	3.18%	0.1700	28.715
33096	132				0.0021	24.72%	0.3200	13.434
1337	4				0.0001	0.75%	0.0400	33.733

Plenty of ports were used but one that stood out was port 21, as it is a “well-known port primarily used for FTP” (SFTPCloud, 2024). Furthermore, it highlighted that file sharing may have occurred between the client and server, and this was something to look out for.

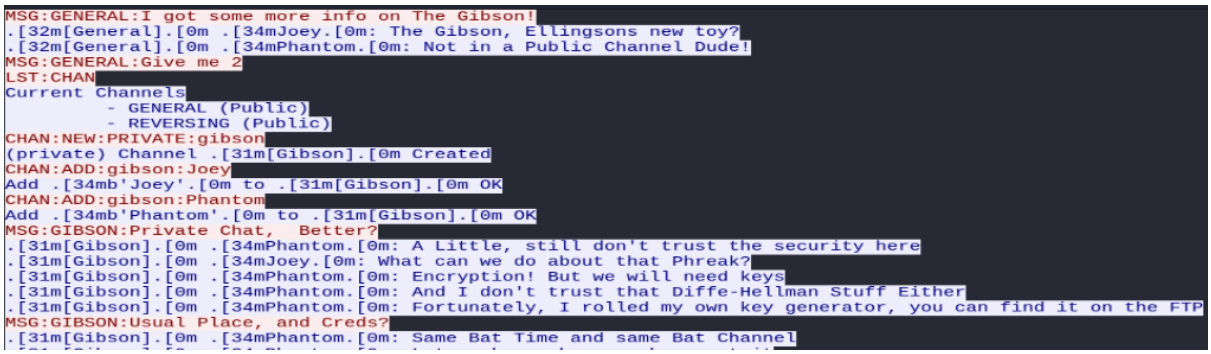
Now that all of this information was found, reversing and trying to understand the flow of the interaction could be done.

Flow

As stated, the client interacted with the server first – furthermore this packet was investigated first by following the TCP stream. Below shows the stream:

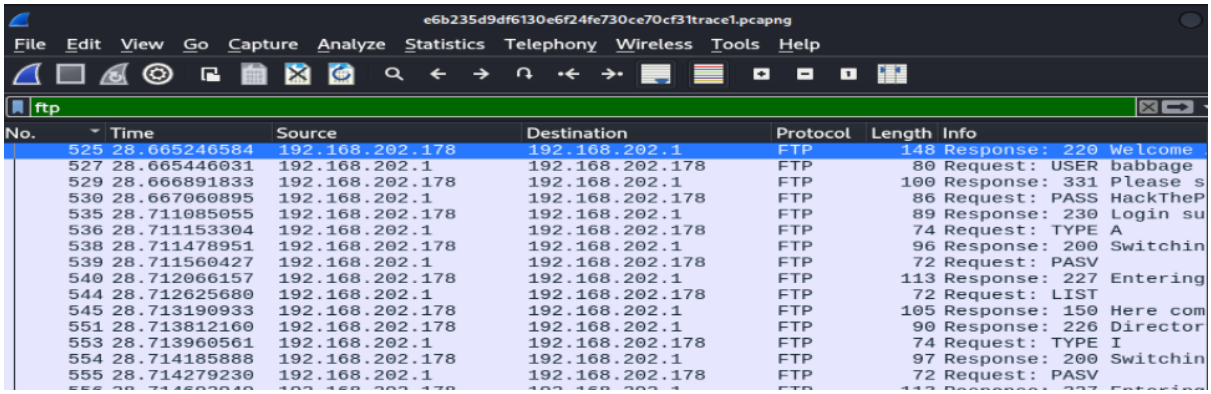
The stream showed a captured session from a chat app/service, and it showed conversations between different users – emulating a client-server communication setup, whereby the red highlighted text came from the client, and the blue highlighted text came from the server. The client is known as Crash Override and the other users in the chat service are Joey and Phantom.

An interesting part of the conversation is pictured below:



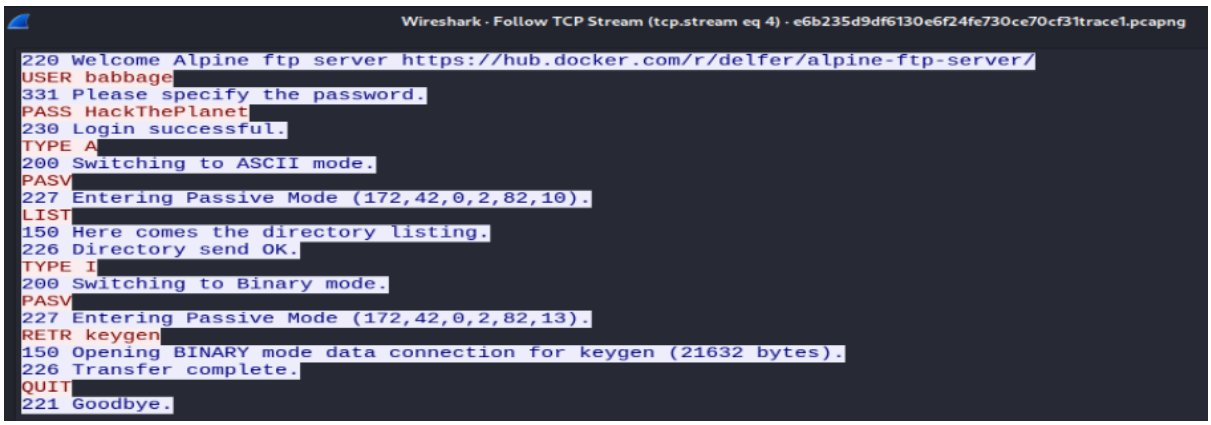
This discussion included the client creating a private channel in order to talk about 'The Gibson'. However, they did not trust the security of the channel, so Phantom wanted to use encryption. In order to generate the keys, a generator that could be found on the FTP server was used.

Furthermore, this was vital information, and the packet relating to the FTP server needed to be found. Below shows the output of using an FTP filter in Wireshark:



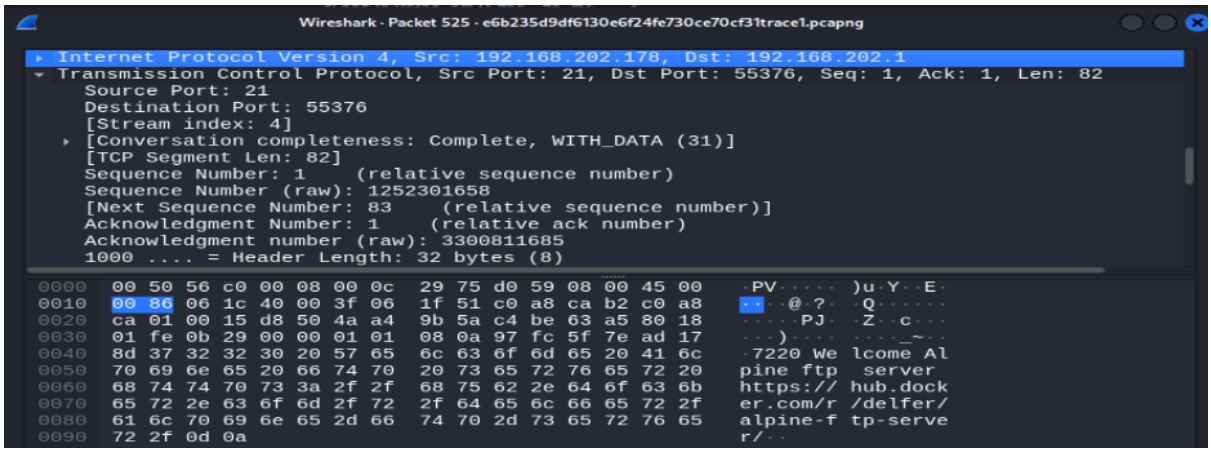
The image displays all packets that used the FTP protocol, and several requests and responses from the client and server respectively.

To find out more information, following the stream was done:

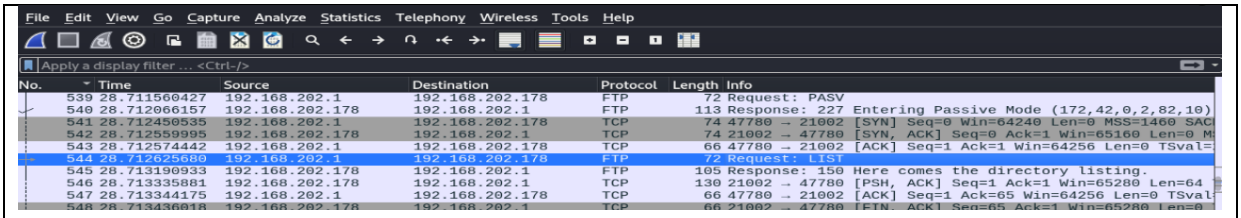


The TCP stream showed an interaction between the client and the FTP server, whereby they successfully logged in and retrieved a file called keygen.

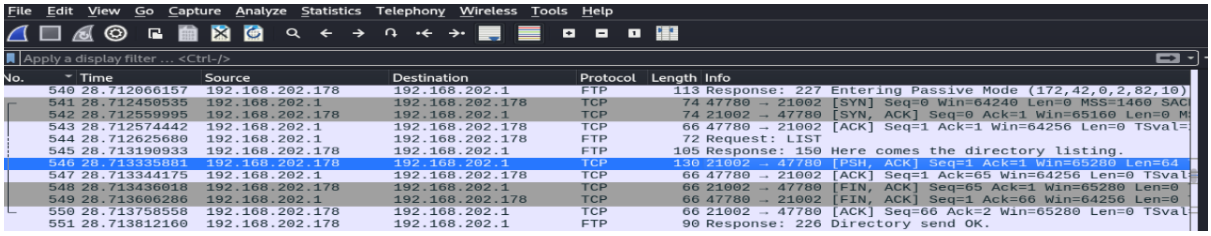
Below confirms that port 21 was used for the file transfer:



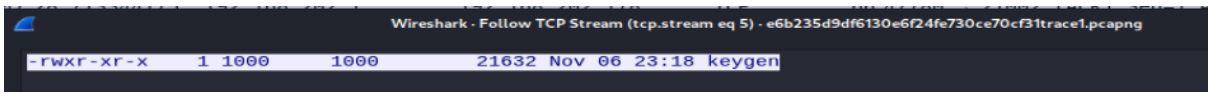
To confirm there was actually a keygen file, below shows the process done to find the file information:



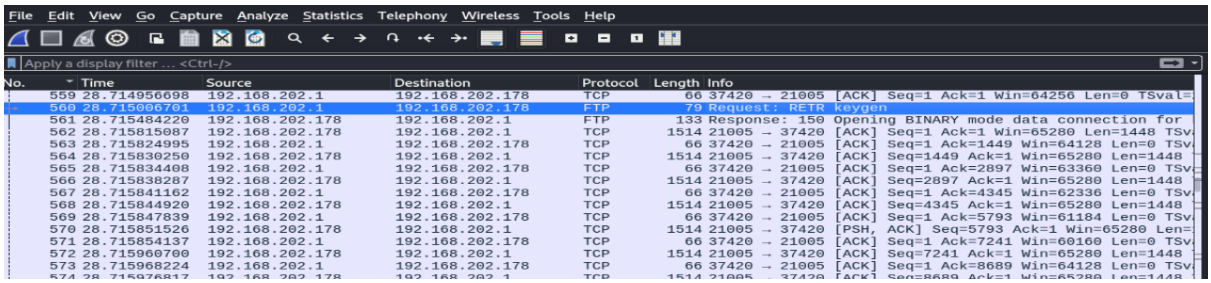
Above shows the FTP packets where the user listed the directory to see the keygen file



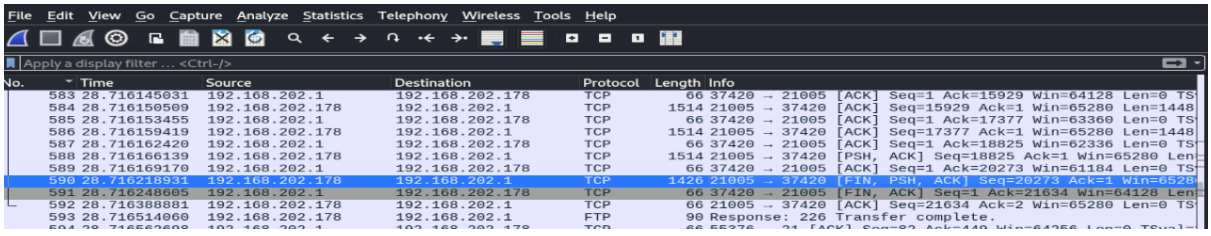
Above shows the 5 TCP packets (no. 546, 547, 548, 549, 550) that dealt with listing the file. Furthermore because of the control flag of PSH, it was clear that this packet had the data to be sent to the application layer.



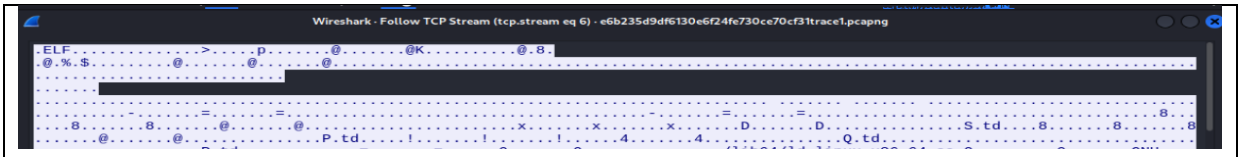
By following the stream of those 5 TCP packets, above shows the actual keygen file and its information.



Above shows that the client downloaded a copy of the keygen file. Below the FTP packet, there was a large stream of TCP packets that acknowledged the request.



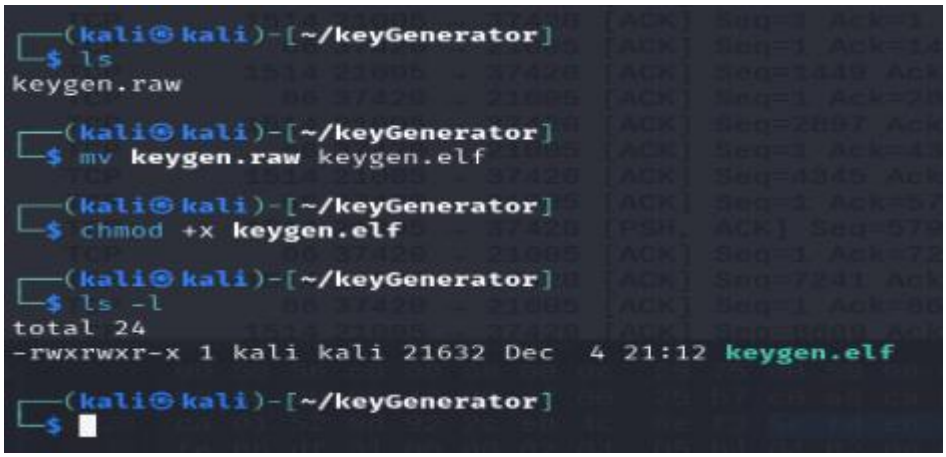
Above shows the TCP packet with the control flag of FIN, highlighting that the transfer completed and that the user successfully retrieved the file.



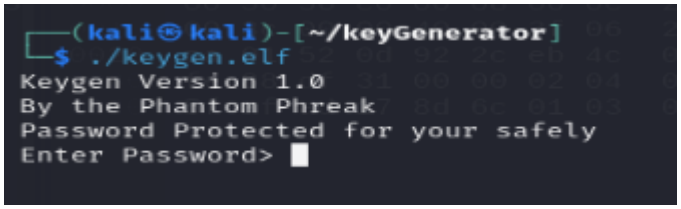
Above shows the stream of that TCP packet and it shows an .ELF file of the keygen generator that was successfully retrieved.

After the FTP server closed, the previous conversation in the private channel continued, as seen below:

This meant that the keygen file could run and below shows the process of that:

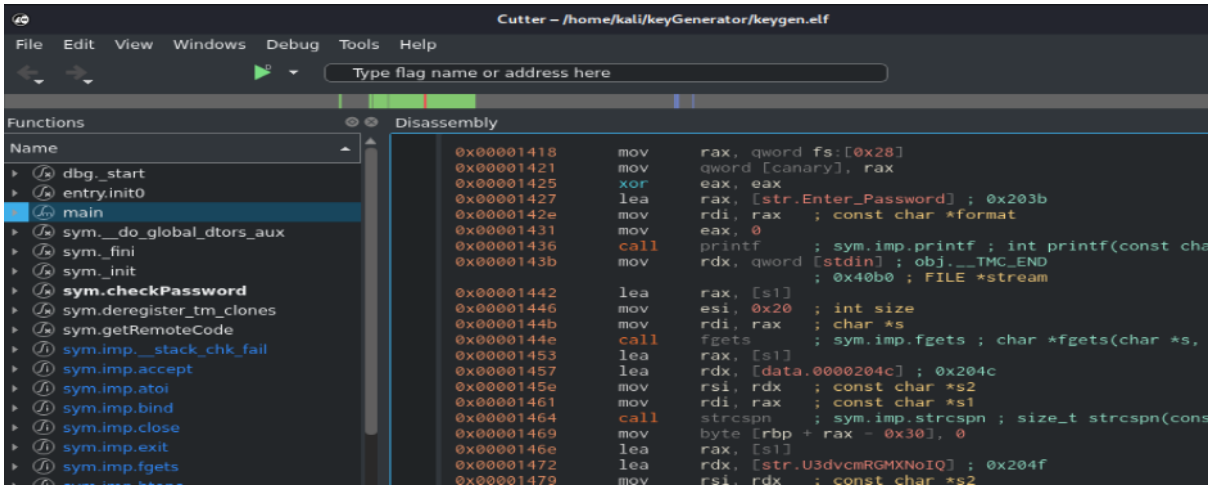


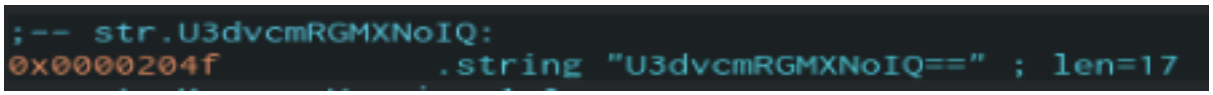
- The raw keygen file was saved from the stream
- The file was then changed to an .elf file
- The permissions of the file were changed to make it executable



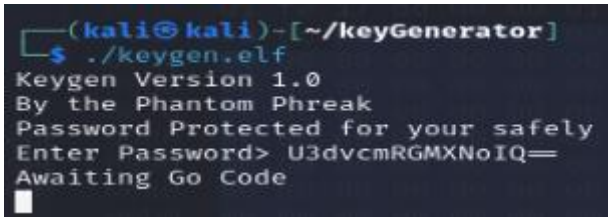
The keygen file was able to run and it asked for a password

To find the password a bit of static analysis was done. Below shows an image of the checkPassword function in the keygen file, and the hardcoded password too.





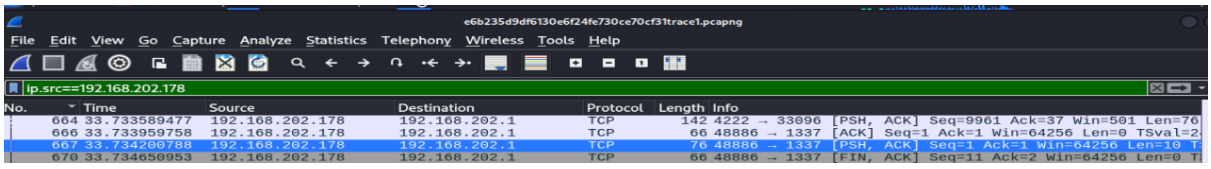
Furthermore, the password was 'U3dvcmRGMXNoIQ==' and this was inputted:



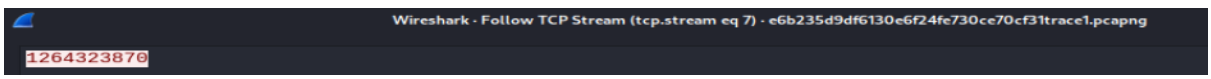
It was successful and the output shows it was waiting for the 'Go Code'.

Going back to the conversation, Phantom stated they would send the key to Crash so they can input it into the generator:

Furthermore, to find the key, more analysis of the packets had to be done. A filter was used so that the IP source was the IP address of the server as that is where the key would be sent from. Below shows a packet from server IP address with a control flag of PSH, meaning that it contains data.



Below shows an image of the TCP stream (no. 667) that was followed. The packet shows the key that was sent from Phantom to Crash:



Analysing packet 667 showed that the source port was 1337, hinting that when the keygen generator is running, this port would listen, and a connection could be made in order to input this key.

Below shows the process of this occurring:

```
┌──(kali㉿kali)-[~/6052-Labs]
└─$ netstat -tuln | grep 1337
tcp        0      0 0.0.0.0:1337            0.0.0.0:*               LISTEN
```

While keygen was running, port 1337 was open

```
┌──(kali㉿kali)-[~/6052-Labs]
└─$ telnet 127.0.0.1 1337
Trying 127.0.0.1 ...
Connected to 127.0.0.1.
Escape character is '^]'.
1264323870
```

A successful connection was made and the key sent from Phantom was inputted.

```
┌──(kali㉿kali)-[~/keyGenerator]
└─$ ./keygen.elf
Keygen Version 1.0
By the Phantom Phreak
Password Protected for your safely
Enter Password> U3dvcmRGMXNoIQ==
Awaiting Go Code
1264323870

Numerial Go Code is 1264323870
27 : 82
53 : 114
8 : 56
61 : 122
13 : 68
25 : 80
19 : 74
41 : 102
12 : 67
52 : 113
51 : 112
7 : 55
40 : 101
15 : 70
34 : 89
33 : 88
Key is Rr8zDPJfCqp7eFYX
```

From the key inputted, another key (Rr8zDPJfCqp7eFYX) was made

According to the conversation, the key that was generated (Rr8zDPJfCqp7eFYX) would be used for encryption/decryption:

```
MSG:GIBSON:What Now
.[31m[Gibson].[0m .[34mPhantom.[0m: Easy Enough
.[31m[Gibson].[0m .[34mPhantom.[0m: Its a well known Crypto Algorithm in CBC Mode
.[31m[Gibson].[0m .[34mPhantom.[0m: We will have a separate stream for Reader and Writer.
.[31m[Gibson].[0m .[34mPhantom.[0m: The Key you have just generated is the Key we will for both Reader and Writer.
```

Moreover, the end of their conversation occurred, and below shows some of their encrypted messages, as well as normal messages detailing their IVs (Initialization vectors):

```
.[31m[Gibson].[0m .[34mPhantom.[0m: We Just Need a set of IV
.[31m[Gibson].[0m .[34mJoey.[0m: Hack The Planet
.[31m[Gibson].[0m .[34mPhantom.[0m: Should work, if we remove white space and add a character to make it 16 long
.[31m[Gibson].[0m .[34mPhantom.[0m: Lets use 'Hack_The_Planet!' for the IV when sending to me
.[31m[Gibson].[0m .[34mPhantom.[0m: Lets and 'CuriosityIsCrime' for the IV when receiving from me
.[31m[Gibson].[0m .[34mPhantom.[0m: Remember its 1-1 So I will translate between you
.[31m[Gibson].[0m .[34mPhantom.[0m: Good to Go?
MSG:GIBSON:Yes
.[31m[Gibson].[0m .[34mPhantom.[0m: R8Ucw6AfL1/voZJEIAd0D7tWfzBffs8J4+ICzHcS9Bugg3FePPzUjF29YDD61OMj
MSG:GIBSON:g7V+JqhvbAy06Y9Y964X8Y7QW0jwxxuvb/hFutJqXCo=
.[31m[Gibson].[0m .[34mPhantom.[0m: rezQlf8RCF2jGbLKogsXPBeIY4No2aK21h+u3pWpnM4=
MSG:GIBSON:XB/GZCmWjTHHvltD9QxYTA==
.[31m[Gibson].[0m .[34mPhantom.[0m: V0hgWmH871koOV896s6qkFOn1kGBfiHQc8k/tW/hGYQ=
QUIT
Input of b'QUIT' not understood
```

Encryption Section

The messages that were encrypted are as followed:

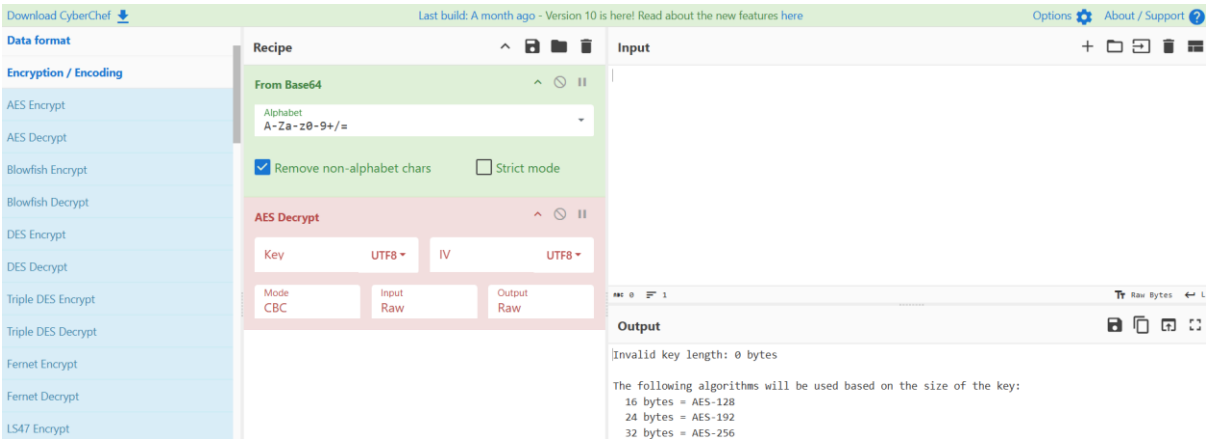
1.	Phantom	R8Ucw6AfL1/voZJEIAd0D7tWfzBffs8J4+ICzHcS9Bugg3FePPzUjF29YDD61OMj
2.	Crash	g7V+JqhvbAy06Y9Y964X8Y7QW0jwxxuvb/hFutJqXCo=
3.	Phantom	rezQlf8RCF2jGbLKogsXPBeIY4No2aK21h+u3pWpnM4=
4.	Crash	XB/GZCmWjTHHvltD9QxYTA==
5.	Phantom	V0hgWmH871koOV896s6qkFOn1kGBfiHQc8k/tW/hGYQ=

- When reading Phantom's messages, 'CuriosityIsCrime' should be used for the IV
- When reading Crash's messages, 'Hack_The_Planet!' Should be used for the IV
- When decrypting, 'Rr8zDPJfCqp7eFYX' is the key

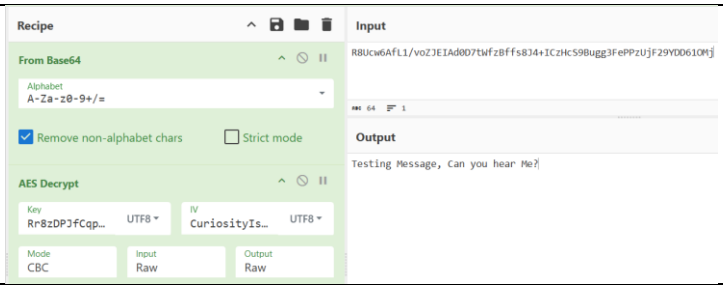
For the encryption algorithm, AES encryption in Cipher Block Chaining (CBC) was used. This is because the text is encoded in base64, and the encryption requires an IV, which is the case for CBC.

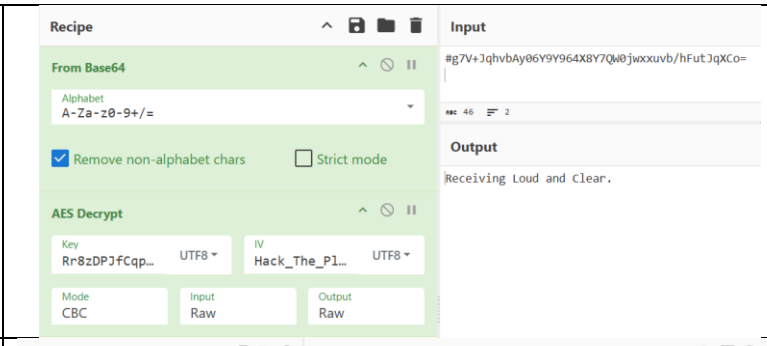
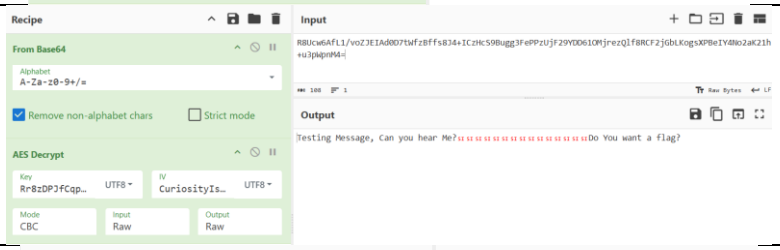
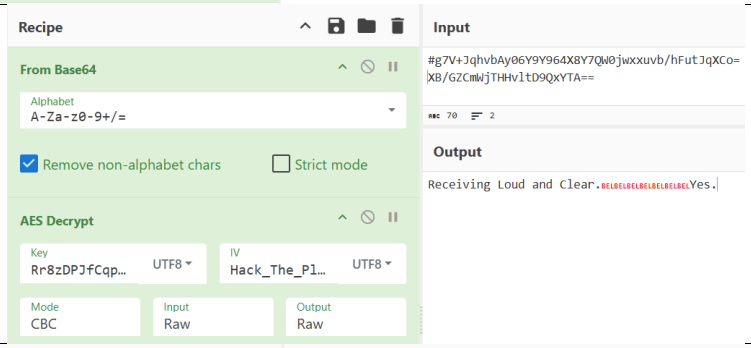
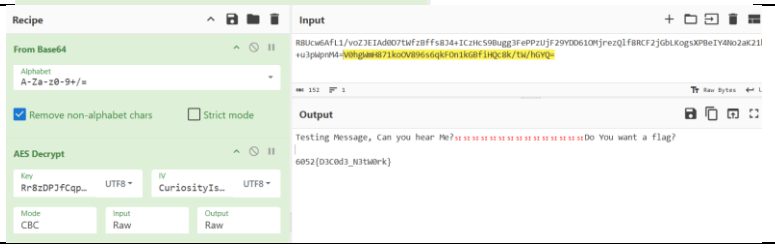
With all of this information now known, as well as knowing the generated key, it was time to decrypt.

To decrypt, cyberchef was used; for the 'recipe', both the 'AES Decrypt' and 'From Base64' parameters had to be used, as shown below:



Therefore, it was time to decrypt, and the table below shows the decrypted text:

1. Phantom to Crash		"Testing Message, Can you hear Me?"

2. Crash to Phantom		"Receiving Loud and Clear."
3. Phantom to Crash		"Do You want a flag?"
4. Crash to Phantom		"Yes."
5. Phantom to Crash		"6052{D3C0d3_N3tW0rk}"

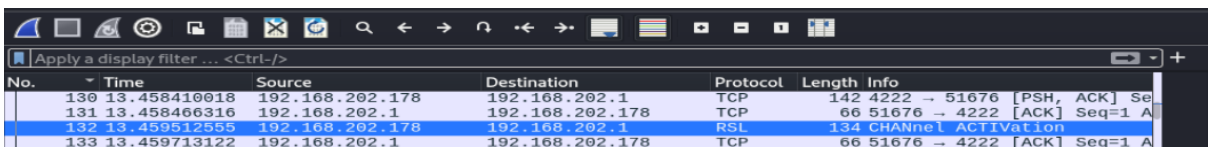
Furthermore, this shows that the conversation history could be decrypted, and a flag was successfully received.

Flag: **6052{D3C0d3_N3tW0rk}"**

Other highlights:

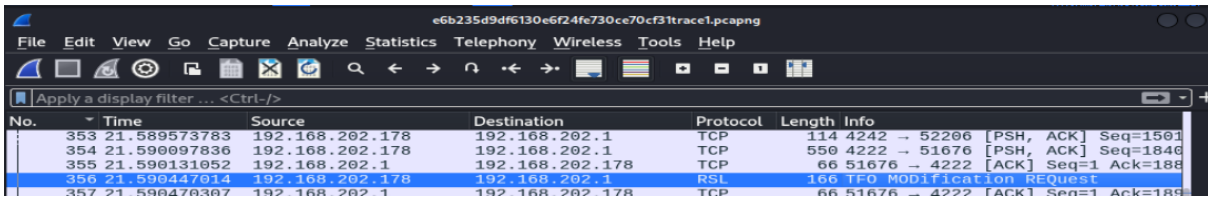
RSL

This protocol was used for managing the channels. There were multiple channels used in the communication found, and RSL was used to do this. As you can see below, when the channel was used, this occurred:



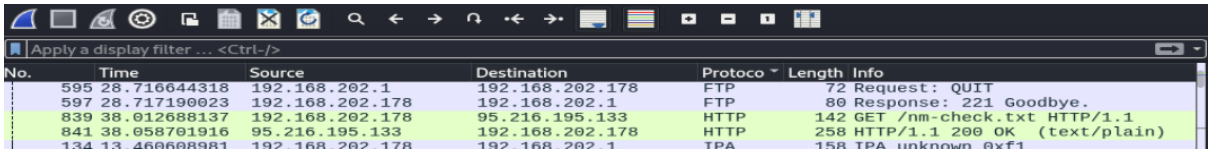
This highlight RSL was used for the channel that was used between Crash, Joey and Phantom.

Furthermore, when the private channel was created, a modification request occurred:



This further highlights how useful RSL was.

HTTP
This protocol was rarely used but it was used for the server to send a request to another server to fetch a file:



Following the HTTP stream, showed that the request was to check the Network Manager's status, below shows the output:

Conclusion

As you can see, this report covered the processes taken to reverse engineer the network protocol and the analysis of the network traffic. Through screenshots and descriptions, the step-by-step walkthrough was shown, and the successful reconstruction of findings was accomplished.

References

Lutkevich, B. (2021, October). *What is TCP (Transmission Control Protocol)?* TechTarget. https://www.techtarget.com/searchnetworking/definition/TCP

SFTPCloud. (2024). *What is Port 21 Used For? - SFTPCloud*. https://sftpcloud.io/learn/ftp/what-is-port-21-used-for