

DESIGN DOCUMENT

How to Run code:

1. Open two terminals in client and server folders.
2. Run python3 client.py and python3 server.py
3. Select encoding style from 0,1,2
4. Type commands in order
 - a. Cwd [gives current working dir of server]
 - b. Ls [list all content in server]
 - c. Cd sys [move in folder name sys]
 - d. Cd cont [move in folder name cont]
 - e. Upd client.txt [upload client.txt from client to server cont dir]
 - f. Dwd server.txt [download server.txt from server's cont dir to client]

For creating the file system I have used three modes of layering:

- 1. File Service:** This is the top most layer of the system. It is like an application layer. This file service performs five commands through which client and server share information with each other. These five commands perform tasks like fetching current working directory, changing directory, fetching list of content and doing upload and download operations. This layer relies on the layer below it

which is a crypto service for encryption and decryption of data which needs to be exchanged between client and server.

2. Crypto Service: The second layer provides the encryption and decryption of the data sent between client and server. Whenever a client uses any command or sends any file it will use this service to encrypt the data using one of the any three encryption formats.

- a. Plain Text: This is a very simple form of service in which there is no need for encryption. The file is sent as a string as if there is no change.
- b. Caesar cipher: Not exactly like a caesar cipher but every lower or upper case letter is replaced by a letter with an offset of 2. I am maintaining two lists of alphabets and whenever we encounter upper or lowercase letters we set that letter with a letter with index equal to index of letter +2. Also we put a %26 which ensures that y and z matches to a and b. There is a decryptor function which reverses these offset by actually giving an offset of 24 which will make sure that letters are back to their original form. Whenever we send data either from client to server or server to client we encode using caesar cipher and when reading anything on client or server we use the decrypt function to decrypt it.
- c. Transpose/Reverse: In this encoding style we split every word of the data and reverse it individually. The string “My name is Prakram” will be encoded as “yM eman si markarP”. When sending any data we call reverse_encoding function which reverses the string in a given format and while reading data we again call the same function because reverse or reverse will give the actual data.

3. Networking service: The bottom most layer is the networking layer. It is responsible for the protocols for file transmission. I am using the TCP protocol in this layer to set up a connection between client and server. I have designed the client and server system in such a way that if a server is started once then it does

not need to shutdown and we can again and again connect the client to it. Clients on the other hand can continuously give commands until it exits by giving a “q” command.

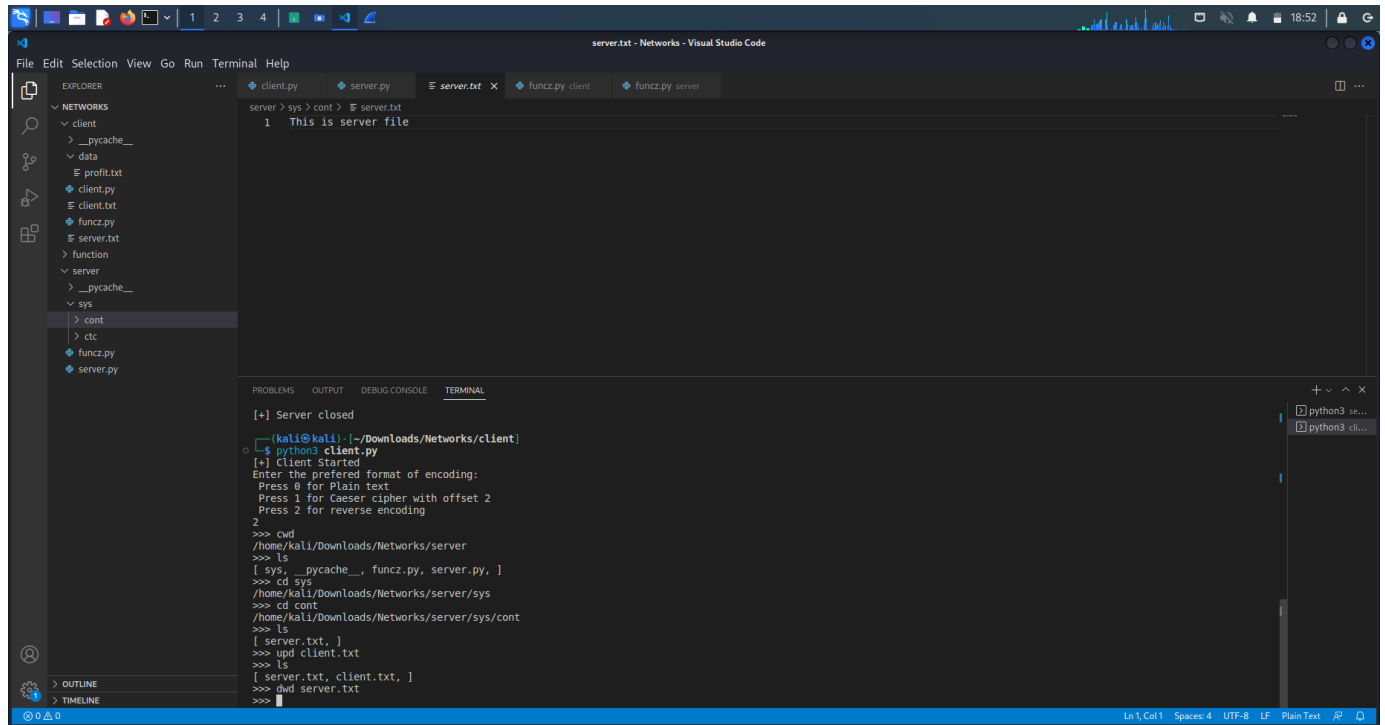
CHALLENGES FACED:

1. I tried implementing the file system using C language but was unable to do so because of the book which was not indexed properly. Compared to making the file system using python, C was a tougher choice as Python is more compatible and user friendly in doing such jobs.
2. Setting up the connection between client and server properly was a huge headache. When and how will the while loop end and how to make our server and client persistent so that the client does not close after giving one command. I tried making it look like an actual terminal.
3. To see the changes in file created while uploading and downloading commands, the connection between client and server needs to be closed in a proper fashion so that the changes in data of file created can be seen otherwise the data will not load into the file properly.
4. Error handling and debugging whenever there is an error was a difficult job to be done.
5. The encryption and decryption needed to be done properly. Writing code for it was a little bit difficult as every time we send or receive we need to take care of encryption and decryption and the encoding style that we are using in the starting of the system.
6. Downloading data from any directory of server and uploading data to any directory of the server was more challenging to achieve than simple downloading and uploading to server directory.

WIRESHARK SCREENSHOTS:

Encoding style: Caesar cipher

Note: Zoom in to see the commands clearly

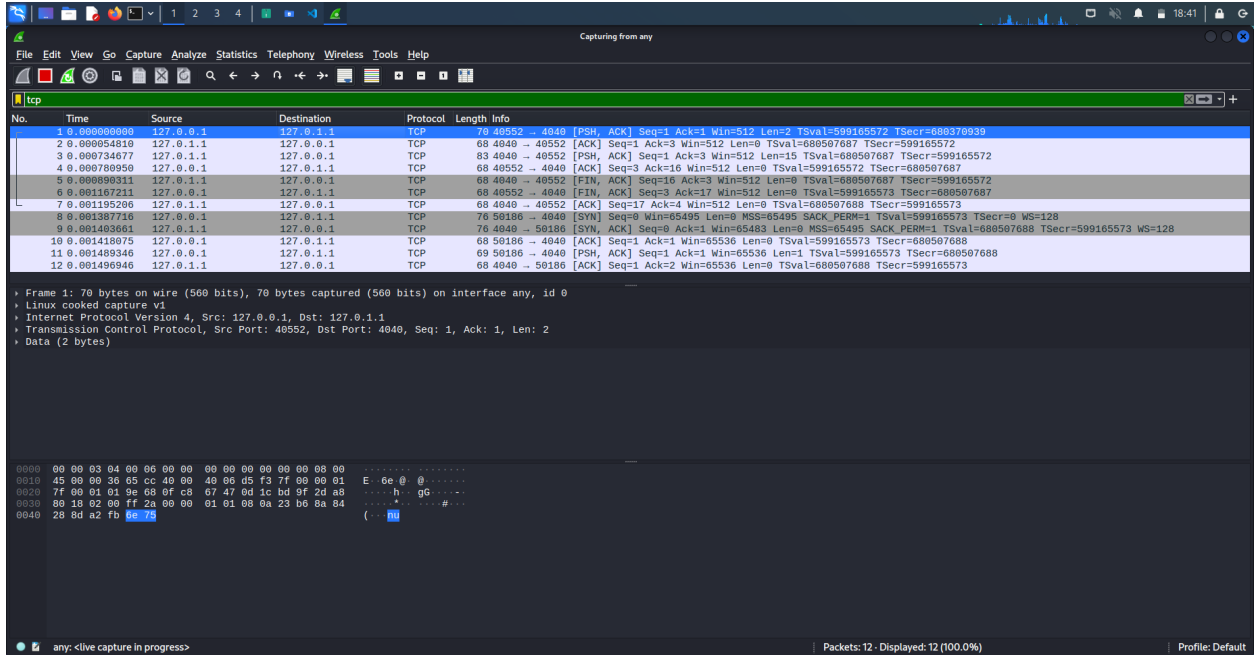


The screenshot shows a Visual Studio Code editor with a project named 'server.txt - Networks'. The Explorer sidebar on the left shows a file tree with folders 'client', 'data', 'function', 'server', and 'sys'. The 'server' folder is expanded, showing 'cont' and 'ctc'. The 'server.txt' file is open in the editor, containing the text '1 This is server file'. The Terminal at the bottom shows the following commands and output:

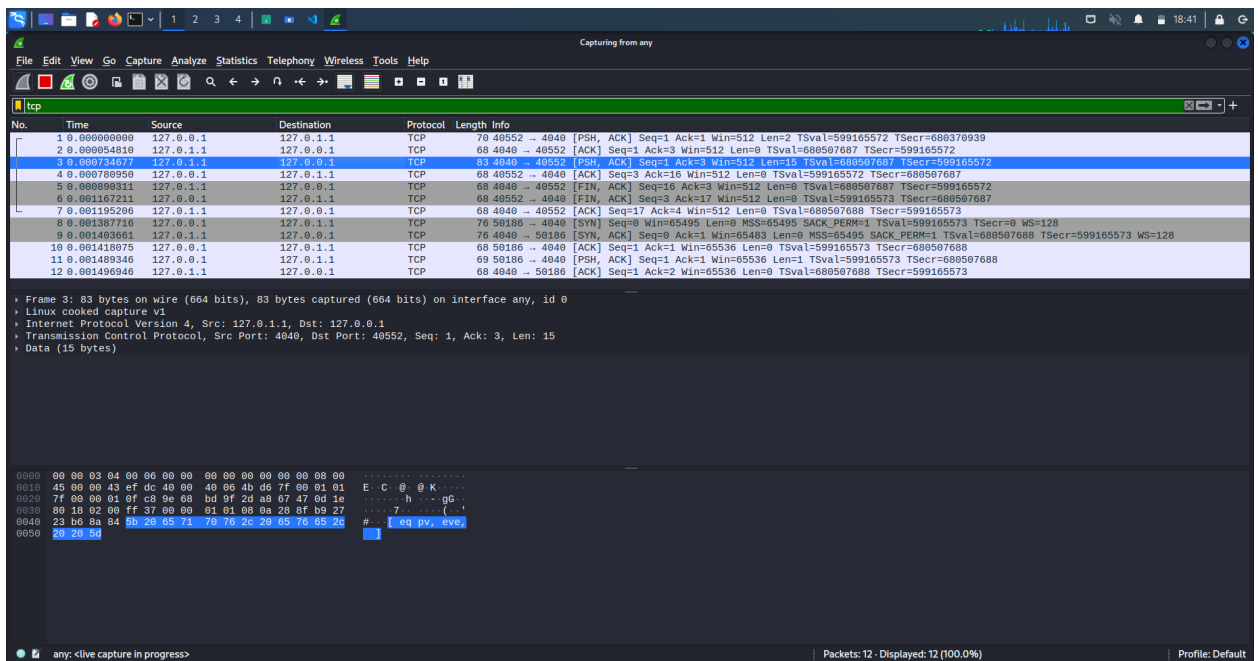
```
[+] Server closed
(kali@kali) ~/Downloads/Networks/client
$ python3 client.py
[+] Client Started
Enter the preferred format of encoding:
Press 0 for Plain text
Press 1 for Caesar cipher with offset 2
Press 2 for reverse encoding
2
>>> cwd
/home/kali/Downloads/Networks/server
>>> ls
[ sys, __pycache__, funcz.py, server.py, ]
>>> cd sys
/home/kali/Downloads/Networks/server/sys
>>> cd cont
/home/kali/Downloads/Networks/server/sys/cont
>>> ls
[ server.txt, ]
>>> upd client.txt
>>> ls
[ server.txt, client.txt, ]
>>> dwd server.txt
>>>
```

1. Cwd

cwd is encrypted as eyf [offset of 2]

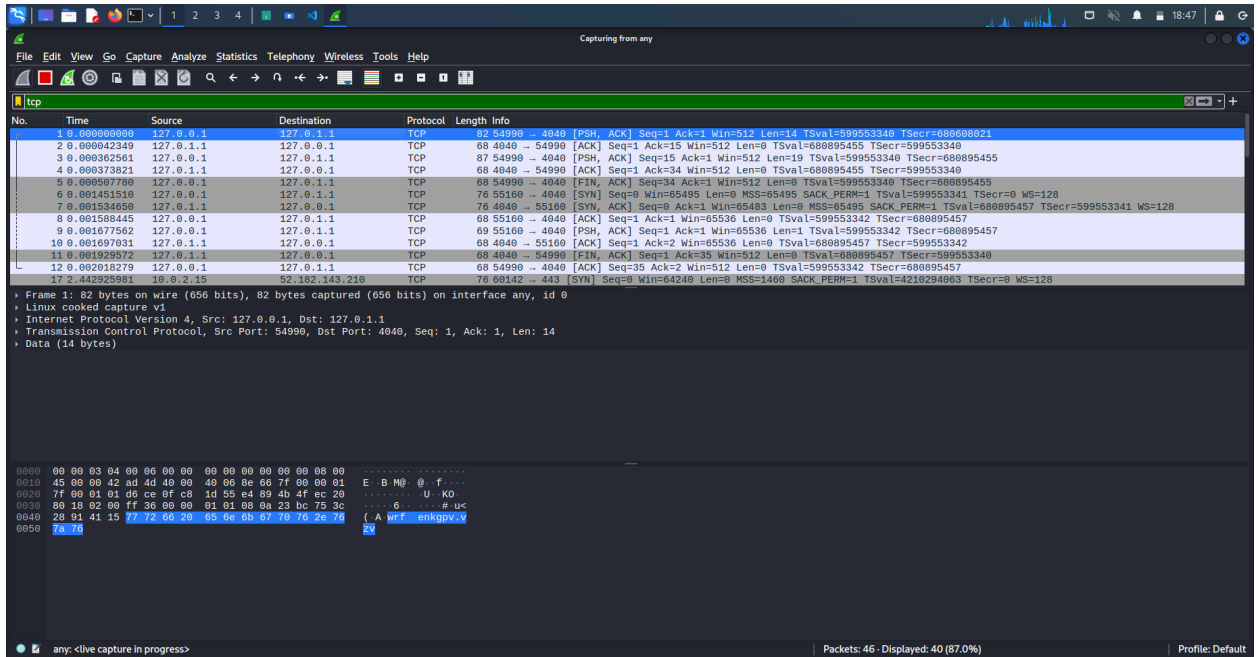


Getting the list of content with offset of 2

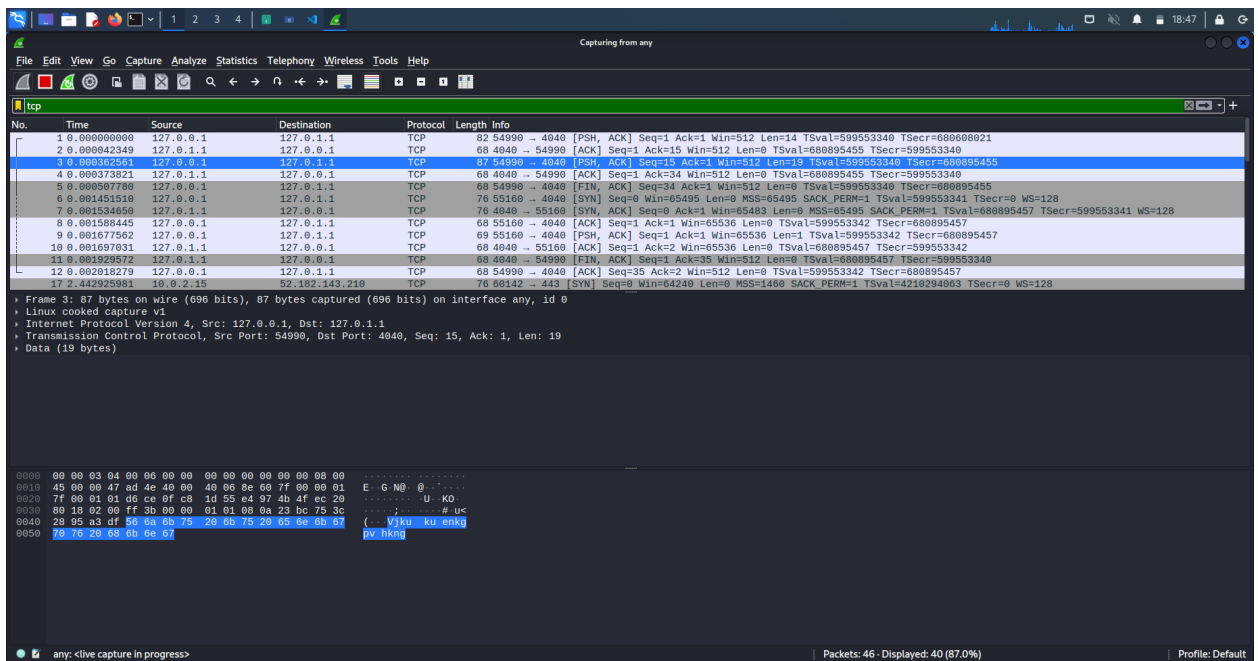


4. Upd

Upd client.txt is encrypted as wrf enkgpv.vzv



Passing the content of client.txt with an offset of 2



5. Dwd

Dwd server.py is encrypted with offset of 2

Wireshark interface showing a packet capture of a TCP connection. The packet list shows a sequence of packets from source 127.0.0.1 to destination 127.0.0.1. The packet details pane shows the selected packet (No. 1) with the following information:

- Frame 1: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface any, id 0
- Linux cooked capture v1
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- Transmission Control Protocol, Src Port: 60198, Dst Port: 4040, Seq: 1, Ack: 1, Len: 14
- Data (14 bytes)

The packet bytes pane shows the raw data in hexadecimal and ASCII:

```
0000 00 00 03 04 00 06 00 00 00 00 00 00 00 08 00  .....@FD...
0010 45 00 00 42 f5 6f 49 08 40 00 48 44 7f 00 00 01  E..@..@..X...
0020 7f 00 01 01 eb 76 0f c0 1f c5 58 00 87 eb 7c 02  ....&..X...|b
0030 00 18 02 00 ff 36 00 00 01 01 00 0a 23 be a8 02  ....6.....#...
0040 28 97 70 2e 86 79 66 20 75 67 74 78 67 74 26 78  (.p.fyfy..ugt>xt.v
0050 78 76 26 78 67 74 26 78 67 74 26 78 67 74 26 78  26 78 67 74 26 78 67 74 26 78 67 74 26 78 67 74 26 78
```

Content of server.txt with an offset of 2

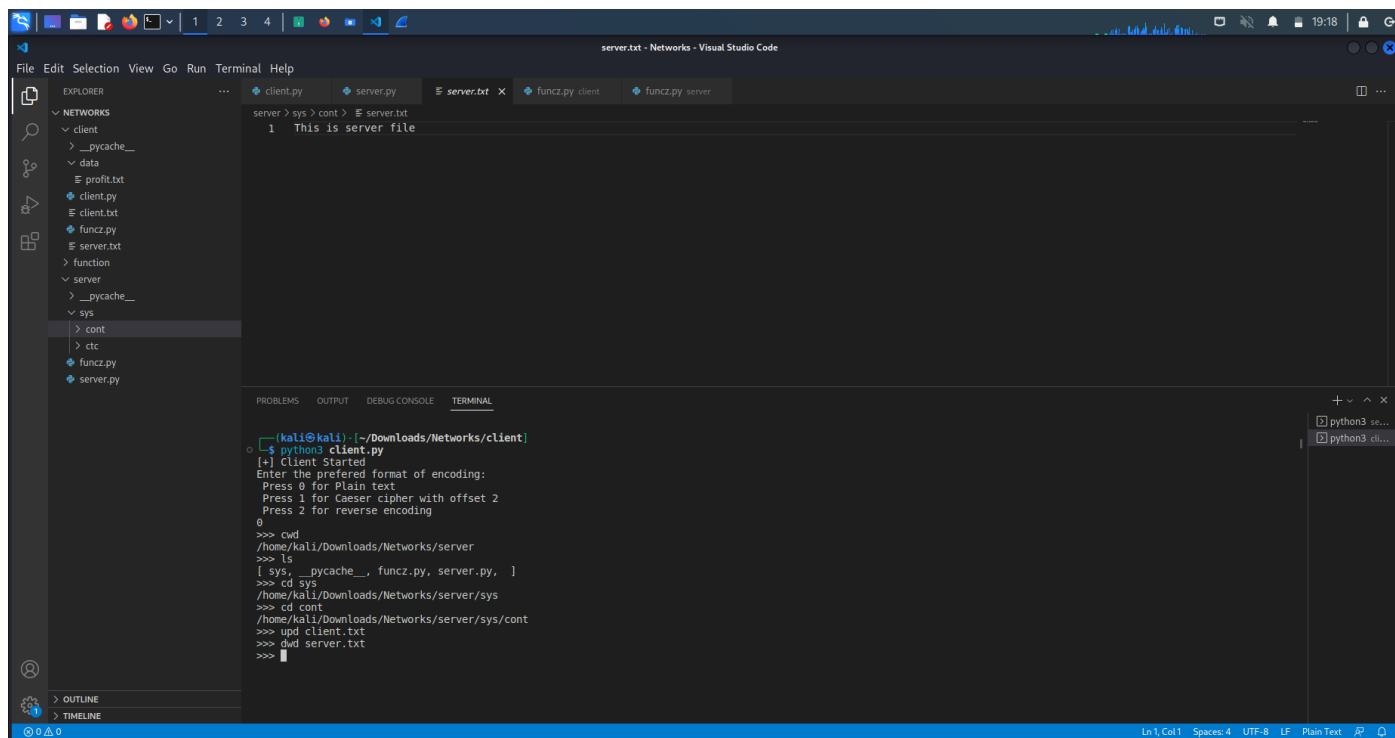
Wireshark interface showing a packet capture of a TCP connection. The packet list shows a sequence of packets from source 127.0.0.1 to destination 127.0.0.1. The packet details pane shows the selected packet (No. 3) with the following information:

- Frame 3: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface any, id 0
- Linux cooked capture v1
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- Transmission Control Protocol, Src Port: 4040, Dst Port: 60198, Seq: 1, Ack: 15, Len: 19
- Data (19 bytes)

The packet bytes pane shows the raw data in hexadecimal and ASCII:

```
0000 00 00 03 04 00 06 00 00 00 00 00 00 00 08 00  .....@FD...
0010 45 00 00 47 f5 6f 49 08 40 00 45 dd 7f 00 01 01  E..@..@..@E...
0020 7f 00 01 01 eb c8 eb 26 87 eb 7c 62 1f c5 58 e6  ....&..|b..X...
0030 00 18 02 00 ff 3b 00 00 01 01 00 0a 28 97 d6 a0  ....;.....(...
0040 23 be a8 02 86 8a 0b 75 20 6b 75 20 75 67 74 78  #..Vku..ku>gtx
0050 67 74 20 6b 0b 0b 67 74 20 6b 0b 0b 67 74 20 6b  67 74 20 6b 0b 0b 67 74 20 6b 0b 0b 67 74 20 6b
```

Encoding style: Plain text

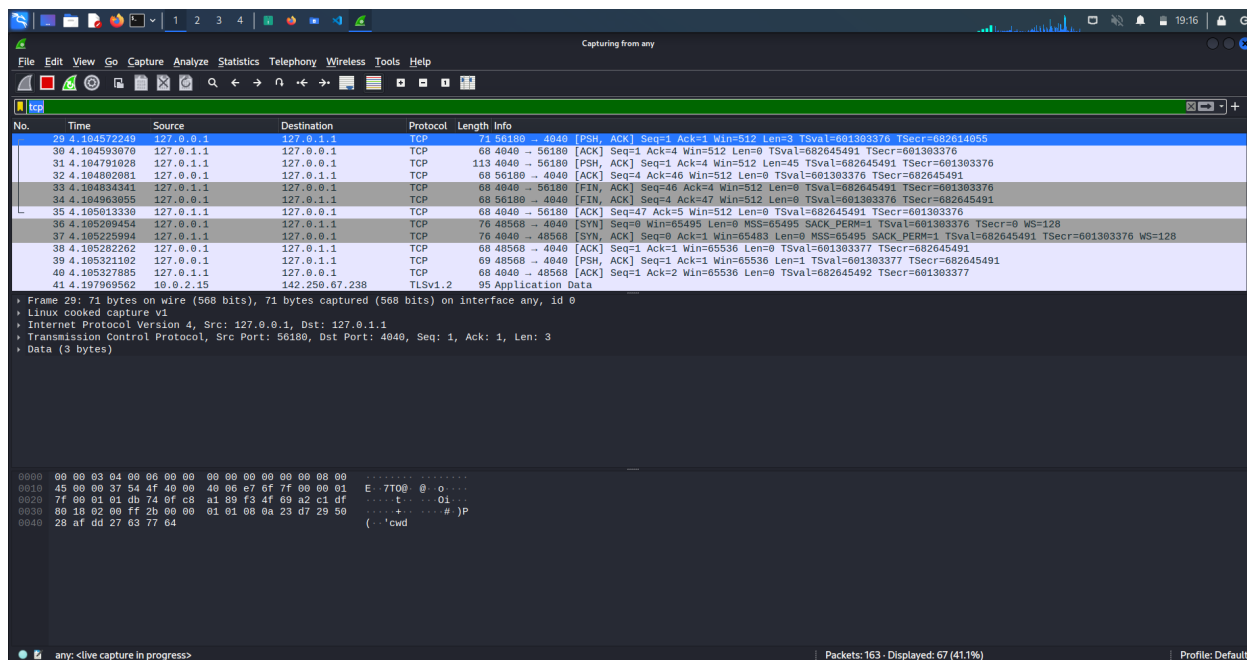


```
server.py
1 This is server file

(kali@kali)~/Downloads/Networks/client
$ python3 client.py
[*] Client Started
Enter the preferred format of encoding:
Press 0 for Plain text
Press 1 for Caesar cipher with offset 2
Press 2 for reverse encoding
0
>>> cwd
/home/kali/Downloads/Networks/server
>>> ls
ls sys, __pycache__, funcz.py, server.py, ]
>>> cd sys
/home/kali/Downloads/Networks/server/sys
>>> cd cont
/home/kali/Downloads/Networks/server/sys/cont
>>> upd client.txt
>>> dwd server.txt
>>>
```

1. Cwd

Cwd is encrypted same as cwd



No.	Time	Source	Destination	Protocol	Length	Info
29	4.184572248	127.0.0.1	127.0.0.1	TCP	71	56188 → 4040 [PSH, ACK] Seq=1 Ack=1 Win=512 Len=3 TSval=601303376 TSecr=682614895
30	4.184593078	127.0.0.1	127.0.0.1	TCP	68	4040 → 56188 [ACK] Seq=1 Ack=4 Win=512 Len=0 TSval=682645491 TSecr=601303376
31	4.184791928	127.0.0.1	127.0.0.1	TCP	113	4040 → 56188 [PSH, ACK] Seq=1 Ack=4 Win=512 Len=45 TSval=682645491 TSecr=601303376
32	4.184802081	127.0.0.1	127.0.0.1	TCP	68	56188 → 4040 [ACK] Seq=4 Ack=46 Win=512 Len=0 TSval=601303376 TSecr=682645491
33	4.184834341	127.0.0.1	127.0.0.1	TCP	68	4040 → 56188 [FIN, ACK] Seq=46 Ack=4 Win=512 Len=0 TSval=682645491 TSecr=601303376
34	4.184963095	127.0.0.1	127.0.0.1	TCP	68	56188 → 4040 [FIN, ACK] Seq=4 Ack=47 Win=512 Len=0 TSval=601303376 TSecr=682645491
35	4.185013330	127.0.0.1	127.0.0.1	TCP	68	4040 → 56188 [ACK] Seq=47 Ack=5 Win=512 Len=0 TSval=682645491 TSecr=601303376
36	4.185299454	127.0.0.1	127.0.0.1	TCP	76	48568 → 4040 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=601303376 TSecr=0 WS=128
37	4.185225994	127.0.0.1	127.0.0.1	TCP	76	4040 → 48568 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=682645491 TSecr=601303376 WS=128
38	4.185282262	127.0.0.1	127.0.0.1	TCP	68	48568 → 4040 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=601303377 TSecr=682645491
39	4.185321102	127.0.0.1	127.0.0.1	TCP	69	48568 → 4040 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=1 TSval=601303377 TSecr=682645491
40	4.185327885	127.0.0.1	127.0.0.1	TCP	68	4040 → 48568 [ACK] Seq=1 Ack=2 Win=65536 Len=0 TSval=682645492 TSecr=601303377
41	4.197969562	10.0.2.15	142.250.67.238	TLSv1.2	95	Application Data

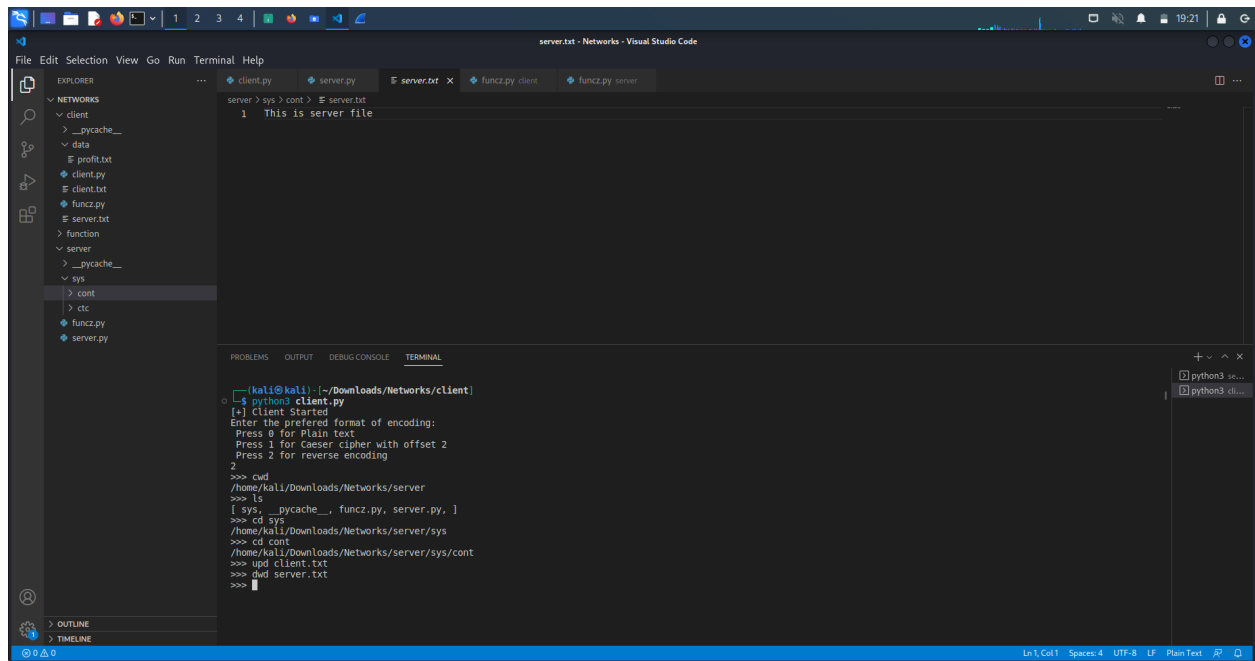
Frame 29: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface any, id 0

- Linux cooked capture v1
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- Transmission Control Protocol, Src Port: 56188, Dst Port: 4040, Seq: 1, Ack: 1, Len: 3
- Data (3 bytes)

```
0000  00 00 03 04 00 00 00 00 00 00 00 00 00 00 00 00  .....
0010  45 00 00 37 54 4f 40 00 40 00 e7 6f 7f 00 00 01  E..7T00.0.0...
0020  7f 00 01 01 0b 74 0f c8 a1 00 f3 4f 00 a2 c1 df  ....t...01...
0030  80 18 02 00 ff 2b 00 00 01 01 08 0a 23 d7 29 50  (...)+...#...)P
0040  28 af dd 27 03 77 64  ...}cwd
```

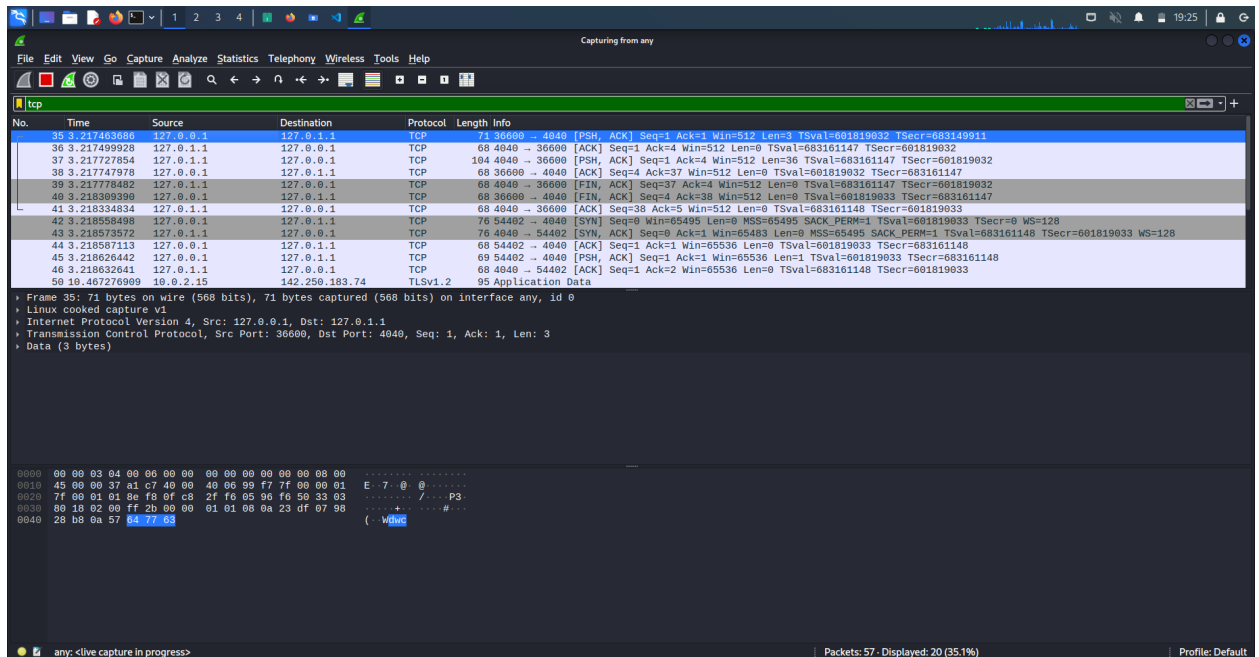
[illegible]

We have successfully analyzed caesar cipher encoding for all commands, we have analyzed request and response of all 5 commands and proved that it is working correctly. Now, we have showed that cwd is working fine with Plain text encoding style so by these two conditions it will also work same for commands like cd, ls, upd, dwd

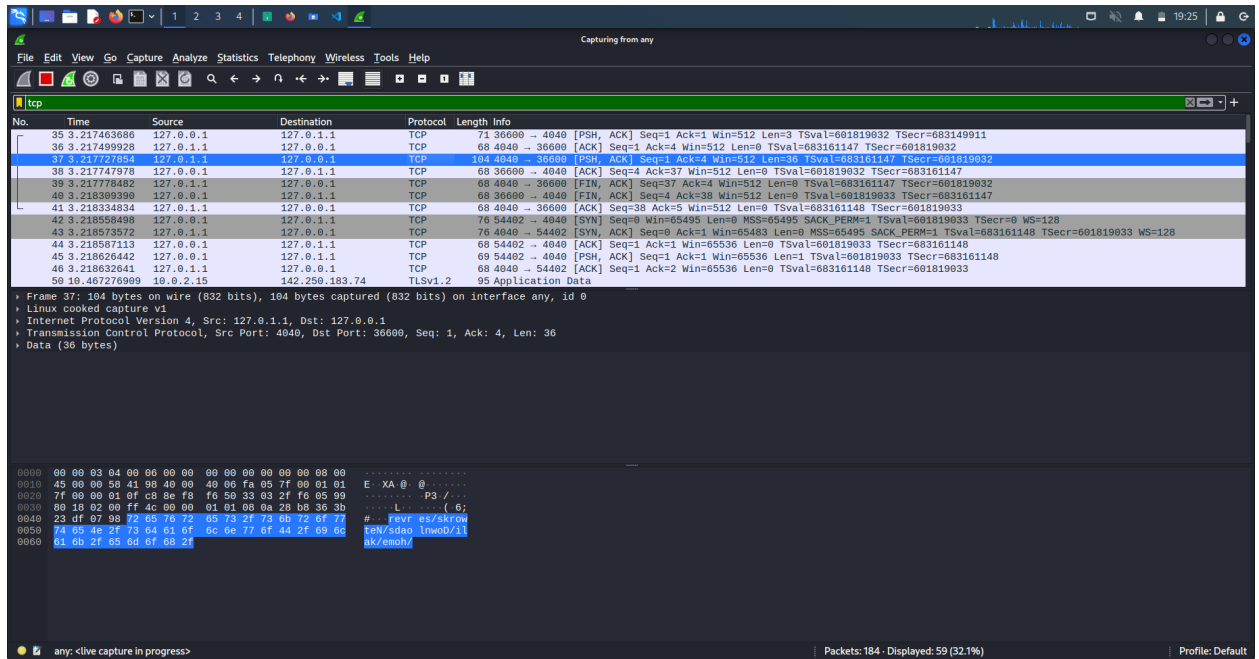


1. Cwd

Cwd is encrypted as dwc



In response we are getting current working directory in reversed fashion



We have successfully analyzed caesar cipher encoding for all commands, we have analyzed request and response of all 5 commands and proved that it is working correctly. Now, we have showed that cwd is working fine with reverse encoding style so by these two conditions it will also work same for commands like cd, ls, upd, dwd