

# Network Protocols: Analysing a PCAP file

Trevor Wachaga

## Contents

Introduction.....	3
Overall Information Flow .....	3
Hosts, Protocols and Ports.....	3
Flow.....	5
Encryption Section .....	11
Other highlights: .....	12
Conclusion.....	13
References .....	13

# 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:

Wireshark - Conversations - e6b235d9df6130e6f24fe730ce70cf31trace1.pcapng									
Conversation Settings		Ethernet - 4	IPv4 - 4	IPv6	TCP - 9	UDP - 3			
Name resolution	Absolute start time	Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A
		192.168.202.1	192.168.202.178	1,031	171 kB	497	34 kB	534	138 kB
		192.168.202.178	95.216.195.133	11	934 bytes	6	456 bytes	5	478 bytes
		192.168.202.178	192.168.202.2	8	856 bytes	4	312 bytes	4	544 bytes
		192.168.202.178	192.168.202.254	2	660 bytes	1	318 bytes	1	342 bytes

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.

Wireshark - Conversations - e6b235d9df6130e6f24fe730ce70cf31trace1.pcapng									
Conversation Settings		Ethernet - 4	IPv4 - 4	IPv6	TCP - 9	UDP - 3	Apply as Filter	Selected	A ↔ B
Name resolution	Absolute start time	Address A	Address B	Packets	Bytes		Prepare as Filter	Not Selected	A → B
		192.168.202.1	192.168.202.178	1,031	171 kB		Find	...and Selected	B → A
		192.168.202.178	95.216.195.133	11	934 bytes		Colorize	...or Selected	A ↔ Any
		192.168.202.178	192.168.202.2	8	856 bytes			...and not Selected	A → Any
		192.168.202.178	192.168.202.254	2	660 bytes				

Furthermore, below shows that 192.168.202.1 initiated the connection as it sent a packet with the SYN flag set and then 192.168.202.178 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.

e6b235d9df6130e6f24fe730ce70cf31trace1.pcapng									
No.	Time	Source	Destination	Protocol	Length	Info			
7	0.303921405	192.168.202.1	192.168.202.178	TCP	74	52206 → 4242 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_P			
8	0.394153386	192.168.202.178	192.168.202.1	TCP	74	4242 → 52206 [SYN, ACK] Seq=0 Ack=1 Win=65166 Len=0 MSS=1460 SACK_P			

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 I
- Frame	100.0	1056	100.0	173891	22 k	0
- Ethernet	100.0	1056	8.5	148444	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 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:

```
Wireshark - Follow TCP Stream (tcp.stream eq 0) - e6b235d9df6130e6f24fe730ce70cf31trace1.pcapng

[32mWelcome to HRC.[0m
[32mVersion 3.14.[0m
[31mLogin to continue.[0m

AUTH:USR:Crash-- Unknown User .
AUTH:USR:Crash OverrideAUTH:PASS:15b29ffdce06e10527a65bc6d71ad94dWelcome to the Server .[31mCrash Override.[0m
You have joined the .[32mGeneral.[0m Channel
[32m[General].[0m .[34mJoey.[0m: Sup Crash!
MSG:GENERAL:Lo Joey
[32m[General].[0m .[34mJoey.[0m: Seen the Phreak?
LST:USR
Current User
- Crash Override
MSG:GENERAL:Doesn't appear to be here yet
[32m[General].[0m .[34mPhantom.[0m: Has joined the channel
[32m[General].[0m .[34mJoey.[0m: Phreak-Phreak-Phreak-Phreak! Dude-dude-dude-dude-dude!
[32m[General].[0m .[34mPhantom.[0m: Joey, Joey! One more "dude" outta you, and I'mma slap the life outta you, ok?! Now I keep trying to save you from yourself, but you gotta stop letting your mama dress you, man! You're hopeless!
MSG:GENERAL:I got some more info on The Gibson!
[32m[General].[0m .[34mJoey.[0m: The Gibson, Ellingsons new toy?
[32m[General].[0m .[34mPhantom.[0m: Not in a Public Channel Dude!
MSG:GENERAL:Give me 2
LST:CHAN
Current Channels
- GENERAL (Public)
- REVERSING (Public)
```

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:

```
MSG:GENERAL:I got some more info on The Gibson!
[32m[General].[0m .[34mJoey.[0m: The Gibson, Ellingsons new toy?
[32m[General].[0m .[34mPhantom.[0m: Not in a Public Channel Dude!
MSG:GENERAL:Give me 2
LST:CHAN
Current Channels
- GENERAL (Public)
- REVERSING (Public)
CHAN:NEW:PRIVATE:gibson
(private) Channel .[31m[Gibson].[0m Created
CHAN:ADD:gibson:Joey
Add .[34mb'Joey'.[0m to .[31m[Gibson].[0m OK
CHAN:ADD:gibson:Phantom
Add .[34mb'Phantom'.[0m to .[31m[Gibson].[0m OK
MSG:GIBSON:Private Chat, Better?
[31m[Gibson].[0m .[34mPhantom.[0m: A Little, still don't trust the security here
[31m[Gibson].[0m .[34mJoey.[0m: What can we do about that Phreak?
[31m[Gibson].[0m .[34mPhantom.[0m: Encryption! But we will need keys
[31m[Gibson].[0m .[34mPhantom.[0m: And I don't trust that Diffe-Hellman Stuff Either
[31m[Gibson].[0m .[34mPhantom.[0m: Fortunately, I rolled my own key generator, you can find it on the FTP
MSG:GIBSON:Usual Place, and Creds?
[31m[Gibson].[0m .[34mPhantom.[0m: Same Bat Time and same Bat Channel
```

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:

e6b235d9df6130e6f24fe730ce70cf31trace1.pcapng							
No.	Time	Source	Destination	Protocol	Length	Info	
525	28.665246584	192.168.202.178	192.168.202.1	FTP	148	Response: 220 Welcome	
527	28.665446031	192.168.202.1	192.168.202.178	FTP	80	Request: USER babbage	
529	28.666891833	192.168.202.178	192.168.202.1	FTP	100	Response: 331 Please s	
530	28.667060895	192.168.202.1	192.168.202.178	FTP	88	Request: PASS HackTheP	
535	28.711085055	192.168.202.178	192.168.202.1	FTP	89	Response: 230 Login su	
536	28.711153304	192.168.202.1	192.168.202.178	FTP	74	Request: TYPE A	
538	28.711478951	192.168.202.178	192.168.202.1	FTP	96	Response: 200 Switchin	
539	28.711560427	192.168.202.1	192.168.202.178	FTP	72	Request: PASV	
540	28.712066157	192.168.202.178	192.168.202.1	FTP	113	Response: 227 Entering	
544	28.712625688	192.168.202.1	192.168.202.178	FTP	72	Request: LIST	
545	28.713190933	192.168.202.178	192.168.202.1	FTP	108	Response: 150 Here com	
551	28.713812160	192.168.202.178	192.168.202.1	FTP	90	Response: 226 Director	
553	28.713960561	192.168.202.1	192.168.202.178	FTP	74	Request: TYPE I	
554	28.714185888	192.168.202.178	192.168.202.1	FTP	97	Response: 200 Switchin	
555	28.714279230	192.168.202.1	192.168.202.178	FTP	72	Request: PASV	

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:

```
Wireshark - Follow TCP Stream (tcp.stream eq 4) - e6b235d9df6130e6f24fe730ce70cf31trace1.pcapng

220 Welcome Alpine ftp server https://hub.docker.com/r/delfer/alpine-ftp-server/
USER babbage
331 Please specify the password.
PASS HackThePlanet
230 Login successful.
TYPE A
200 Switching to ASCII mode.
PASV
227 Entering Passive Mode (172,42,0,2,82,10).
LIST
150 Here comes the directory listing.
226 Directory send OK.
TYPE I
200 Switching to Binary mode.
PASV
227 Entering Passive Mode (172,42,0,2,82,13).
RETR keygen
150 Opening BINARY mode data connection for keygen (21632 bytes).
226 Transfer complete.
QUIT
221 Goodbye.
```

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:

Wireshark - Packet 525 · e6b235d9df6130e6f24fe730ce70cf31trace1.pcapng															
Internet Protocol Version 4, Src: 192.168.202.178, Dst: 192.168.202.1															
Transmission Control Protocol, Src Port: 21, Dst Port: 55376, Seq: 1, Ack: 1, Len: 82															
Source Port: 21 Destination Port: 55376 [Stream index: 4] [Conversation completeness: Complete, WITH_DATA (31)] [TCP Segment Len: 82] Sequence Number: 1 (relative sequence number) Sequence Number (raw): 1252301658 [Next Sequence Number: 83 (relative sequence number)] Acknowledgment Number: 1 (relative ack number) Acknowledgment number (raw): 3300811685 1000 .... = Header Length: 32 bytes (8)															
0000 00 50 56 c0 00 08 00 0c 29 75 d0 59 08 00 45 00 .PV.....)u.Y..E. 0010 00 86 06 1c 40 00 3f 06 1f 51 c0 a8 ca b2 c0 a8 ..@?..Q... 0020 ca 01 00 15 d8 50 4a a4 9b 5a c4 be 63 a5 80 18 .....PJ..Z..c... 0030 01 fe 0b 29 00 00 01 01 08 0a 97 fc 5f 7e ad 17 .....).~...~... 0040 8d 37 32 32 30 20 57 65 6c 63 f6 6d 65 20 41 6c 7220 We lcome Al 0050 70 69 6e 65 20 66 74 70 20 73 65 72 76 65 72 20 pine ftp server 0060 68 74 74 70 73 3a 2f 2f 68 75 62 2e 64 6f 63 6b https:// hub.dock 0070 65 72 2e 63 6f 6d 2f 72 2f 64 65 6c 66 65 72 2f er.com/r /delfer/ 0080 61 6c 70 69 6e 65 2d 66 74 70 2d 73 65 72 76 65 alpine-f tp-serv 0090 72 2f 0d 0a r/..															

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

No.	Time	Source	Destination	Protocol	Length	Info
539	28.71156027	192.168.202.1	192.168.202.178	FTP	72	Request: PASV
540	28.71260817	192.168.202.178	192.168.202.1	FTP	113	Response: 227 Entering Passive Mode (172,42,0,2,82,10)
541	28.712559538	192.168.202.1	192.168.202.178	TCP	74	47780 → 21002 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACI
542	28.712559995	192.168.202.1	192.168.202.178	TCP	74	21002 → 47780 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 M
543	28.712574442	192.168.202.1	192.168.202.178	TCP	66	47780 → 21002 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=65280 Lseq=65280
544	28.712625680	192.168.202.1	192.168.202.178	FTP	72	Request: LIST
545	28.713190933	192.168.202.178	192.168.202.1	FTP	105	Response: 150 Here comes the directory listing.
546	28.713335881	192.168.202.178	192.168.202.1	TCP	130	21002 → 47780 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=64
547	28.713344175	192.168.202.1	192.168.202.178	TCP	66	47780 → 21002 [ACK] Seq=1 Ack=65 Win=64256 Len=0 TSval=65280 Lseq=65280
548	28.713436018	192.168.202.178	192.168.202.1	TCP	66	21002 → 47780 [FIN, ACK] Seq=65 Ack=1 Win=65280 Len=0

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

No.	Time	Source	Destination	Protocol	Length	Info
541	28.712606157	192.168.202.178	192.168.202.1	FTP	113	Response: 227 Entering Passive Mode (172,42,0,2,82,10)
542	28.712559538	192.168.202.1	192.168.202.178	TCP	74	47780 → 21002 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACI
543	28.712574442	192.168.202.1	192.168.202.178	TCP	66	47780 → 21002 [ACK] Seq=1 Ack=1 Win=65160 Len=0 M
544	28.712625680	192.168.202.1	192.168.202.178	FTP	72	Request: LIST
545	28.713190933	192.168.202.178	192.168.202.1	FTP	105	Response: 150 Here comes the directory listing.
546	28.713345881	192.168.202.178	192.168.202.1	TCP	130	21002 → 47780 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=64
547	28.713344175	192.168.202.1	192.168.202.178	TCP	66	47780 → 21002 [ACK] Seq=65 Ack=65 Win=64256 Len=0 TSval=65280 Lseq=65280
548	28.713366928	192.168.202.1	192.168.202.178	TCP	66	47780 → 21002 [FIN, ACK] Seq=66 Ack=66 Win=64256 Len=0
550	28.713758558	192.168.202.178	192.168.202.1	TCP	66	21002 → 47780 [ACK] Seq=66 Ack=2 Win=65280 Len=0 TSval=65280 Lseq=65280
551	28.713812100	192.168.202.178	192.168.202.1	FTP	90	Response: 226 Directory send OK.

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.

Wireshark - Follow TCP Stream (tcp.stream eq 5) - e6b235d9df6130e6f24fe730ce70cf31trace1.pcapng						
-rwxr-Xr-X	1	1000	1000	21632	Nov 06 23:18	keygen

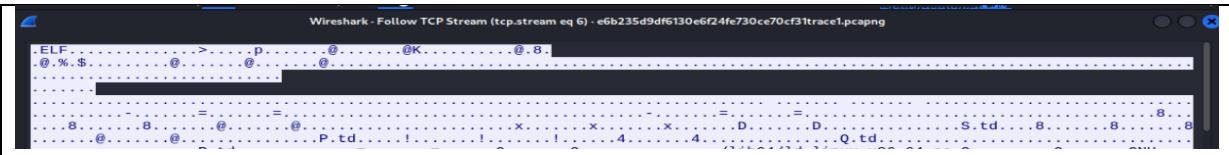
By following the stream of those 5 TCP packets, above shows the actual keygen file and its information.

No.	Time	Source	Destination	Protocol	Length	Info
540	28.71000707	192.168.202.1	192.168.202.178	FTP	66	21005 → 21005 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=65280 Lseq=65280
561	28.71548220	192.168.202.178	192.168.202.1	FTP	133	Response: 150 Opening BINARY mode data connection for keygen
562	28.715815087	192.168.202.178	192.168.202.1	TCP	1514	21005 → 37420 [ACK] Seq=1 Ack=1 Win=65280 Len=1448 TSval=65280 Lseq=65280
563	28.715824995	192.168.202.1	192.168.202.178	TCP	66	37420 → 21005 [ACK] Seq=1 Ack=1449 Win=64128 Len=0 TSval=64128 Lseq=64128
564	28.715825058	192.168.202.178	192.168.202.1	TCP	1514	21005 → 37420 [ACK] Seq=1449 Ack=1 Win=65280 Len=1448 TSval=65280 Lseq=65280
565	28.715834408	192.168.202.1	192.168.202.178	TCP	66	37420 → 21005 [ACK] Seq=2897 Ack=1 Win=63360 Len=0 TSval=63360 Lseq=63360
566	28.715834887	192.168.202.178	192.168.202.1	TCP	1514	21005 → 37420 [ACK] Seq=2897 Ack=1 Win=62336 Len=0 TSval=62336 Lseq=62336
568	28.715844920	192.168.202.178	192.168.202.1	TCP	1514	21005 → 37420 [ACK] Seq=445 Ack=1 Win=65280 Len=1448 TSval=65280 Lseq=65280
569	28.715847839	192.168.202.1	192.168.202.178	TCP	66	37420 → 21005 [ACK] Seq=1 Ack=5793 Win=61184 Len=0 TSval=61184 Lseq=61184
570	28.715851526	192.168.202.178	192.168.202.1	TCP	1514	21005 → 37420 [PSH, ACK] Seq=5793 Ack=1 Win=65280 Len=0 TSval=65280 Lseq=65280
571	28.715851437	192.168.202.1	192.168.202.178	TCP	66	37420 → 21005 [ACK] Seq=1 Ack=7241 Win=60160 Len=0 TSval=60160 Lseq=60160
572	28.715966700	192.168.202.178	192.168.202.1	TCP	1514	21005 → 37420 [ACK] Seq=7241 Ack=1 Win=65280 Len=1448 TSval=65280 Lseq=65280
573	28.715968224	192.168.202.1	192.168.202.178	TCP	66	37420 → 21005 [ACK] Seq=1 Ack=8689 Win=64128 Len=0 TSval=64128 Lseq=64128

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.

No.	Time	Source	Destination	Protocol	Length	Info
583	28.716145081	192.168.202.1	192.168.202.178	TCP	66	37420 → 21005 [ACK] Seq=1 Ack=15929 Win=64128 Len=0 TSval=64128 Lseq=64128
584	28.716150509	192.168.202.178	192.168.202.1	TCP	1514	21005 → 37420 [ACK] Seq=15929 Ack=1 Win=65280 Len=1448 TSval=65280 Lseq=65280
585	28.716153455	192.168.202.1	192.168.202.178	TCP	66	37420 → 21005 [ACK] Seq=1 Ack=17377 Win=63360 Len=0 TSval=63360 Lseq=63360
586	28.716159419	192.168.202.178	192.168.202.1	TCP	1514	21005 → 37420 [ACK] Seq=17377 Ack=1 Win=65280 Len=1448 TSval=65280 Lseq=65280
587	28.716162420	192.168.202.1	192.168.202.178	TCP	66	37420 → 21005 [ACK] Seq=1 Ack=18825 Win=62336 Len=0 TSval=62336 Lseq=62336
588	28.716166139	192.168.202.178	192.168.202.1	TCP	1514	21005 → 37420 [PSH, ACK] Seq=18825 Ack=1 Win=65280 Len=0 TSval=65280 Lseq=65280
589	28.716169170	192.168.202.1	192.168.202.178	TCP	66	37420 → 21005 [ACK] Seq=1 Ack=20273 Win=61184 Len=0 TSval=61184 Lseq=61184
590	28.716170007	192.168.202.178	192.168.202.1	TCP	1420	21005 → 37420 [ACK] Seq=20273 Ack=1 Win=65280 Len=0 TSval=65280 Lseq=65280
591	28.716174060	192.168.202.1	192.168.202.178	TCP	66	37420 → 21005 [FIN, ACK] Seq=21634 Ack=2 Win=64128 Len=0 TSval=64128 Lseq=64128
592	28.716388881	192.168.202.178	192.168.202.1	TCP	66	21005 → 37420 [ACK] Seq=21634 Ack=2 Win=65280 Len=0 TSval=65280 Lseq=65280
593	28.716514060	192.168.202.178	192.168.202.1	FTP	90	Response: 226 Transfer complete.
594	28.716567660	192.168.202.1	192.168.202.178	TCP	66	37420 → 21005 [ACK] Seq=21634 Ack=2 Win=65280 Len=0 TSval=65280 Lseq=65280

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:

MSG:GIBSON:Got It  
MSG:GIBSON:Instructions?  
. [31m[Gibson]. [0m . [34mPhantom.[0m: Let me know when the tool is running, I will send the go-code to the generator

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

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.

Cutter – /home/kali/keyGenerator/keygen.elf

File Edit View Windows Debug Tools Help

Type flag name or address here

Functions Disassembly

Name	Address	Disassembly
dbg_start	0x00001420	mov rax, qword fs:[0x28]
entry.init0	0x00001421	mov qword [canary], rax
main	0x00001425	xor eax, eax
sym._do_global_dtors_aux	0x00001427	lea rax, [str.Enter_Password] ; 0x203b
sym._fini	0x0000142e	mov rdi, rax ; const char *format
sym._init	0x00001431	mov eax, 0
sym.checkPassword	0x00001436	call printf ; sym.imp.printf : int printf(const char *
sym.deregister_tm_clones	0x0000143b	rdx, qword [stdin] ; obj.__TMC_END
sym.getRemoteCode	0x00001442	; 0x40b0 ; FILE *stream
sym.imp._stack_chk_fail	0x00001446	lea rax, [s1]
sym.imp.accept	0x0000144b	mov esi, 0x20 ; int size
sym.imp.atoi	0x0000144e	mov rdi, rax ; char *
sym.imp.bind	0x00001453	call fgets ; sym.imp.fgets : char *fgets(char *,
sym.imp.close	0x00001457	lea rax, [s1]
sym.imp.exit	0x00001459	rdx, [data.0000204c] ; 0x204c
sym.imp.fgets	0x00001461	mov rsi, rdx ; const char *s2
sym.imp.fread	0x00001464	mov rdi, rax ; const char *s1
sym.imp.fwrite	0x00001469	call strcspn ; sym.imp.strcspn : size_t strcspn(const
sym.imp.getenv	0x00001472	byte [rbp + rax - 0x30], 0
sym.imp.getenv	0x00001479	lea rax, [s1]
sym.imp.getenv	0x00001479	rdx, [str.U3dvcRGMXNoIQ] ; 0x204f
sym.imp.getenv	0x00001479	mov rsi, rdx ; const char *s2

```
;-- str.U3dvcRGMXNoIQ:
0x0000204f .string "U3dvcRGMXNoIQ==" ; len=17
```

Furthermore, the password was ‘U3dvcRGMXNoIQ==’ and this was inputted:

```
(kali㉿kali)-[~/keyGenerator]
$ ./keygen.elf
Keygen Version 1.0
By the Phantom Phreak
Password Protected for your safely
Enter Password> U3dvcRGMXNoIQ=
Awaiting Go Code
```

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

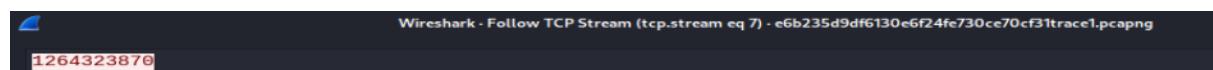
Going back to the conversation, Phantom stated they would send the key to Crash so they can input it into the generator:

```
MSG:GIBSON:Running
.[31m[Gibson].[0m .[34mPhantom.[0m: Init Packet Sent. That should spit out a key
```

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.

No.	Time	Source	Destination	Protocol	Length	Info
664	33.733589477	192.168.202.178	192.168.202.1	TCP	142	4222 → 33096 [PSH, ACK] Seq=9961 Ack=37 Win=501 Len=76
666	33.733957528	192.168.202.178	192.168.202.1	TCP	66	48886 → 1337 [ACK] Seq=1 Ack=1 Win=64256 Len=0 Tsvl=2
667	33.734200788	192.168.202.178	192.168.202.1	TCP	76	48886 → 1337 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=10 T
670	33.734650953	192.168.202.178	192.168.202.1	TCP	66	48886 → 1337 [FIN, ACK] Seq=11 Ack=2 Win=64256 Len=0 T

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
TCP      66 48886 → 1337    4888
Numerial Go Code is 1264323870
27 : 82
53 : 114
8 : 56
61 : 122
13 : 68
25 : 80
19 : 74
41 : 102
12 : 67    00 0c 29 75 d0 59 01
52 : 113    00 34 20 e5 40 00 41
51 : 112    ca b2 05 39 be f6 31
7 : 55    01 fd a4 ca 00 00 01
40 : 101    67 31
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+ICzHcs9Bugg3FePPzUjF29YDD610M]
MSG:GIBSON:g7V+jqhvbaY06Y9Y964X8Y7QW0jwxuxvb/hFutJqXCo=
. [31m[Gibson]: [0m . [34mPhantom.[0m: rezqlf8RCF2jGbLKogsXPBeIY4No2ak21h+u3pwpm4=
MSG:GIBSON:XB/GZcmWjTHHvltD9QxYTA==_
. [31m[Gibson]: [0m . [34mPhantom.[0m: V0hgwmHB71ko0V896s6qkFOn1kGBfiHQc8k/tw/hGYQ=
QUIT
Input of 'b'QUIT' not understood
```

## Encryption Section

The messages that were encrypted are as follows:

1.	Phantom	R8Ucw6AfL1/voZJEIAd0D7tWfzBffs8J4+ICzHcS9Bugg3FePPzUjF29YDD61OMj
2.	Crash	g7V+JqhbAy06Y9Y964X8Y7QW0jwxuvb/hFutJqXCo=
3.	Phantom	rezQlf8RCF2jGbLKogsXPBeIY4No2aK21h+u3pWpnM4=
4.	Crash	XB/GZCmWjTHHvltD9QxYTA==
5.	Phantom	V0hgWmH871koOV896s6qkFOn1kGBfiHQc8k/tW/hGYQ=
<ul style="list-style-type: none"> <li>- When reading Phantom's messages, 'CuriosityIsCrime' should be used for the IV</li> <li>- When reading Crash's messages, 'Hack_The_Planet!' Should be used for the IV</li> <li>- When decrypting, 'Rr8zDPJfCqp7eFYX' is the key</li> </ul>		

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:

The screenshot shows the CyberChef interface with the following configuration:

- Recipe:** From Base64
- Input:** R8Ucw6AfL1/voZJEIAd0D7tWfzBffs8J4+ICzHcS9Bugg3FePPzUjF29YDD61OMj
- AES Decrypt Parameters:**
  - Key: Rr8zDPJfCqp7eFYX
  - IV: CuriosityIsCrime
  - Mode: CBC
  - Input: Raw
  - Output: Raw
- Output:** Invalid key length: 0 bytes
- Note:** The following algorithms will be used based on the size of the key:  
16 bytes = AES-128  
24 bytes = AES-192  
32 bytes = AES-256

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

1. Phantom to Crash	<p>The screenshot shows the CyberChef interface with the following configuration:</p> <ul style="list-style-type: none"> <li><b>Recipe:</b> From Base64</li> <li><b>Input:</b> R8Ucw6AfL1/voZJEIAd0D7tWfzBffs8J4+ICzHcS9Bugg3FePPzUjF29YDD61OMj</li> <li><b>AES Decrypt Parameters:</b> <ul style="list-style-type: none"> <li>Key: Rr8zDPJfCqp7eFYX</li> <li>IV: CuriosityIsCrime</li> <li>Mode: CBC</li> <li>Input: Raw</li> <li>Output: Raw</li> </ul> </li> <li><b>Output:</b> Testing Message, Can you hear Me?</li> </ul>	"Testing Message, Can you hear Me?"
---------------------------	---	-------------------------------------

2. Crash to Phantom	<p>Recipe</p> <p>From Base64</p> <p>Alphabet: A-Za-z0-9+=</p> <p><input checked="" type="checkbox"/> Remove non-alphabet chars <input type="checkbox"/> Strict mode</p> <p>AES Decrypt</p> <p>Key: Rr8zDPJfcqp... IV: Hack_The_P1... UTF8</p> <p>Mode: CBC Input: Raw Output: Raw</p> <p>Input: #g7V+JqhvbAy06Y9Y964X8Y7Q\0jwxuvb/hFutJqXCo=</p> <p>Output: Receiving Loud and Clear.</p>	<p>“Receiving Loud and Clear.”</p>
3. Phantom to Crash	<p>Recipe</p> <p>From Base64</p> <p>Alphabet: A-Za-z0-9+=</p> <p><input checked="" type="checkbox"/> Remove non-alphabet chars <input type="checkbox"/> Strict mode</p> <p>AES Decrypt</p> <p>Key: Rr8zDPJfcqp... IV: CuriosityIs... UTF8</p> <p>Mode: CBC Input: Raw Output: Raw</p> <p>Input: RBUU6aFL1/v0ZETIA0007tWfzbffsB34+ICzilcS9Bugg3FePPzUjF29Y0D610Hjrezq1f8RCF2jgbLkogsXpBeIY4m02aC2lru3phpmM4-V0hgdm871ko0v99566qkr0nk@f1hqcik/7u/HgQk</p> <p>Output: Testing Message, Can you hear Me? Do You want a flag?</p>	<p>“Do You want a flag?”</p>
4. Crash to Phantom	<p>Recipe</p> <p>From Base64</p> <p>Alphabet: A-Za-z0-9+=</p> <p><input checked="" type="checkbox"/> Remove non-alphabet chars <input type="checkbox"/> Strict mode</p> <p>AES Decrypt</p> <p>Key: Rr8zDPJfcqp... IV: Hack_The_P1... UTF8</p> <p>Mode: CBC Input: Raw Output: Raw</p> <p>Input: #g7V+JqhvbAy06Y9Y964X8Y7Q\0jwxuvb/hFutJqXCo=XB/GZCmWjTHh1tD9QxyTA==</p> <p>Output: Receiving Loud and Clear. REBELBELBELBELBELBEL Yes.</p>	<p>“Yes.”</p>
5. Phantom to Crash	<p>Recipe</p> <p>From Base64</p> <p>Alphabet: A-Za-z0-9+=</p> <p><input checked="" type="checkbox"/> Remove non-alphabet chars <input type="checkbox"/> Strict mode</p> <p>AES Decrypt</p> <p>Key: Rr8zDPJfcqp... IV: CuriosityIs... UTF8</p> <p>Mode: CBC Input: Raw Output: Raw</p> <p>Input: RBUU6aFL1/v0ZETIA0007tWfzbffsB34+ICzilcS9Bugg3FePPzUjF29Y0D610Hjrezq1f8RCF2jgbLkogsXpBeIY4m02aC2lru3phpmM4-V0hgdm871ko0v99566qkr0nk@f1hqcik/7u/HgQk</p> <p>Output: Testing Message, Can you hear Me? Do You want a flag? 6052{D3C0d3_N3tW0rk}</p>	<p>“6052{D3C0d3_N3tW0rk}”</p>

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:

No.	Time	Source	Destination	Protocol	Length	Info
130	13.458410018	192.168.202.178	192.168.202.1	TCP	142	4222 → 51676 [PSH, ACK] Seq=1
131	13.458466316	192.168.202.1	192.168.202.178	TCP	66	51676 → 4222 [ACK] Seq=1 A
132	13.459512555	192.168.202.178	192.168.202.1	RSL	134	CHANnel ACTIVation
133	13.459713122	192.168.202.1	192.168.202.178	TCP	66	51676 → 4222 [ACK] Seq=1 A

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:

A Wireshark capture window titled "e6b235d9df6130e6f24fe730ce70cf31trace1.pcapng". The packet list shows several TCP connections between 192.168.202.1 and 192.168.202.178. A specific RST packet (packet 356) is highlighted, labeled as "166 TFO Modification REQuest".

No.	Time	Source	Destination	Protocol	Length	Info
353	21.589573783	192.168.202.178	192.168.202.1	TCP	114	4242 → 52206 [PSH, ACK] Seq=1501
354	21.590097836	192.168.202.178	192.168.202.1	TCP	550	4222 → 51676 [PSH, ACK] Seq=1840
355	21.590131052	192.168.202.1	192.168.202.178	TCP	66	51676 → 4222 [ACK] Seq=1 Ack=188
356	21.590447014	192.168.202.178	192.168.202.1	RST	166	TFO Modification REQuest
357	21.590470307	192.168.202.1	192.168.202.178	TCP	66	51676 → 4222 [ACK] Seq=1 Ack=189

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:

A Wireshark capture window showing an HTTP GET request from 192.168.202.1 to 95.216.195.133. The request is labeled "142 GET /nm-check.txt HTTP/1.1". The response is labeled "142 258 HTTP/1.1 200 OK (text/plain)".

No.	Time	Source	Destination	Protocol	Length	Info
595	28.716644318	192.168.202.1	192.168.202.178	FTP	72	Request: QUIT
597	28.717190023	192.168.202.178	192.168.202.1	FTP	80	Response: 221 Goodbye.
839	38.012688137	192.168.202.178	95.216.195.133	HTTP	142	GET /nm-check.txt HTTP/1.1
841	38.058701916	95.216.195.133	192.168.202.178	HTTP	258	HTTP/1.1 200 OK (text/plain)
134	13.466608981	192.168.202.178	192.168.202.1	TPA	158	TPA unknown 0xf1

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

A Wireshark follow-up window titled "Wireshark - Follow HTTP Stream (tcp.stream eq 8) · e6b235d9df6130e6f24fe730ce70cf31trace1.pcapng". It displays the raw HTTP response body, which includes the status line "HTTP/1.1 200 OK", headers (Server: nginx, Date: Sun, 06 Nov 2022 23:32:21 GMT, Content-Type: text/plain, Content-Length: 25, Connection: close, Cache-Control: max-age=0, must-revalidate), and the message "NetworkManager is online".

```
GET /nm-check.txt HTTP/1.1
Host: ping.archlinux.org
Accept: */*
Connection: close

HTTP/1.1 200 OK
Server: nginx
Date: Sun, 06 Nov 2022 23:32:21 GMT
Content-Type: text/plain
Content-Length: 25
Connection: close
Cache-Control: max-age=0, must-revalidate

NetworkManager is online
```

## 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>