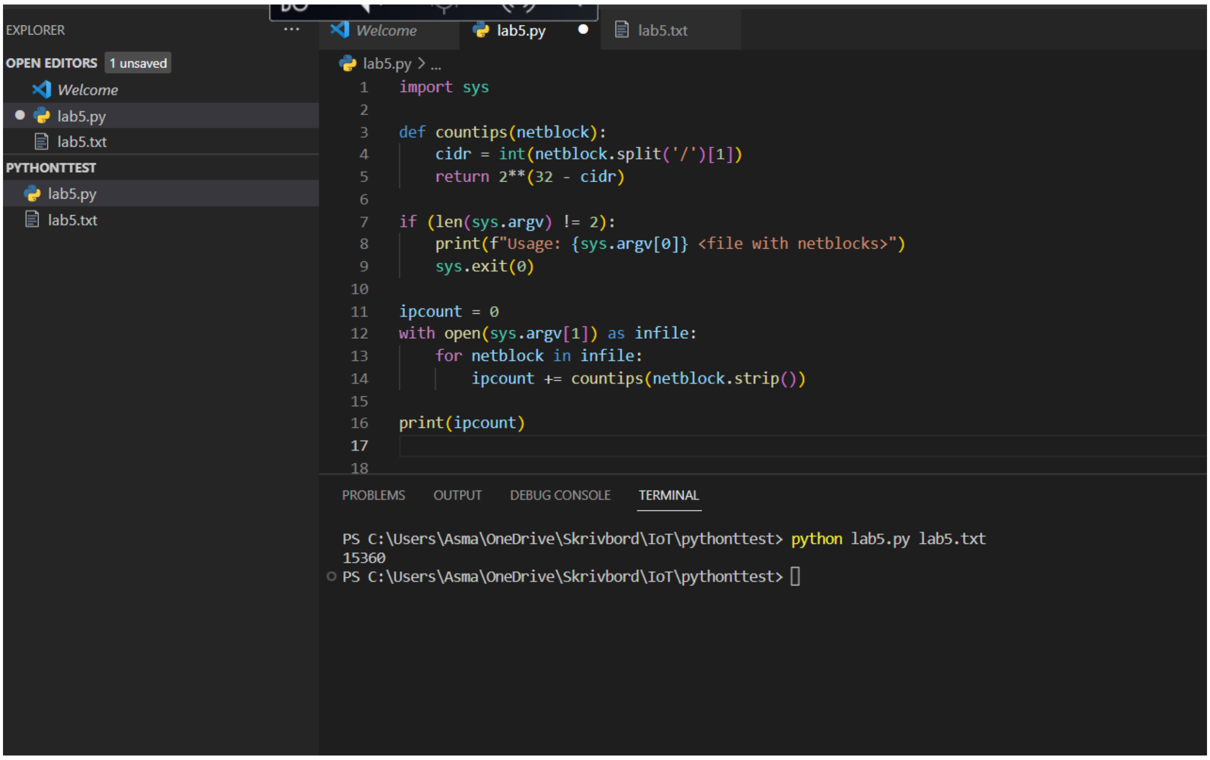
5.1.1 Cloud Scanning

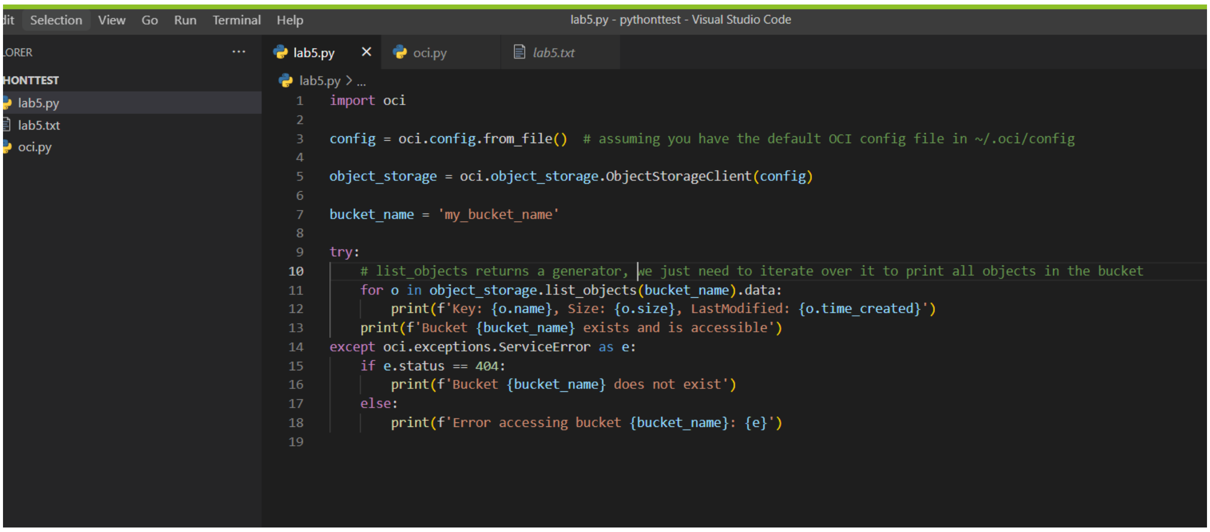
A)



B)

5.1.2 Cloud Bucket Discovery

A)



```python
import oci

config = oci.config.from_file()  # assuming you have the default OCI config file in ~/.oci/config

object_storage = oci.object_storage.ObjectStorageClient(config)

bucket_name = 'my_bucket_name'

try:
    # list_objects returns a generator, we just need to iterate over it to print all objects in the bucket
    for o in object_storage.list_objects(bucket_name).data:
        print(f'Key: {o.name}, Size: {o.size}, LastModified: {o.time_created}')
    print(f'Bucket {bucket_name} exists and is accessible')
except oci.exceptions.ServiceError as e:
    if e.status == 404:
        print(f'Bucket {bucket_name} does not exist')
    else:
        print(f'Error accessing bucket {bucket_name}: {e}')
```

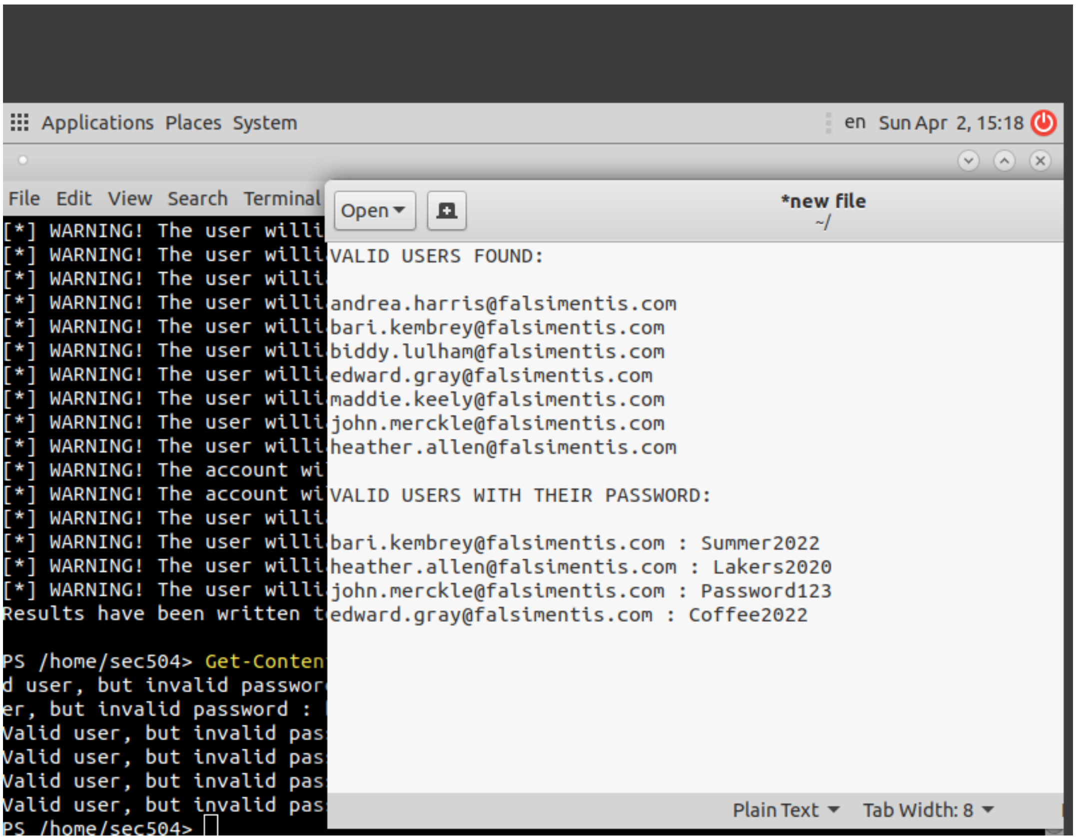
B)

If a bucket exists and is private, then a request to access the bucket will result in an access denied error. This means that the bucket exists, but the requester does not have sufficient permissions to access it.
On the other hand, if a bucket does not exist at all, then a request to access the bucket will result in a bucket not found error. This indicates that the bucket does not exist and could be due to the fact that the bucket was never created or was deleted.

5.2 SANS – Cloud Application Attacks

5.2.1 Microsoft 365 Password Attack

A)



b)

MFA can be bypassed using different methods such as,

Social engineering, where an attacker may try to trick the user into revealing their MFA code through phishing or other social engineering tactics.

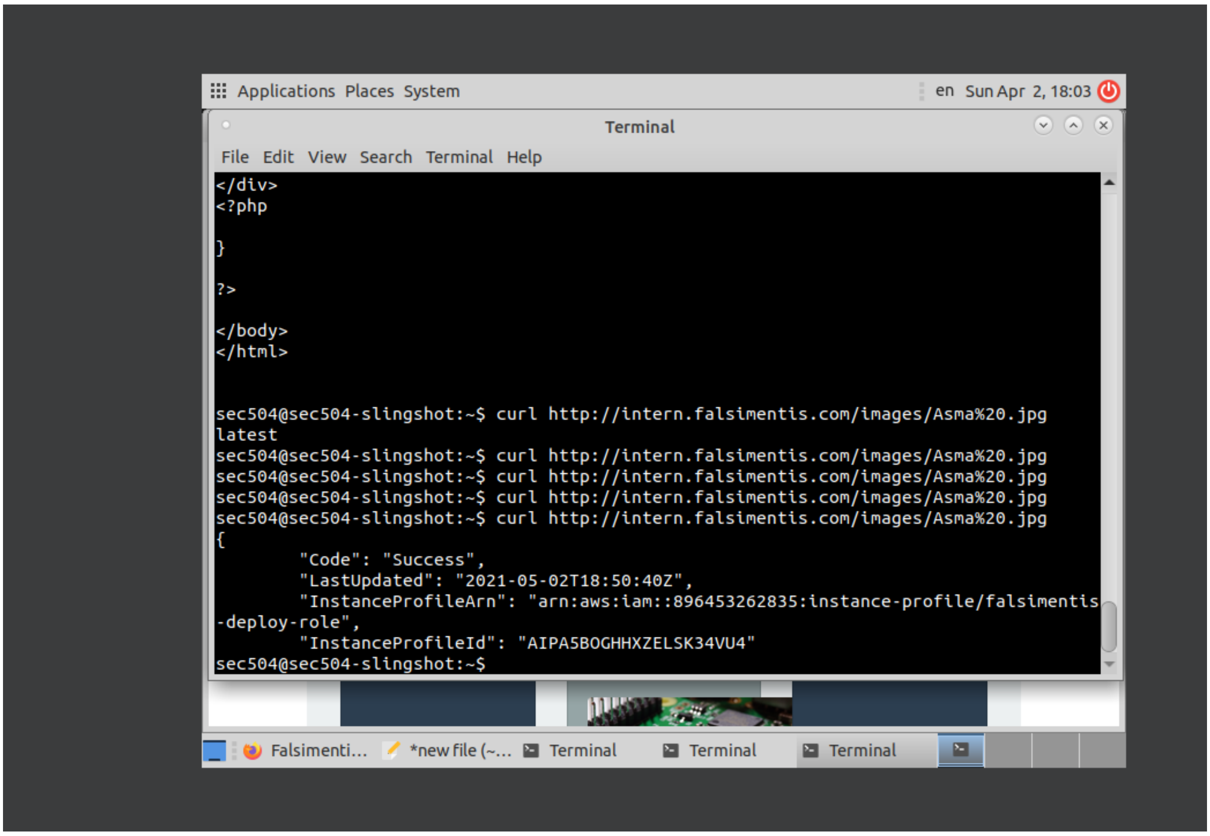
Keylogging, where an attacker could use a keylogger to record the MFA code as it is entered.
Then there is session hijacking, where the attacker could hijack a user's active session and gain access to the account without the need for MFA.
One could also set the browser user to android etc, when making the requests which may have no MFA policy.
These are just some examples, there are a lot more methods that could be used to bypass MFA.

5.2.2 Cloud SSRF/IMDS Attack

A)

The 169.254.169.254 IP address is used as a link-local address in a cloud computing environment to provide access to the metadata service provided by the cloud provider. It allows cloud instances to access important configuration data without the need for complex networking configuration.

B)



c)

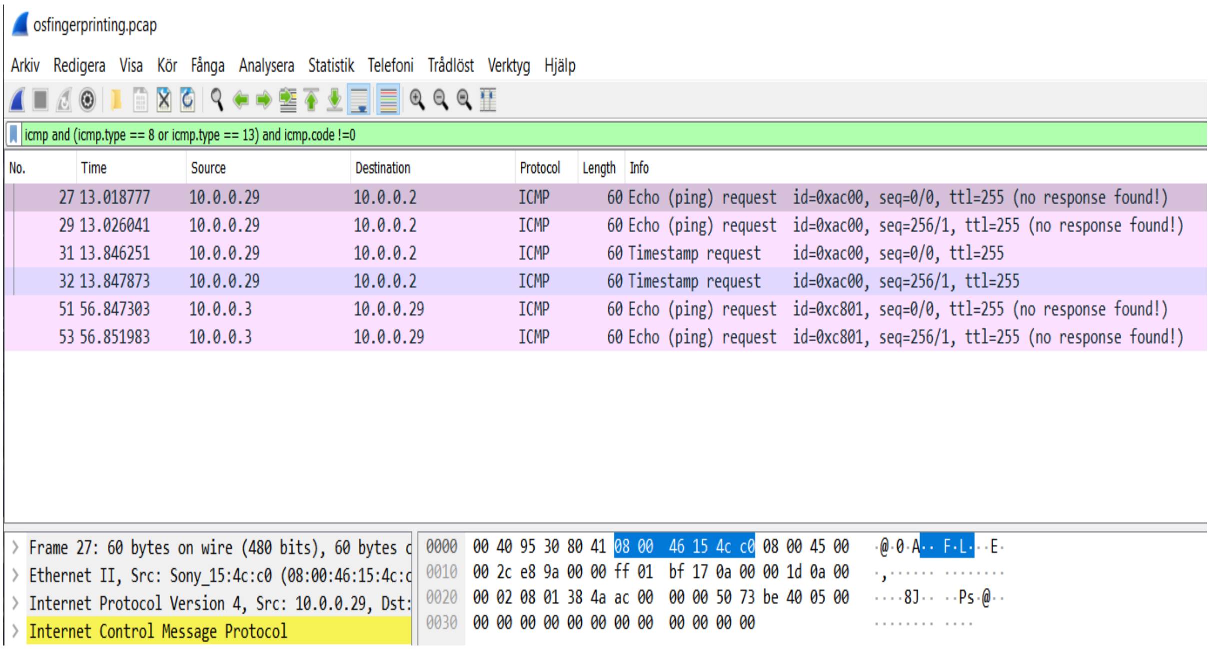
If a server-side request forgery (SSRF) vulnerability exists, it may be possible to obtain sensitive instance metadata using the ssrf.py script. The script can be used to modify the URL parameter to point to the instance metadata service endpoint, which is usually located at http://169.254.169.254/latest/meta-data/. By doing so, an attacker can obtain sensitive information such as IAM security credentials.
For instance,
http://localhost:8000/uptime?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/

d)

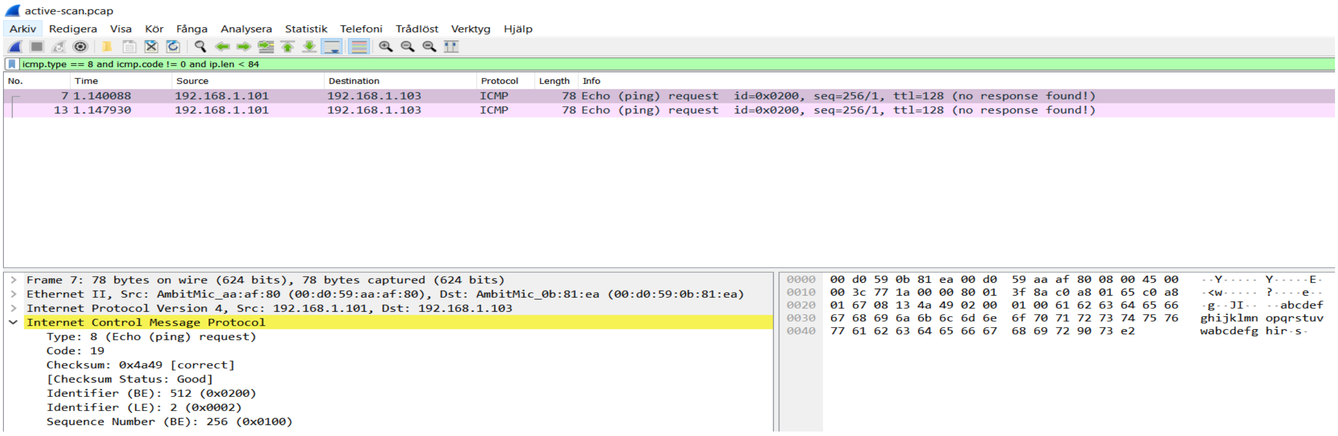
To prevent the abuse of Server-Side Request Forgery (SSRF) and (IMDS) attacks in a cloud environment, organizations should, for example, use security groups and network access control lists (ACLs) to restrict access to the IMDS.
They could implement proper input validation and sanitization in web applications and other services that may be vulnerable to SSRF attacks.
Organizations can use a strong AWS IAM policy for EC2 instances that restricts access to the IMDS.
Also a WAF to protect against SSRF attacks by blocking incoming requests that contain known SSRF payloads. By also regularly monitoring access to the IMDS and looking for unusual patterns of behavior or suspicious activity.
Other AWS security practices could be followed, such as enabling AWS Config and CloudTrail to monitor and audit access to the IMDS.

5.4.1

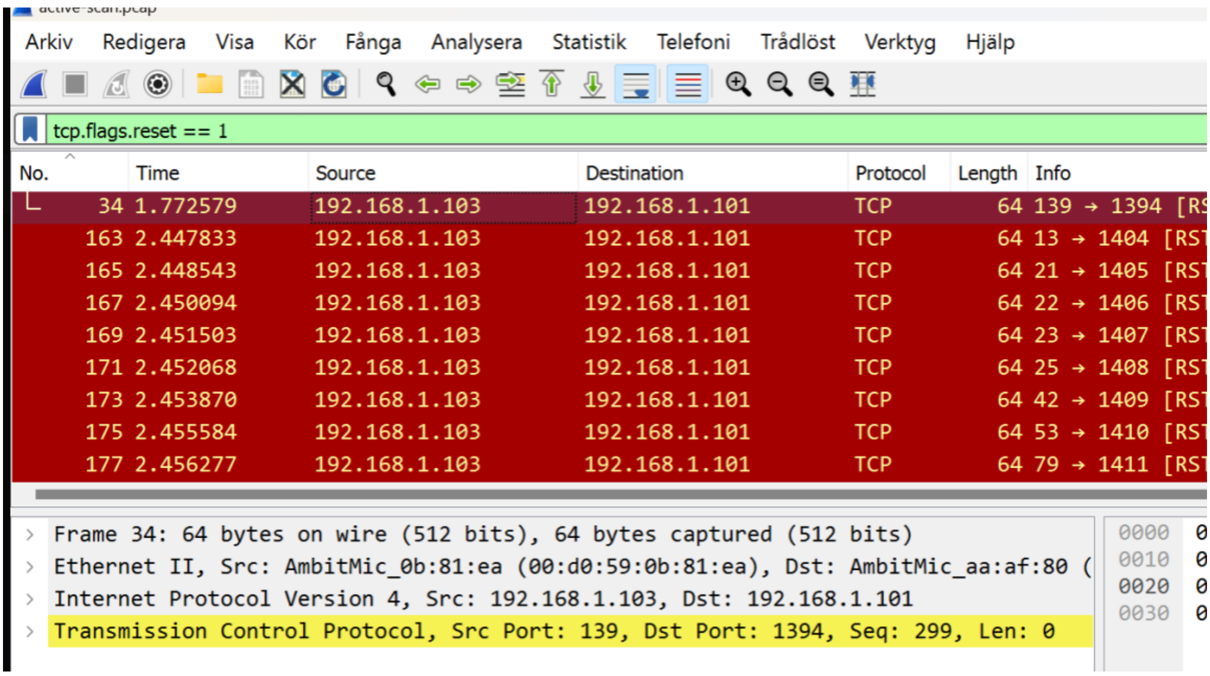
A)



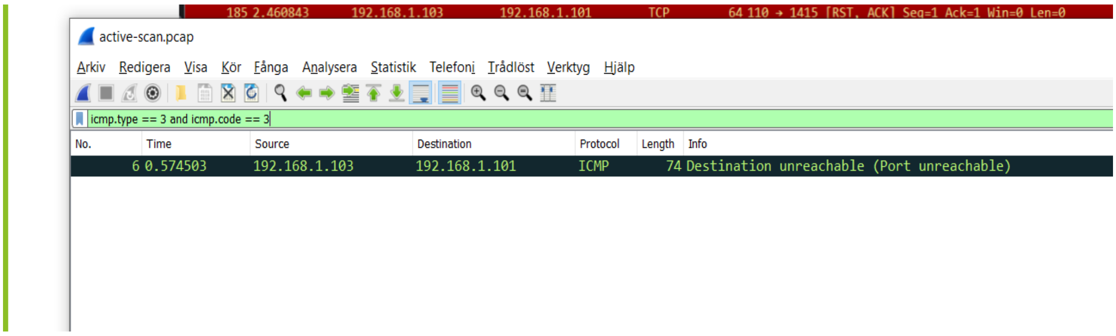
B)


active-scan.pcap
Arkiv Redigera Visa Kör Fånga Analysera Statistik Telefoni Trådlöst Verktyg Hjälp
icmp.type == 8 and icmp.code != 0 and ip.len < 84
No. Time Source Destination Protocol Length Info
7 1.140088 192.168.1.101 192.168.1.103 ICMP 78 Echo (ping) request id=0x0200, seq=256/1, ttl=128 (no response found!)
13 1.147930 192.168.1.101 192.168.1.103 ICMP 78 Echo (ping) request id=0x0200, seq=256/1, ttl=128 (no response found!)
Frame 7: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
Ethernet II, Src: AmbitMic_aa:af:80 (00:d0:59:aa:af:80), Dst: AmbitMic_0b:81:ea (00:d0:59:0b:81:ea)
Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.103
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 19
Checksum: 0x4a49 [correct]
[Checksum Status: Good]
Identifier (BE): 512 (0x0200)
Identifier (LE): 2 (0x0002)
Sequence Number (BE): 256 (0x0100)
0000 00 d0 59 0b 81 ea 00 d0 59 aa af 80 08 00 45 00
0010 00 3c 77 1a 00 00 80 01 3f 8a c0 a8 01 65 c0 a8
0020 01 67 08 13 4a 49 02 00 01 00 61 62 63 64 65 66
0030 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76
0040 77 61 62 63 64 65 66 67 68 69 72 90 73 e2

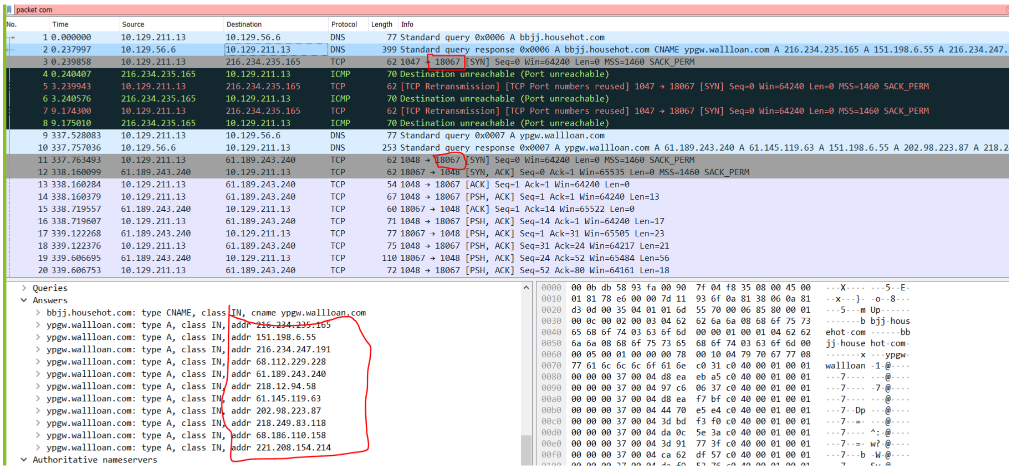
C)

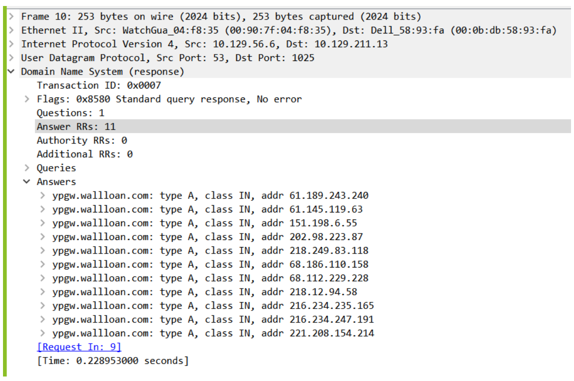

Arkiv Redigera Visa Kör Fånga Analysera Statistik Telefoni Trådlöst Verktyg Hjälp
tcp.flags.reset == 1
No. Time Source Destination Protocol Length Info
34 1.772579 192.168.1.103 192.168.1.101 TCP 64 139 → 1394 [RS
163 2.447833 192.168.1.103 192.168.1.101 TCP 64 13 → 1404 [RST
165 2.448543 192.168.1.103 192.168.1.101 TCP 64 21 → 1405 [RST
167 2.450094 192.168.1.103 192.168.1.101 TCP 64 22 → 1406 [RST
169 2.451503 192.168.1.103 192.168.1.101 TCP 64 23 → 1407 [RST
171 2.452068 192.168.1.103 192.168.1.101 TCP 64 25 → 1408 [RST
173 2.453870 192.168.1.103 192.168.1.101 TCP 64 42 → 1409 [RST
175 2.455584 192.168.1.103 192.168.1.101 TCP 64 53 → 1410 [RST
177 2.456277 192.168.1.103 192.168.1.101 TCP 64 79 → 1411 [RST
Frame 34: 64 bytes on wire (512 bits), 64 bytes captured (512 bits)
Ethernet II, Src: AmbitMic_0b:81:ea (00:d0:59:0b:81:ea), Dst: AmbitMic_aa:af:80 (
Internet Protocol Version 4, Src: 192.168.1.103, Dst: 192.168.1.101
Transmission Control Protocol, Src Port: 139, Dst Port: 1394, Seq: 299, Len: 0

D



E)

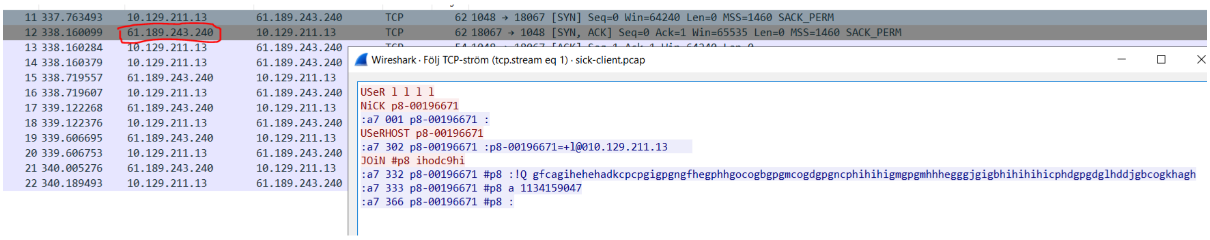




Analyzing the first packet which is the host 10.129.211.13. This host goes to 10.129.56.6 and does a DNS query to "*bbjj.househot.com*." and it gets a response from the website "*ypgw.wallloan.com.*"

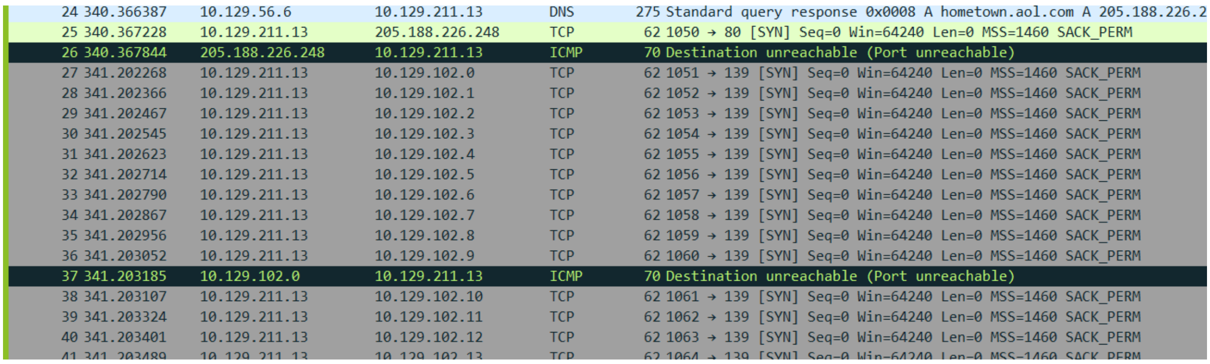
The information panel under the *"Domain Name System (response)"* in *"Answer RRs"* shows 11 records and that is not normal. It should show one or two records.

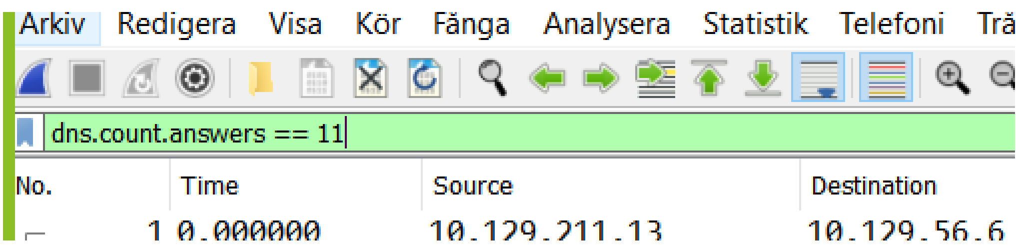
In the picture above on the 3'rd packet, the host 10.129.211.13 goes to 216.234.235.165, and it does an SYN request on port 18067 which is not a standard port. It should receive an SYN+ACK or RST response, but it only gets *"ICMP Destination Unreachable"*.
All of this is telling that something wrong is happening.

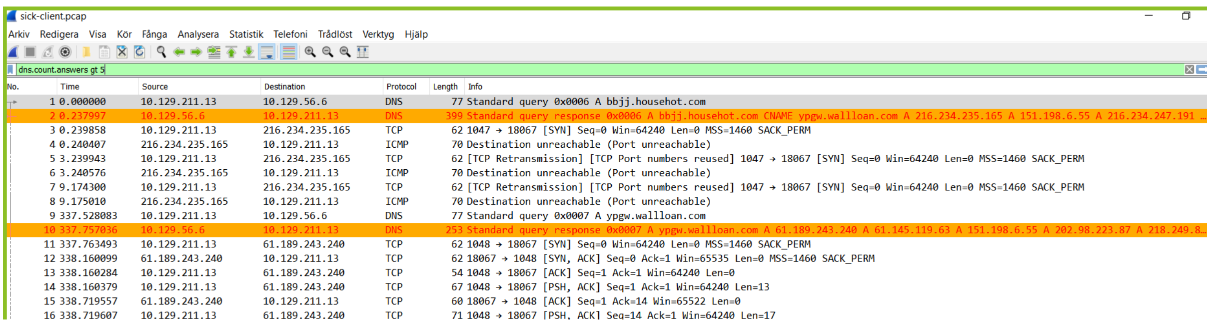


Using a *"Follow TCP Stream"* to look further into it. It clearly tells that this exchange is based on the user, nick, and join command information.

And when *"bbjj.househot.com"* searched from Google, it shows this is a bot activity.





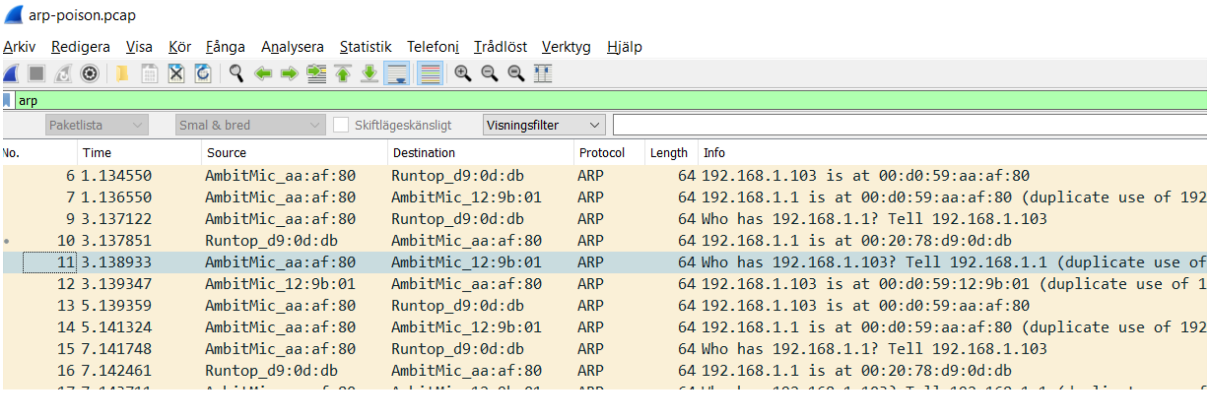


I filtered the activity and in order to detect them right away. Also I changed the selected field to "*dns.count.answers gt 5*" to detect activities that have "*Answer RRs*" with values higher than five.

F)

The protocol hierarchy shows that the majority of the traffic is using TCP, with a significant amount of UDP traffic as well. The Trivial File Transfer Protocol is being used to transfer files, which is a technique used by attackers to exfiltrate data from compromised systems. There are also some DCE/RPC protocols being used, which can be an indication of remote code execution or lateral movement. Additionally, there is Internet Relay Chat (IRC) traffic, which can be a sign of the system being used as part of a botnet. The high CPU utilization and system lockup suggest that the system is under heavy load. It could be due to malicious activity such as cryptojacking or DDoS attacks. Overall, the traffic indicates that the system is compromised and being used for unauthorized activities.

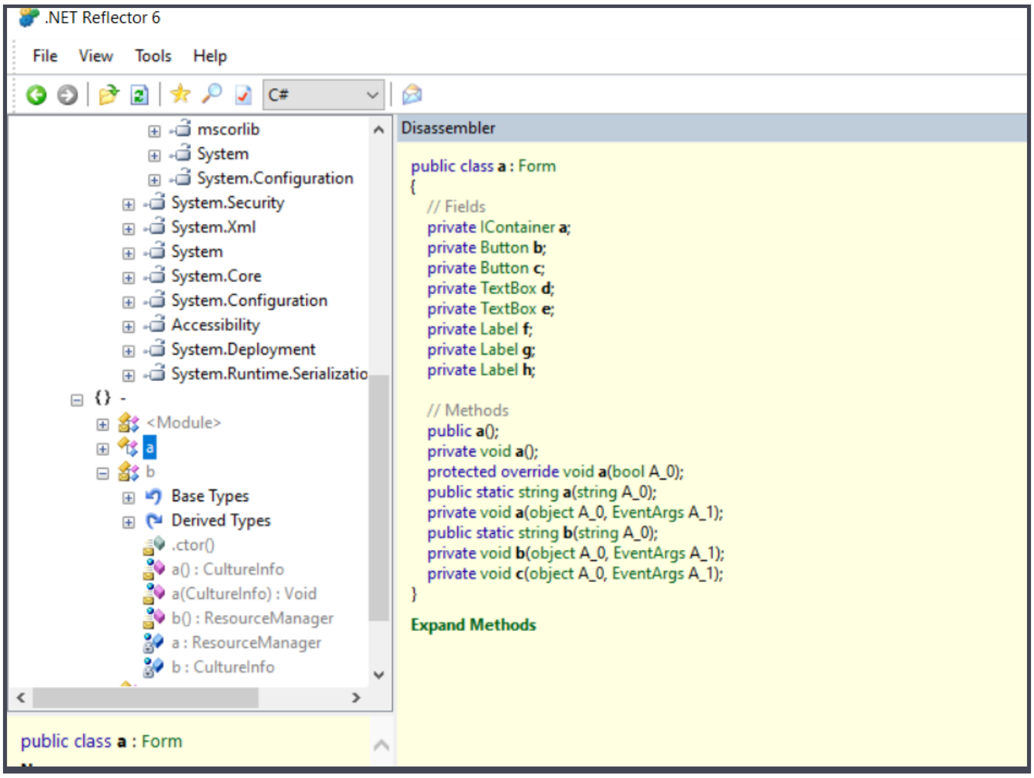
G)



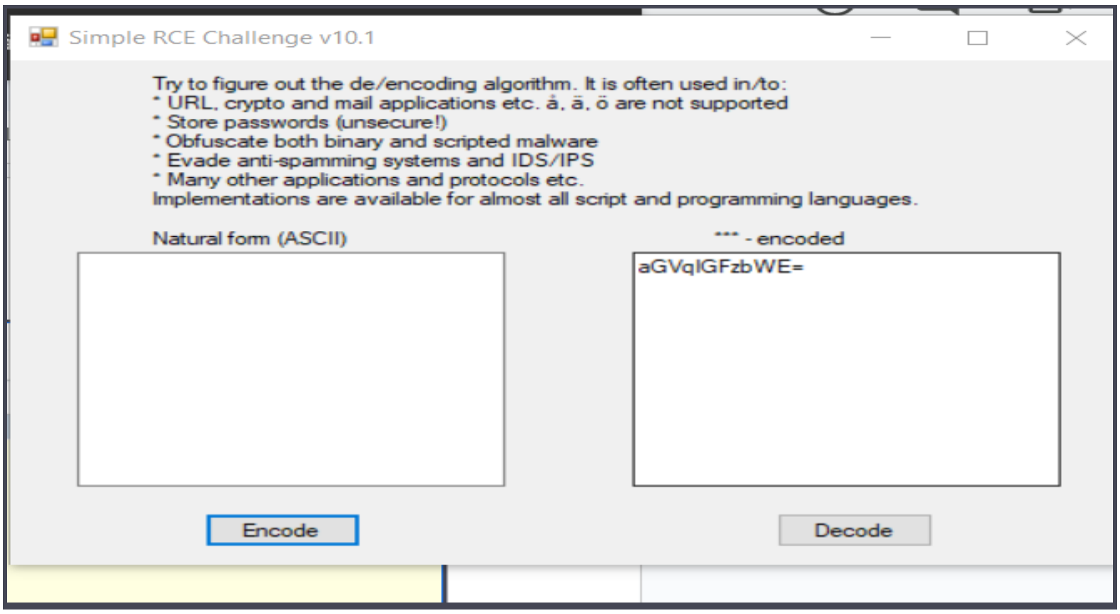
According to the data, the attacker's IP and MAC address are 192.168.1.103 and 00:d0:59:aa:af:80, respectively. The victims in this scenario are Runtop with MAC address d9:0d:db and AmbitMic_12 with MAC address 12:9b:01.

This data above shows a man-in-the-middle attack, where the attacker is intercepting and manipulating the communication between the two victims. The attacker is using ARP spoofing to perform the attack. In this case, the attacker is sending ARP packets with false information to both Runtop and AmbitMic_12, tricking them into thinking that the attacker's MAC address is the MAC address of the other victim's IP address.

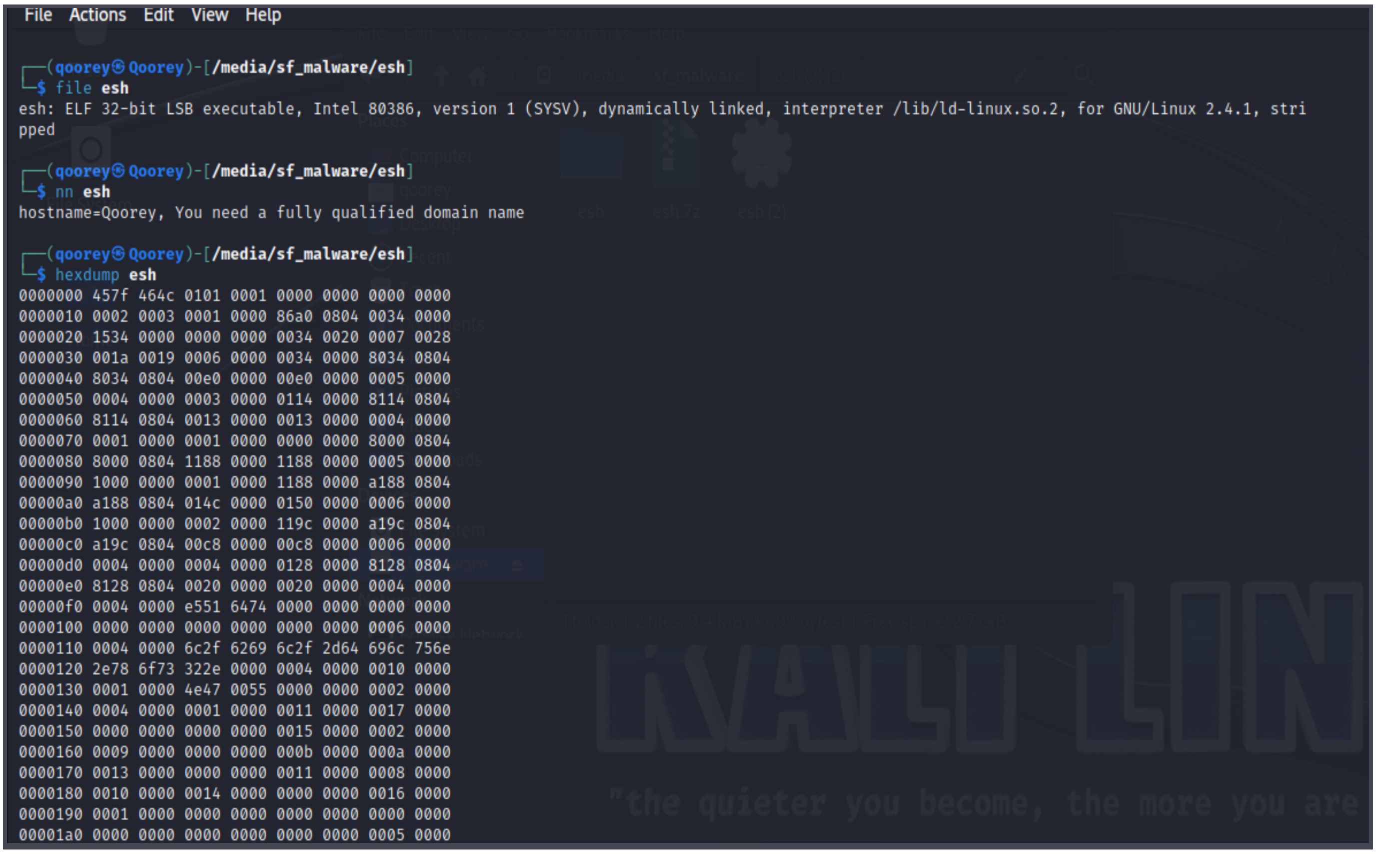
5.5 Reverse engineer managed code



The encoding and decoding algorithms are used to transform the bytecode into machine code that can be executed on the target system. The algorithm behind this program involves techniques such as code obfuscation, compression and encryption to make it more difficult for attackers to reverse engineer the code and discover its true purpose.

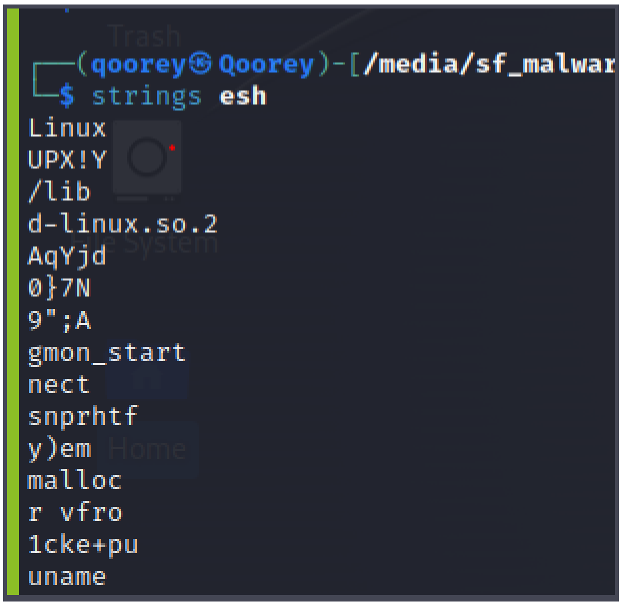


5.6 Analysis of an unknown binary file



"file" command determines the type of a given file. It can identify the format of the file, such as whether it is an executable, a library, or a text file.

The "hexdump" command displays the contents of a file in hexadecimal format.



String displays printable strings in a binary file and helps identify any hardcoded strings, URLs, or other data that the malware may u

```
┌──(qoorey㉿Qoorey)-[/media/sf_malware/esh]
└─$ objdump -d esh

esh:     file format elf32-i386


Disassembly of section .init:

0804851c <.init>:
 804851c:       55                      push   %ebp
 804851d:       89 e5                   mov    %esp,%ebp
 804851f:       83 ec 08                sub    $0x8,%esp
 8048522:       e8 9d 01 00 00          call   80486c4 <__gmon_start__@plt+0x40>
 8048527:       e8 f4 01 00 00          call   8048720 <__gmon_start__@plt+0x9c>
 804852c:       e8 3f 0b 00 00          call   8049070 <__gmon_start__@plt+0x9ec>
 8048531:       c9                      leave
 8048532:       c3                      ret

Disassembly of section .plt:

08048534 <recvfrom@plt-0x10>:
 8048534:       ff 35 6c a2 04 08       push   0x804a26c
 804853a:       ff 25 70 a2 04 08       jmp    *0x804a270
 8048540:       00 00                   add    %al,(%eax)
        ...

08048544 <recvfrom@plt>:
 8048544:       ff 25 74 a2 04 08       jmp    *0x804a274
 804854a:       68 00 00 00 00          push   $0x0
 804854f:       e9 e0 ff ff ff          jmp    8048534 <recvfrom@plt-0x10>

08048554 <close@plt>:
 8048554:       ff 25 78 a2 04 08       jmp    *0x804a278
 804855a:       68 08 00 00 00          push   $0x8
 804855f:       e9 d0 ff ff ff          jmp    8048534 <recvfrom@plt-0x10>

08048564 <fork@plt>:
 8048564:       ff 25 7c a2 04 08       jmp    *0x804a27c
 804856a:       68 10 00 00 00          push   $0x10
 804856f:       e9 c0 ff ff ff          jmp    8048534 <recvfrom@plt-0x10>

08048574 <signal@plt>:
 8048574:       ff 25 80 a2 04 08       jmp    *0x804a280
 804857a:       68 18 00 00 00          push   $0x18
```

The "objdump -d" is used to disassemble a binary file into its assembly code.

```
┌──(qoorey㉿Qoorey)-[/media/sf_malware/esh]
└─$ nm esh
nm: esh: no symbols
```

This command displays symbols from the files. It is helpful in malware analysis to identify the functions and variables within the binary file. In my case I did not get anything.

```
┌──(qoorey㉿Qoorey)-[/media/sf_malware/esh]
└─$ ldd esh
        linux-gate.so.1 (0×f7ef9000)
        libc.so.6 ⇒ /lib32/libc.so.6 (0×f7c00000)
        /lib/ld-linux.so.2 (0×f7efb000)

┌──(qoorey㉿Qoorey)-[/media/sf_malware/esh]
└─$ 
```

The "ldd" command is used to print the shared object dependencies required by a binary file. It can help identify which libraries the file relies on, which can provide insight into its behavior.

```
┌──(qoorey㉿Qoorey)-[/media/sf_malware/esh]
└─$ upx -d esh
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96        Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
      6468 <-      4968   76.81%   linux/i386    esh

Unpacked 1 file.

┌──(qoorey㉿Qoorey)-[/media/sf_malware/esh]
```

"upx -d" command is used to unpack a binary file that has been compressed or obfuscated using the UPX packer