

# SSN-4th-Lab-JOEL\_OKORE

## TASK 1 - Introduction

1a. What is PKI, and what is the purpose of PKI with OpenVPN?

Public Key Infrastructure (PKI) is a system used to manage public-private key pairs and digital certificates, enabling safe network communication and identity authentication. It is a framework comprising of policies, roles and procedures required to create, manage, distribute, use, store and revoke digital certificates. The X.509 standard specifies what a digital certificate used within a Public Key Infrastructure should contain, for example, version number, serial number, certificate algorithm identifier, Certificate Authority (CA) signature, e.t.c.

PKI, in the context of OpenVPN, secure connection between server and client by providing:

- Authentication: The server and clients authenticate each other (mutual authentication) using certificates, preventing man-in-the-middle attacks.
- Encryption: Private Communication is ensured by encrypting data, using the public-private key pairs
- Certificate Management: The server, as a Certificate Authority (CA), issues and manages client certificates for secure access.
- Revocation: A Certificate Revocation List (CRL) can revoke compromised certificates to block unauthorized access by clients with compromised certificates.

1b. Distinguish between a master certificate authority (CA) and a separate certificate authority (CA)

A Master Certificate Authority (CA) is the root of trust in a PKI. It is generally self-signed and kept offline for security purposes. It also delegates responsibility of issuing certificates to Separate Certificate Authority (CAs), also called subordinate or intermediate CAs. These subordinate CAs are signed by the master CA. The subordinate CAs are usually kept online, and handle most of the certificate management and issuing, thereby providing flexibility and scalability.

## TASK 2- Setup

I started by installing OpenVPN by using the advanced package tool. See picture 1.

### Picture 1 - Installing OpenVPN

- A separate certificate (also known as a public key) and private key for the server and each client
- A primary Certificate Authority (CA) certificate and key, used to sign the server and client certificates

SSN-4th-Lab-JOEL OKORE

```
(okore_joel@kali) - [~]
$ sudo make-cadir /etc/openvpn/easy-rsa
[sudo] password for okore_joel:
```

Picture 2 - Copy easy-rsa/ directory to /etc/openvpn directory

In order to initialize a PKI and build a Certificate Authority (CA), I executed the 'easy-rsa' program with 'init-pki' and 'build-ca' arguments respectively as illustrated in pic. 3

```
(okore_joel@kali) - [~]
$ sudo /etc/openvpn/easy-rsa/./easyrsa init-pki
Notice
-----
'init-pki' complete; you may now create a CA or requests.
Your newly created PKI dir is:
* /home/okore_joel/pki

Using Easy-RSA configuration:
* undefined

(okore_joel@kali) - [~]
$ sudo /etc/openvpn/easy-rsa/./easyrsa build-ca
Enter New CA Key Passphrase:
Confirm New CA Key Passphrase:
.....
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [Easy-RSA CA]:okore_joel
Notice
-----
CA creation complete. Your new CA certificate is at:
* /home/okore_joel/pki/ca.crt

(okore_joel@kali) - [~]
$
```

Picture 3 - Initializing PKI and Building a CA.

2b. What did you notice about the prompt when creating a CA? What is the reason behind creating and using a passphrase?

The prompt requested a passphrase to safeguard the private key linked to the Certificate Authority (CA). This is so because the integrity and security of the CA depend on the private key. If the attacker gains access, they could issue false certificates.

A passphrase improves security by requiring authentication in order to use the private key of the CA. It also encrypts the key, making it unusable in the event that the key is stolen.

2c. Generate Diffie Hellman parameters and key pair for the server. Show the location(path) of the key pair generated.

[illegible]

To generate Diffie Hellman parameters which will be used during connection for key exchange, I utilized the 'gen-dh' argument when executing the 'easysrsa' program. See picture 5.

[illegible]

Path to server key pair: /home/okore\_joel/pki/private/vpn\_server.key

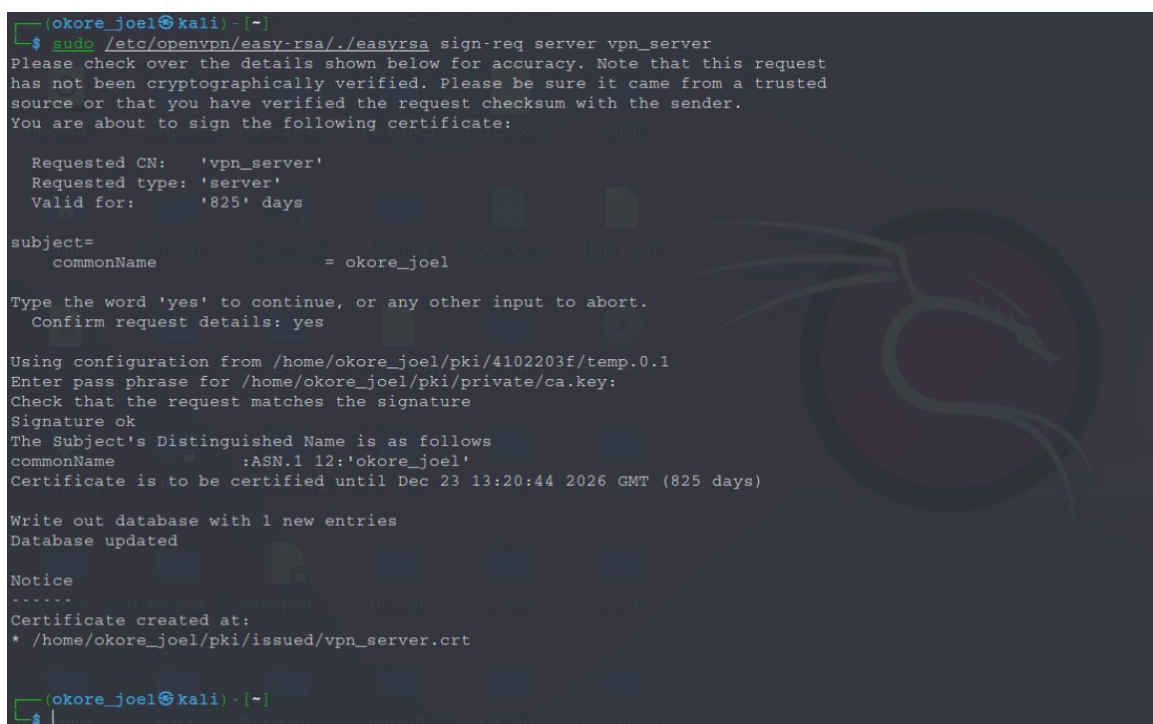
Path to Diffie Hellman parameters: /home/okore\_joel/pki/dh.pem

2d. What is the size of the Diffie Hellman parameters and their locations?

The size of the Diffie Hellman Parameters is 2048 bits and they are located in /home/okore\_joel/pki/dh.pem.

2e. Create a certificate for the server. What is the commonName value, and how many days are the certificates signed on the server?

To create a certificate for the server, I used the 'sign-req' argument and specified the server name as 'vpn\_server', as shown in picture 6.

A terminal window showing the execution of the 'easyrsa sign-req server vpn\_server' command. The output displays the requested certificate details: 'vpn\_server' for the common name, 'server' for the type, and 825 days for the validity period. It prompts for confirmation, which is given as 'yes'. The terminal then shows the signing process, including the use of the CA key and the final creation of the certificate at /home/okore\_joel/pki/issued/vpn\_server.crt.

```
(okore_joel@kali) ~  
$ sudo /etc/openvpn/easy-rsa/./easyrsa sign-req server vpn_server  
Please check over the details shown below for accuracy. Note that this request  
has not been cryptographically verified. Please be sure it came from a trusted  
source or that you have verified the request checksum with the sender.  
You are about to sign the following certificate:  
  
Requested CN: 'vpn_server'  
Requested type: 'server'  
Valid for: '825' days  
  
subject=  
commonName = okore_joel  
  
Type the word 'yes' to continue, or any other input to abort.  
Confirm request details: yes  
  
Using configuration from /home/okore_joel/pki/4102203f/temp.0.1  
Enter pass phrase for /home/okore_joel/pki/private/ca.key:  
Check that the request matches the signature  
Signature ok  
The Subject's Distinguished Name is as follows  
commonName :ASN.1 12:'okore_joel'  
Certificate is to be certified until Dec 23 13:20:44 2026 GMT (825 days)  
  
Write out database with 1 new entries  
Database updated  
  
Notice  
-----  
Certificate created at:  
* /home/okore_joel/pki/issued/vpn_server.crt  
  
(okore_joel@kali) ~  
$
```

Picture 6 - Creating certificate for vpn\_server

The commonName value of the created certificate is 'okore\_joel' and it will remain certified for 825 days.

2f. Create a key for the client and also a client certificate. What is the expiration date of the certificate?

To create a key for the client (vpn\_client1), I executed the easyrsa program with the 'gen-req' argument. I specified a client-type key and named the client 'vpn\_client1', as shown in picture 7.



```

(okore_joel@kali) ~$ sudo /etc/openvpn/easy-rsa/.easyrsa gen-req vpn_client1 nopass
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [vpn_client1]:okore_joel

Notice
-----
Private-Key and Public-Certificate-Request files created.
Your files are:
* req: /home/okore_joel/pki/reqs/vpn_client1.req
* key: /home/okore_joel/pki/private/vpn_client1.key

(okore_joel@kali) ~$

```

Picture 7 - Creating key for client (vpn\_client1)

Next, I created a certificate for the OpenVPN client (vpn\_client1) by running the easyrsa program with the 'sign-req' argument. I specified a client-type certificate for 'vpn\_client1', as shown in picture 8.

```

(okore_joel@kali) ~$ sudo /etc/openvpn/easy-rsa/.easyrsa sign-req client vpn_client1
Please check over the details shown below for accuracy. Note that this request
has not been cryptographically verified. Please be sure it came from a trusted
source or that you have verified the request checksum with the sender.
You are about to sign the following certificate:

Requested CN: 'vpn_client1'
Requested type: 'client'
Valid for: '825' days

-----
subject=
commonName = okore_joel

Type the word 'yes' to continue, or any other input to abort.
Confirm request details: yes

Using configuration from /home/okore_joel/pki/ff4b419c2/temp.0.1
Enter pass phrase for /home/okore_joel/pki/private/ca.key:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName :ASN.1 12:'okore_joel'
Certificate is to be certified until Dec 23 13:29:46 2026 GMT (825 days)

Write out database with 1 new entries
Database updated

Notice
-----
Certificate created at:
* /home/okore_joel/pki/issued/vpn_client1.crt

(okore_joel@kali) ~$

```

Picture 8 - Creating certificate for client (vpn\_client1)

The client certificate expires on December 23, 2026, at 13:29:46 GMT. It remains valid for 825 days from the date of creation.

3a. What is the TLS Authentication (TA) Key, and why is it essential in OpenVPN?

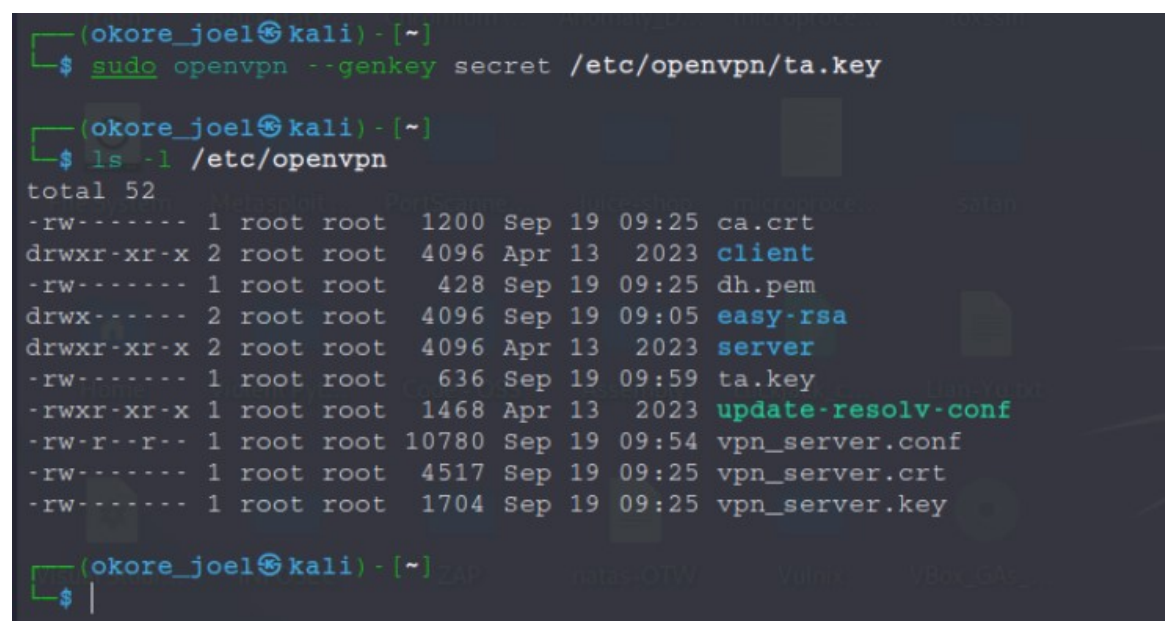
The TLS Authentication (TA) key is a pre-shared key used in OpenVPN to enhance security by adding an extra layer of protection to the TLS handshake process. It is basically a HMAC signature added to all SSL/TLS handshake packets for the purpose of verifying integrity.

The TA key is essential because:

- it prevents unauthenticated clients or attackers from initiating a TLS handshake, thereby reducing the risk of Denial of Service (DoS) attacks by dropping unsigned packets early.
- it protects a UDP port scanning.
- it protects against Buffer overflow vulnerabilities in SSL/TLS implementation.
- it improves authentication process by ensuring that only clients and servers that possess the TA key can communicate

### 3b. Generate TLS Authentication (server) and show the server key information

To generate the TLS Authentication key, I ran the OpenVPN utility with the '-genkey' flag. This generated an OpenVPN static key and wrote it to the specified TA key path (/etc/openvpn/ta.key). See picture 9.



```
(okore_joel@kali) - [~]
$ sudo openvpn --genkey secret /etc/openvpn/ta.key

(okore_joel@kali) - [~]
$ ls -l /etc/openvpn
total 52
-rw-r--r-- 1 root root 1200 Sep 19 09:25 ca.crt
drwxr-xr-x 2 root root 4096 Apr 13 2023 client
-rw-r--r-- 1 root root 428 Sep 19 09:25 dh.pem
drwxr-xr-x 2 root root 4096 Sep 19 09:05 easy-rsa
drwxr-xr-x 2 root root 4096 Apr 13 2023 server
-rw-r--r-- 1 root root 636 Sep 19 09:59 ta.key
-rwxr-xr-x 1 root root 1468 Apr 13 2023 update-resolv-conf
-rw-r--r-- 1 root root 10780 Sep 19 09:54 vpn_server.conf
-rw-r--r-- 1 root root 4517 Sep 19 09:25 vpn_server.crt
-rw-r--r-- 1 root root 1704 Sep 19 09:25 vpn_server.key

(okore_joel@kali) - [~]
$
```

Picture 9 - Generating TLS Authentication Key for OpenVPN

server

The following image shows the contents of the 2048-bit TLS Authentication Key (pic. 10)

```
(okore_joel@kali) - [/etc/openvpn]
$ sudo cat ta.key
#
# 2048 bit OpenVPN static key
#
-----BEGIN OpenVPN Static key V1-----
0cdf3a909b558df26278a3cabe71c339
a46d7e0167fea6e5dd8d48eb858b30f8
0649e51b1bd340db4c133ef2e436dd83
4bd235a4a91caed7d9fb4f3478181798
df4896f1ab01cc8bef792cf06d8a2f0a
039e788a6dd7f1d7feb85e3d02714fb7
69b1def639c2f80c658e2ba62df7037f
8713adb046876744afc7aa5b663d66d6
5d59ba24b3779270a2a695463812c214
228f91696a94a57a9efa94ab1c2bb808
a8040eadfab86a598a16326cbcd5e6da
691c9544bc66d8ae25e7ceaae2edaff2
ec43dc13740bd4570413f34356f8cdf7
a9e39713fbfcb969b366a387a7b0b98b
e8dea5e455ecbf6804f26fff88e88dc1
d66b6838129bb8bcd20fae7ad76ac96
-----END OpenVPN Static key V1-----
(okore_joel@kali) - [/etc/openvpn]
$ |
```

Picture 10 - Information within servers' TA key

3c. Generate a TLS Authentication key (client) and show the client key information.

The TLS Authentication key is a shared secret between the server and client, meaning both use the same TA key. The only difference lies in how it's specified in the configuration files. For the server, the second parameter for 'tls-auth' (after the path to the TA key) is 0, while for clients it's 1. See pictures 11 and 12.

```
# Generate with:
#   openvpn --genkey tls-auth ta.key
#
# The server and each client must have
# a copy of this key.
# The second parameter should be '0'
# on the server and '1' on the clients.
;tls-auth ta.key 0 # This file is secret
```



Picture 11 - Section of server configuration file specifying the path to TA key

```
;http-proxy [proxy server] [proxy port #]

# Wireless networks often produce a lot
# of duplicate packets. Set this flag
# to silence duplicate packet warnings.
;mute-replay-warnings

# SSL/TLS parms.
# See the server config file for more
# description. It's best to use
# a separate .crt/.key file pair
# for each client. A single ca
# file can be used for all clients.
ca ca.crt
cert vpn_client1.crt
key vpn_client1.key

# Verify server certificate by checking that the
# certificate has the correct key usage set.
# This is an important precaution to protect against
# a potential attack discussed here:
# http://openvpn.net/howto.html#mitm
#
# To use this feature, you will need to generate
# your server certificates with the keyUsage set to
# digitalSignature, keyEncipherment
# and the extendedKeyUsage to
# serverAuth
# EasyRSA can do this for you.
remote-cert-tls server

# Allow to connect to really old OpenVPN versions
# without AEAD support (OpenVPN 2.3.x or older)
# This adds AES-256-CBC as fallback cipher and
# keeps the modern ciphers as well.
;data-ciphers AES-256-GCM:AES-128-GCM:?CHACHA20-POLY1305:AES-256-CBC

# If a tls-auth key is used on the server
# then every client must also have the key.
;tls-auth ta.key 1

# Set log file verbosity.
verb 3
-- INSERT --
```

Picture 12 - Section of client configuration file specifying the path to TA key

## TASK 3 - Verification

4a. Simulate a communication between the OpenVPN server and the client.

To simulate communication using OpenVPN, I utilized two Kali Linux virtual machines: one serving as the server (kali@kali) and the other as the client (okore@kali). After completing all previous steps on the kali@kali machine, I needed to transfer the required files to the client (okore@kali). To

accomplish this, I employed Python's http.server library to host an HTTP server on port 9000, as shown in picture 13.

```
(kali㉿kali) - [/etc/openvpn]
$ sudo python3 -m http.server 9000
Serving HTTP on 0.0.0.0 port 9000 (http://0.0.0.0:9000/) ...
192.168.0.17 - - [20/Sep/2024 08:54:30] "GET /ca.crt HTTP/1.1" 200 -
192.168.0.17 - - [20/Sep/2024 08:54:30] "GET /client.conf HTTP/1.1" 200 -
192.168.0.17 - - [20/Sep/2024 08:54:30] "GET /ta.key HTTP/1.1" 200 -
192.168.0.17 - - [20/Sep/2024 08:54:30] "GET /vpn_client1.crt HTTP/1.1" 200 -
192.168.0.17 - - [20/Sep/2024 08:54:30] "GET /vpn_client1.key HTTP/1.1" 200 -
```

Picture 13 - Serving required files to client by using Python's http.server library

To acquire these files from the server, I used the wget Linux utility, a non-interactive network downloader, as shown in picture 14.

```
(ohore@kali) ~$ sudo wget 192.168.0.7:9000/ca.crt 192.168.0.7:9000/client.conf 192.168.0.7:9000/ta.key 192.168.0.7:9000/vpn_client1.crt 192.168.0.7:9000/vpn_client1.key
--2024-09-20 08:54:30-- http://192.168.0.7:9000/ca.crt
Connecting to 192.168.0.7:9000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1200 (1.2K) [application/x-x509-ca-cert]
Saving to: 'ca.crt'

ca.crt                               100%[=====] 1.17K --KB/s in 0s

2024-09-20 08:54:30 (63.6 MB/s) - 'ca.crt' saved [1200/1200]

--2024-09-20 08:54:30-- http://192.168.0.7:9000/client.conf
Connecting to 192.168.0.7:9000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3457 (3.4K) [application/octet-stream]
Saving to: 'client.conf'

client.conf                           100%[=====] 3.38K --KB/s in 0s

2024-09-20 08:54:30 (671 MB/s) - 'client.conf' saved [3457/3457]

--2024-09-20 08:54:30-- http://192.168.0.7:9000/ta.key
Connecting to 192.168.0.7:9000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 636 [application/pgp-keys]
Saving to: 'ta.key'

ta.key                                100%[=====] 636 --KB/s in 0.04s

2024-09-20 08:54:30 (17.1 KB/s) - 'ta.key' saved [636/636]

--2024-09-20 08:54:30-- http://192.168.0.7:9000/vpn_client1.crt
Connecting to 192.168.0.7:9000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4495 (4.4K) [application/x-x509-ca-cert]
Saving to: 'vpn_client1.crt'

vpn_client1.crt                       100%[=====] 4.39K --KB/s in 0s

2024-09-20 08:54:30 (640 MB/s) - 'vpn_client1.crt' saved [4495/4495]

--2024-09-20 08:54:30-- http://192.168.0.7:9000/vpn_client1.key
Connecting to 192.168.0.7:9000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1704 (1.7K) [application/pgp-keys]
Saving to: 'vpn_client1.key'

vpn_client1.key                       100%[=====] 1.66K --KB/s in 0s

2024-09-20 08:54:30 (354 MB/s) - 'vpn_client1.key' saved [1704/1704]

FINISHED --2024-09-20 08:54:30--
Total wall clock time: 0.05s
Downloaded: 5 files, 11K in 0.04s (309 KB/s)
```

Picture 14 - Using wget to acquire necessary OpenVPN client files from server

I proceeded to start the OpenVPN server using the systemctl command. To verify the server is running, I checked for the presence of the tun0 interface with the IP address 10.8.0.1 among the network interfaces by executing the 'ip addr' command. Refer to Picture 15.

```
(kali@kali) - [/etc/openvpn]
$ sudo systemctl start openvpn@vpn_server

(kali@kali) - [/etc/openvpn]
$ ip addr show dev tun0
319: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 10.8.0.1/24 scope global tun0
        valid_lft forever preferred_lft forever
    inet6 fe80::b97a:d10b:edaa:251d/64 scope link stable-privacy
        valid_lft forever preferred_lft forever

(kali@kali) - [/etc/openvpn]
$
```

Picture 15 - Starting OpenVPN server on kali@kali

I performed the same steps on the OpenVPN client machine (okore@kali) using the systemctl command. The presence of the tun0 interface with the IP address 10.8.0.2 confirms that the client is running successfully. See Picture 16.

```
(okore@kali) - [~]
$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  ca.crt  client.conf  hydra.restore  ta.key  vpn_client1.crt  vpn_client1.key

(okore@kali) - [~]
$ sudo mv ca.crt client.conf ta.key vpn_client1.crt vpn_client1.key /etc/openvpn

(okore@kali) - [~]
$ sudo systemctl start openvpn@client

(okore@kali) - [~]
$ ip addr show dev tun0
3: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 10.8.0.2/24 scope global tun0
        valid_lft forever preferred_lft forever
    inet6 fe80::ed97:c697:6573:e8cf/64 scope link stable-privacy proto kernel_l1
        valid_lft forever preferred_lft forever
```

Picture 16 - Starting OpenVPN client on okore@kali

I set up a message-passing channel between the client and server over the VPN tunnel using Netcat, with communication taking place on port 7777. Refer to Pictures 17 and 18 for details.

```
(kali@kali) - [/etc/openvpn]
$ sudo nc -nv 10.8.0.2 7777
(UNKNOWN) [10.8.0.2] 7777 (?) open
Hello from server
Hello from client
```

Picture 17 - Using Netcat for message passing on server machine (kali@kali)

```

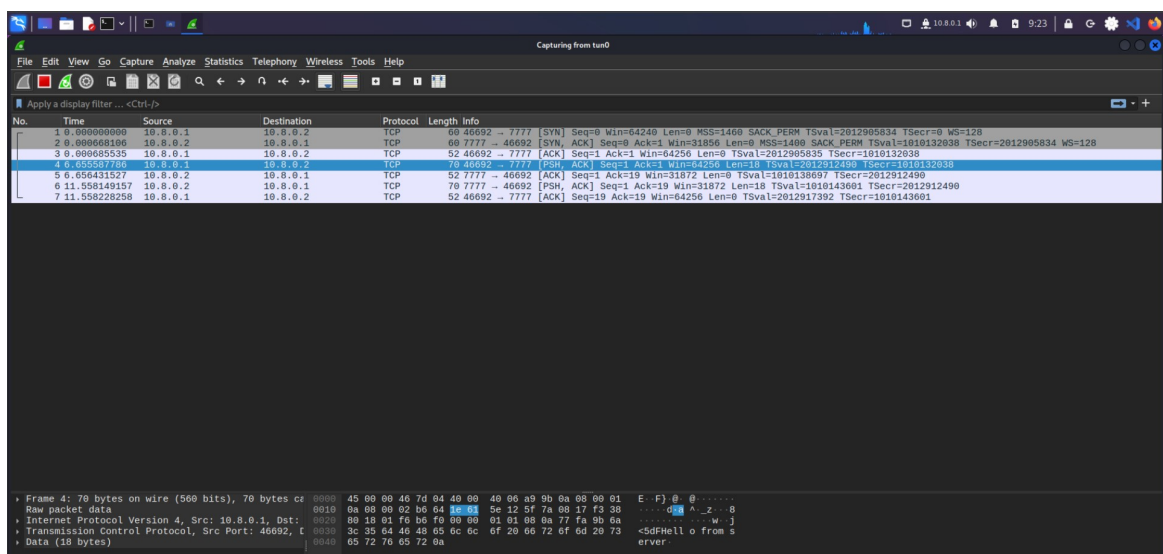
(okore@kali)-[~]
$ sudo nc -lvnp 7777
listening on [any] 7777 ...
connect to [10.8.0.2] from (UNKNOWN) [10.8.0.1] 46692
Hello from server
Hello from client
|

```

Picture 18 - Using Netcat for message passing on client machine (okore@kali)

4b. Using a traffic sniffer like Wireshark, inspect the interface of the VPN traffic. Show the information about this traffic.

To validate the message exchange between the client and server through the VPN tunnel, I captured the network traffic using Wireshark. As depicted in Picture 19, the packet capture provides a detailed breakdown of the TCP communication over port 7777. The fourth packet in the sequence clearly shows the server's message ('Hello from server') being transmitted to the client. This confirms the successful operation of the message-passing channel established via Netcat between the two machines, with the data securely routed through the VPN tunnel



Picture 19 - Wireshark Packet capture of messages exchange through the VPN tunnel

## **REFERENCES**

1. OpenVPN in Ubuntu: <https://ubuntu.com/server/docs/how-to-install-and-use-openvpn>
2. Keyfactor: <https://www.keyfactor.com/education-center/what-is-pki/>
3. OpenVPN Community Resources: <https://openvpn.net/community-resources/hardening-openvpn-security/>