PROJECT 2

Disclaimer:
In this report we do not explicitly consider the acknowledgement (ACK) and other control messages involved in TCP communication like all the various control messages, such as RSTs, SYN, and more, which play a critical role in maintaining the reliability, flow control, and error handling of network communications. The provided comments focus only on the security aspects.

INITIAL SETUP

In this chapter we will explain how to set up the kali vm that we used as Mallory.
This has to be done since the kali vm does not belong to the same LAN network of Alice and Bob and so it couldn't see their messages without this initial configuration.

On Kali (for part c):

1. We start by adding a new ethernet interface eth0 by changing the file:
   /etc/network/interfaces
   And adding this lines:
   auto eth0
   iface eth0 inet static
   address 192.168.1.3
   netmask 255.255.255.0
   In this way we have given a static ip address to eth0 with the same netmask as Alice and Bob and in their same network

2. To start the ethernet interface it is only necessary to type in:
   ifup eth0

3. Now since Kali does not know to whom the ip 192.168.1.1 belongs to, doing Part C would generate an error when Alice tries to navigate to https://bob/ since, as mentioned, kali is unable to associate the name bob with its ip address.
   To avoid this, it is necessary to modify the file:
   /etc/hosts
   And add:
   192.168.1.1 bob
   192.168.1.2 alice

4. Then we need to allow Kali to behave as a sort of gateway since it will receive packets from Alice and needs to forward them to Bob, and vice versa.
   To do this we uncomment the line:
   net.ipv4.ip_forward=1
   Inside the file:
   /etc/sysctl.conf

And to make the change effective we need to run:
sysctl -p /etc/sysctl.conf

5. Finally we change the settings in mitmproxy:
   connection_strategy -> lazy
   tls_version_client_min -> TLS1
   tls_version_server_min -> TLS1
   This has to be in order to avoid errors when trying to open https://bob/ on Alice

On alice:

1. ONLY FOR PART C
   We have to change firefox settings to set Kali as the proxy server:
   - HTTP_Proxy: 192.168.1.3 and Port: 8080
   - SSL_Proxy: 192.168.1.3 and Port: 8080

Once all these steps have been done, you can then proceed with the exercises.

PART A

0.688857106	PcsCompu_ed:5b:f5	Broadcast	ARP	60 Who has 192.168.1.1? Tell 192.168.1.2
0.689348625	PcsCompu_2a:ab:8d	PcsCompu_ed:5b:f5	ARP	60 192.168.1.1 is at 08:00:27:2a:ab:8d

In the first line of the capture we can observe the broadcast arp request from alice (ip address 192.168.1.2) to discover who has the ip address 192.168.1.1.
The second message is the ARP reply from bob (whose address is 192.168.1.1) to alice.
The next few messages are used to negotiate specific information, for example about window size,....

0.787254029	192.168.1.1	192.168.1.2	TELNET	69 Telnet Data ...
0.787982026	192.168.1.2	192.168.1.1	TELNET	69 Telnet Data ...
0.800631518	192.168.1.1	192.168.1.2	TELNET	69 Telnet Data ...
0.801528255	192.168.1.2	192.168.1.1	TELNET	69 Telnet Data ...

These 4 messages contain different information about echos that client (Alice) and server (Bob) want to do.

- The first is a Do Echo from Bob in which asks to see the characters typed in it's own screen
- The second is a Won't Echo from Alice. This means that Alice won't echo back the characters that Bob types in.
- The third message is a Will Echo from Bob which means that Bob will echo back the characters typed in by Alice
- The fourth message contains a Do Echo from Alice.

Now the communication begins and the server asks for Bob's credential.

```
21 0.842086880   192.168.1.1        192.168.1.2        TELNET     77 Telnet Data ...
Telnet
    Data: bob login:
```

It is possible to see in clear the username sent letter by letter from Alice to Bob because Telnet is character-oriented. In the image we can see the first character being sent.
We are able to see the whole username but for clarity we put only the first letter.

```
23 2.957902185   192.168.1.2        192.168.1.1        TELNET     67 Telnet Data ...
Telnet
    Data: b
```

After the username insertion, the server asks for the password:

```
35 4.469351375   192.168.1.1        192.168.1.2        TELNET     76 Telnet Data ...
Telnet
    Data: Password:
```

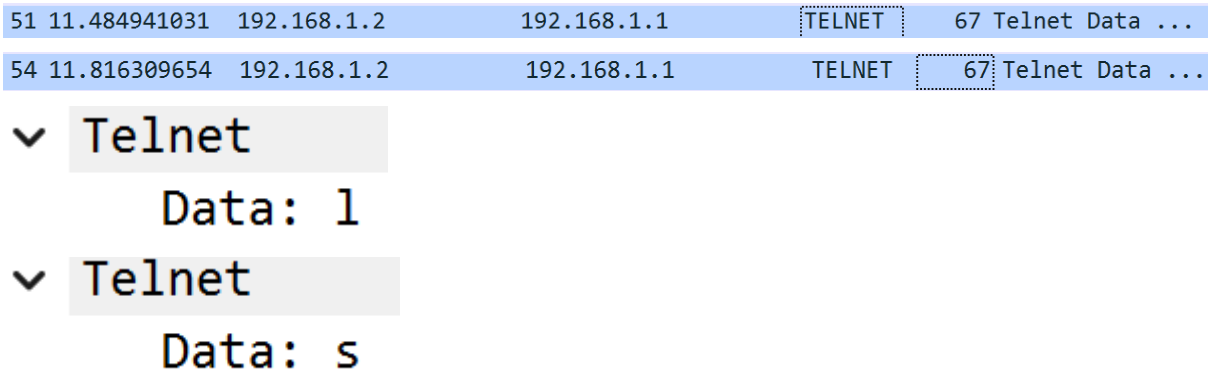
It's also possible to see in clear the password which is sent letter by letter from Alice to Bob. like before we only put the screenshot of the first character but we are able to see the whole password.

```
37 5.715821206   192.168.1.2        192.168.1.1        TELNET     67 Telnet Data ...
Telnet
    Data: b
```

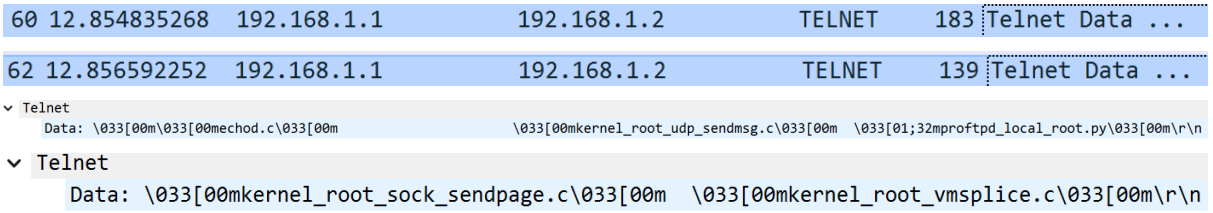
After the password is inserted the login happens successfully and first information are displayed on Alice's terminal

```
47 6.396794373   192.168.1.1        192.168.1.2        TELNET    487 Telnet Data ...
Telnet
    Data: Last login: Wed Oct 18 16:59:42 CEST 2023 on tty1\r\n
    Data: Linux bob 2.6.17.1 #1 SMP Mon Jul 5 18:50:04 CEST 2010 i686\r\n
    Data: \r\n
    Data: The programs included with the Debian GNU/Linux system are free software;\r\n
    Data: the exact distribution terms for each program are described in the\r\n
    Data: individual files in /usr/share/doc/*/copyright.\r\n
    Data: \r\n
    Data: Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent\r\n
    Data: permitted by applicable law.\r\n
    Data: You have mail.\r\n
```

At this point Alice sends the ls command:



And it is now possible to see the 5 different files in Bob's directory inside some escape characters(\033[00m)



After this Alice logs out and it is possible to see in clear every character typed in and the communication ends.

PART B

setup:

ON bob we run, from command line, the following:

1. `su` to get root access,
2. `apt-get update`,
3. `apt-get install apache2`,
4. `a2enmod ssl`,
5. `a2ensite default-ssl`,
6. `/etc/init.d/apache2 reload`

in this way we are able to enable SSL

findings:

the first few messages are the same as in the PART A, but after those messages then we are able to see the tls handshake:

```
4 0.001626172   192.168.1.2     192.168.1.1     TLSv1   222 Client Hello
```

It begins with the client Alice sending a "ClientHello" message to the server (Bob). This message includes information about the TLS version the client supports, the cryptographic algorithms it can use, and a random value.

```
6 0.002998519   192.168.1.1     192.168.1.2     TLSv1   549 Server Hello, Certificate, Server Hello Done
```

The server responds with a "ServerHello" message. In this response, the server chooses a TLS version, cipher suite, and provides its own random value. the server also sends its digital certificate to the client:

```
TLSv1 Record Layer: Handshake Protocol: Certificate
   Content Type: Handshake (22)
   Version: TLS 1.0 (0x0301)
   Length: 416
 Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 412
      Certificates Length: 409
   Certificates (409 bytes)
         Certificate Length: 406
      Certificate: 308201923081fc020900b8711d471d6c7aa2300d06092a864886f70d0101050500300e31… (…
         signedCertificate
         algorithmIdentifier (sha1WithRSAEncryption)
         Padding: 0
         encrypted: 95d0be1775127e7c325264e46b149c2295d2981a3dbe415faa803a0811e82a90d10e3e88…
```

```
8 0.006932750   192.168.1.2     192.168.1.1     TLSv1   264 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
9 0.008794297   192.168.1.1     192.168.1.2     TLSv1   332 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
```

the server sends information to the client that will be used to derive encryption keys for securing the communication. Once the handshake is complete, the client and server can begin exchanging encrypted data using the derived session keys. All data transmitted between them is encrypted and can only be decrypted by the intended receiver, who has the key to decrypt.
So from now on what we expected was to not be able to see anything in clear, only encrypted messages which is true until we get to the form submission.
example of encrypted messages:

```
144 0.298101531   192.168.1.1     192.168.1.2     TLSv1    876 Application Data, Application Data
145 0.298787260   192.168.1.1     192.168.1.2     TLSv1   1340 Application Data, Application Data
146 0.299323781   192.168.1.2     192.168.1.1     TLSv1    631 Application Data
147 0.299323858   192.168.1.1     192.168.1.2     TLSv1   4410 Application Data
148 0.299861593   192.168.1.1     192.168.1.2     TLSv1    764 Application Data, Application Data
```

```
TLSv1 Record Layer: Application Data Protocol: Hypertext Transfer Protocol
   Content Type: Application Data (23)
   Version: TLS 1.0 (0x0301)
   Length: 416
   Encrypted Application Data: bd579a94230ffd47485647e1e67bb6b77a40f06d56e209ea7448ff436c53174dcd56
   [Application Data Protocol: Hypertext Transfer Protocol]
```

The content of this encrypted messages are for example the images that were displayed one after the other on Bob's website.

Unexpectedly in the capture we noticed that this message:

```
227 15.702325099  192.168.1.2     192.168.1.1     HTTP    792 POST /index.php?option=com_user&task=login HTTP/1.1  (application/x-www-form-urlencoded)
```

is send through the protocol HTTP and not TLSv1, when alice logs in, so it is not encrypted and we are able to see all Alice's credentials:

```
HTML Form URL Encoded: application/x-www-form-urlencoded
 > Form item: "username" = "alice"
 > Form item: "passwd" = "alice123"
 > Form item: "remember" = "yes"
 > Form item: "Login" = "Login"
 > Form item: "op2" = "login"
 > Form item: "return" = "aW5kZXgucGhwP29wdGlvbj1jb21fY29udGVudCZ2aWV3PWZyb250cGFnZQ=="
 > Form item: "7cb3d8f94da96476c82115ad260b0532" = "1"
```

This happens because only https://bob/ is encrypted with the tls protocol but once Alice fill out the form and click the button to log in, she gets redirected to an http website which is not encrypted anymore and so we are able to see Alice's credentials.
This should not happen on a normal website and, in our case, we should not have seen Alice's credentials but only other data encrypted via tls protocol.

PART C

In this scenario Mallory has successfully set up a Man-in-the-Middle (MitM) proxy on Alice's computer and captured traffic with Wireshark. In practice Mallory places herself between Alice and the web server. Mallory effectively intercepts all traffic that passes between Alice and Bob's web server and With the SSL/TLS keys in hand Mallory will be able to decrypt the encrypted data exchanged between Alice and Bob's web server.

Mallory ensures that both Alice and Bob are unaware of the intrusion. The success of the MitM attack relies on keeping the attack hidden and so, every packet from Alice is forwarded to Bob and every packet from Bob is forwarded to Alice. So even if it is not shown in every step Mallory always forwards the packet as described.

in the capture what we can see is:
the initiation of the HTTPS connection to the https://bob/ :

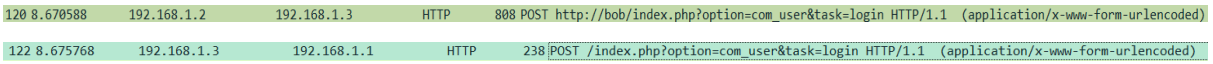
4	0.005530	192.168.1.2	192.168.1.3	HTTP	252	CONNECT bob:443 HTTP/1.1
5	0.005666	192.168.1.3	192.168.1.2	TCP	66	8080 → 41990 [ACK] Seq=1 Ack=187 Win=65024 Len=0 TSval=588396721 TSecr=287236
6	0.007679	192.168.1.3	192.168.1.2	HTTP	105	HTTP/1.1 200 Connection established
7	0.009367	192.168.1.2	192.168.1.3	TCP	66	41990 → 8080 [ACK] Seq=187 Ack=40 Win=5856 Len=0 TSval=287237 TSecr=588396723

Then we see the TLS Handshake: ClientHello, ServerHello, Certificate Exchange, and the establishment of a secure session.
In this way Mallory, who has now intercepted the TLS keys, can decrypt and view all the encrypted traffic.

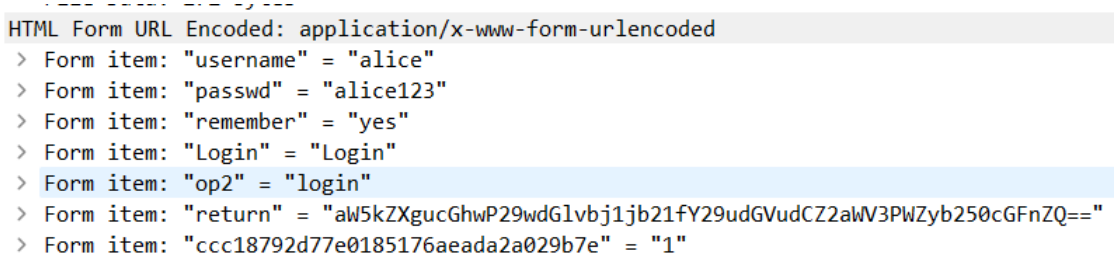
8	0.011133	192.168.1.2	192.168.1.3	TLSv1	414	Client Hello
9	0.015837	192.168.1.3	192.168.1.2	TLSv1	211	Server Hello, Change Cipher Spec, Encrypted Handshake Message
10	0.024452	192.168.1.2	192.168.1.3	TLSv1	610	Change Cipher Spec, Encrypted Handshake Message, Application Data

14	0.039350	192.168.1.3	192.168.1.1	TLSv1	377	Client Hello
15	0.042781	192.168.1.1	192.168.1.3	TCP	66	443 → 33372 [ACK] Seq=1 Ack=312 Win=6864 Len=0 TSval=6153453 TSecr=1863981111
16	0.043844	192.168.1.1	192.168.1.3	TLSv1	549	Server Hello, Certificate, Server Hello Done
17	0.044144	192.168.1.3	192.168.1.1	TCP	66	33372 → 443 [ACK] Seq=312 Ack=484 Win=64128 Len=0 TSval=1863981116 TSecr=6153453
18	0.045883	192.168.1.3	192.168.1.1	TLSv1	264	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
19	0.048818	192.168.1.1	192.168.1.3	TLSv1	332	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message

With the SSL/TLS keys, Mallory can decrypt the encrypted data exchanged between Alice and Bob's web server. Mallory can view the data in plaintext including the login credentials she uses:

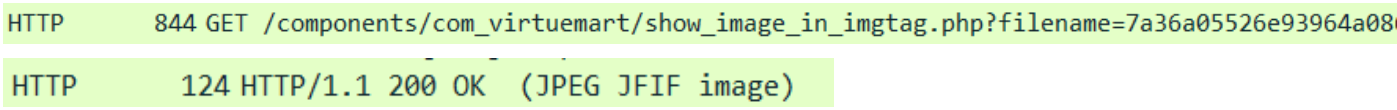


She sees the HTTP POST request with Alice's login credentials from the message that alice sends to her:



Mallory can see this login data in plain text due to her ability to decrypt the SSL/TLS-encrypted traffic. Then she forwards the message to bob.

The last thing Bob is able to see is Alice’s log out.
These other messages below:



are the ones that implement the images shown on the website.