Practical Assignment 1

2023/2024

Master in Informatics Engineering
Information Tecnology Security

Bruno Sequeira 2020235721, brunosequeira@student.dei.uc.pt
Rui Santos 2020225542, rpsantos@student.dei.uc.pt

March 16, 2024

Contents

1 Introduction

The goal of this project is to configure and implement a secure communication system using VPN, two-factor authentication, and PKI. The system consists of a remote client (road warrior), a VPN gateway, an Apache web server, and a private certification authority. The system supports mutual authentication using X.509 certificates, one-time passwords, and OCSP verification. The system also allows the creation, issuance, and revocation of certificates using OpenSSL.

2 Architecture

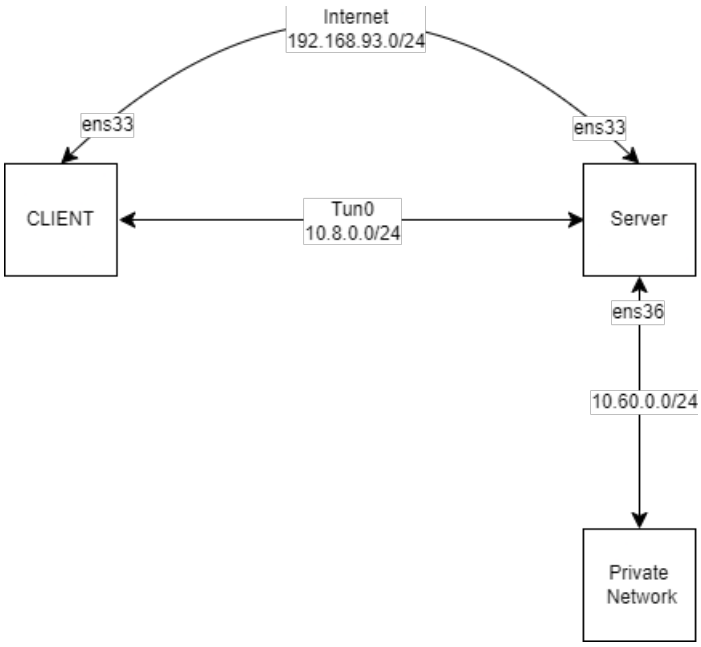

Figure 1: Network Architecture

To do this assignment, he needed two computers, one for the client and one for the server. Both have connections to the internet and we also created a private network only for them. The IP addresses for the Internet connection were given by DHCP, the ones for the private network were created manually. The IP addresses are:

- Server (Internet): 192.168.93.128
- Server (Private): 10.60.0.1
- Server (VPN): 10.8.0.1
- Client (Internet): 192.168.93.129
- Client (VPN): 10.8.0.6

3 Software

To complete the assignment we used two virtual machines, both CentOS Stream 9. In each virtual machine we needed:

- On server and client:
  - epel-release for package signing

  - openvpn to create VPN tunnels allowing a secure connection between the client and the server
  - wireshark to test the connections
  - net-tools to test the connections
- On server:
  - httpd that will be our server
  - mod_ssl to provide strong cryptography for apache
  - openssl to create the certificates and the OCSP server
  - libpam-google-authenticator to implement 2FA authentication

4 Configurations

4.1 OpenVPN

We started to configure our vpn server and for that we needed to create the certificates. Every certificate used in this project was created on the server machine.

4.1.1 Certificates

First of all, we created the certification for the CA (Certification Authority), that is going to be self-signed. But first we created the index.txt file that will contain the information about the certificates created by the CA and also the serial file will have the number of the certificate.

```
cd /etc/pki/CA
touch index.txt
echo 01 > serial

# creation of the private key for the CA
openssl genrsa -out ca.key -des3
# creation of a CSR for the CA
openssl req -new -key ca.key -out ca.csr

C = PT, ST = Coimbra, L = Coimbra, O = DEI, OU = DEI, CN = CA_STI

# creation of a "self-signed" certificate to represent the CA
openssl x509 -req -days 365 -in ca.csr -out ca.crt -signkey ca.key
```

After that, we created the certificates for the vpn client and vpn server:

```
# Server certificate
# creation of the private key for the server
openssl genrsa -out server.key {des3
# creation of a CSR for the server certificate
openssl req -new -key server.key -out server.csr

C = PT, ST = Coimbra, O = DEI, OU = DEI, CN = VPN_SERVER

# creation of the certificate for the server using our private CA
openssl ca -in server.csr -cert ca.crt -keyfile ca.key -out server.crt
```

And then we created the certificates for the client. The steps are the same for the certificates for the server, the only change is the common name which we set to VPN_CLIENT

4.1.2 Configuration files

Once we completed the creation of the certificates, it was time to configure the openvpn server. For that, we use a configuration file sample that we copied to the openvpn server folder

```
cp /usr/share/doc/openvpn/sample/sample-config-files/server.conf /etc/openvpn/server
```

Before we edit the configuration file, we need to generate the file with the Diffie-Hellman algorithm and the file with the secret key file. This secret key file is created in the server, but must be copied to the client. This file adds an additional HMAC signature to all SSL/TLS handshake packets for integrity verification, so it is another layer of security.

```
openvpn --genkey tls-auth ta.key
openssl dhparam -out dh2048.pem 2048
```

Now we can proceed to the configuration file:

```
local 192.168.93.128
proto udp
dev tun
ca /etc/pki/CA/ca.crt
cert /etc/pki/CA/openvpnserver/server.crt
key /etc/pki/CA/openvpnserver/server.key
dh dh2048.pem
server 10.8.0.0 255.255.255.0
ifconfig-pool-persist ipp.txt
push "redirect-gateway def1 local"
keepalive 10 120
tls-auth ta.key 0
cipher AES-256-CBC
persist-key
persist-tun
status openvpn-status.log
verb 3
explicit-exit-notify 1
```

We first indicate the IP and port where the server will be to accept new connections and the protocol used to communicate. Then we will select the type of tunnel to create, in this case dev tun that means that we will create a routed IP tunnel. After that we specify the paths to the CA certificate and the openvpn server private key and certificate and the Diffie-Hellman file. Then we select the range of IP addresses that the VPN clients can use, in this case 10.8.0.0/24. The command *push "redirect-gateway def1 local"* is to force all the trafic to go throw the VPN.

```
└─# traceroute -n www.google.com
traceroute to www.google.com (142.250.200.100), 30 hops max, 60 byte packets
 1  192.168.93.2  0.170 ms  0.164 ms  0.125 ms
```

Figure 2: Traceroute without push redirect

```
└─# traceroute -n www.google.com
traceroute to www.google.com (142.250.178.164), 30 hops max, 60 byte packets
 1  10.8.0.1  6.152 ms  6.220 ms  6.167 ms
 2  192.168.93.2  6.158 ms  6.146 ms  6.358 ms^C
```

Figure 3: Traceroute with push redirect

Traceroute is a network diagnostic tool used to track in real-time the pathway taken by a packet on an IP network from source to destination, reporting the IP addresses of all the devices it pinged in

between. Using this command in the client already connected to the VPN to trace the path between the client and the site www.google.com , as we can see in the pictures above, with the command mention before the traffic goes throw the VPN, going first to 10.8.0.1 (VPN Server) and then to the router (192.168.93.2) and without the command we go directly to the router.

After the push redirect command we have the secret key file ta.key that has the number 0 (required in the server config).

After configure the server we can configure the client:

```
client
–persist-remote-ip
dev tun
proto udp
remote 192.168.93.128 1194
resolv-retry infinite
nobind
persist-key
persist-tun
ca ca.crt
cert client.crt
key client.key
tls-auth ta.key 1
verb 3
```

As I mentioned before, the client have the same ta.key file that we create for the server, but now it must have the number 1. We specify the IP and port of the server in the remote and the path to the CA certificate and the vpn client private key and certificate.

After this the first configuration of the OpenVPN client and server are done (configuration without 2FA and OCSP).

4.2 Apache

4.2.1 Certificates

The process to create the certificates for the apache server is similar to the openvpn. The difference is that as some browsers search for a "subjectAtName" X.509 extension in the server's certificate, we need to create a v3.ext file and include that in the creation of apache certificate:

```
cd /etc/pki/CA
nano v3.ext
```

Figure 4: v3.ext file

After that we can create the certificates using the same CA created for the openvpn server. Note that when creating the csr file, the CN (common name) is the name of the server (later we will explain how to set that URL, but it will be www.sti.pt)

```
# Apache certificate
# creation of the private key for the apache
openssl genrsa -out apache.key -des3
# creation of a CSR for the apache certificate
openssl req -new -key apache.key -out apache.csr
```

```
C = PT, ST = Coimbra, O = DEI, OU = DEI, CN = www.sti.pt

# creation of the certificate for the apache using our private CA
openssl ca -in apache.csr -cert ca.crt -keyfile ca.key -out apache.crt -extfile v3.ext
```

Now, in order to our client access our website, we need to create a client certificate as well.

```
# User certificate
# creation of the private key for the User
openssl genrsa -out user.key -des3
# creation of a CSR for the User certificate
openssl req -new -key user.key -out user.csr

C = PT, ST = Coimbra, O = DEI, OU = DEI, CN = name

# creation of the certificate for the User using our private CA
openssl ca -in user.csr -cert ca.crt -keyfile ca.key -out user.crt
# conversion of the user certificate and key to PKCS#12 format
openssl pkcs12 -export -clcerts -in user.crt -inkey user.key -out user.p12
```

4.2.2 Configuration files

In order to tell the Apache server to ask for a valid certificate, we need to uncomment these lines in the ssl.conf file:
nano /etc/httpd/conf.d/ssl.conf

SSLVerifyClient require
SSLVerifyDepth 10

To conclude the configuration for Apache, we need to add the Apache server name to the hosts file on the Server.

```
nano /etc/hosts
          10.60.0.1 www.sti.pt
```

Now we are done with the more basic configurations of OpenVPN and Apache (without OSCP and 2FA).

4.3 OCSP

OSCP is a protocol used to check the revocation status of digital certificates that came as an alternative to CRL (certificate revogation list). In order to use OSCP in both Apache and OpenVPN we need to make a few changes starting with the openssl configuration file:

```
nano /etc/pki/tls/openssl.cnf
        [usr_cert]
        authorityInfoAccess = OCSP;URI:http://www.stiocsp.pt:81
```

In this case, www.stiocsp.pt will be our ocsp server, listening in port 81.
We now add our ocsp server name in the hosts file:

```
nano /etc/hosts
          10.60.0.1 www.ocspsti.pt
```

Now, in the Apache configuration, we need to add the OCSP suport:

```
nano /etc/httpd/conf.d/ssl.conf
        SSLOCSPEnable on
```

SSLOCSPDefaultResponder "http://www.stiocsp.pt:81"
SSLOCSPOverrideResponder off

SSLOCSDefaultResponder is the URI (Uniform Resource Identifier) with the address and port of the OCSP server. SSLOCSPOverrideResponder is the option to force the use of the OCSP server even if the client certificate has an address of the service. The SSLOCSPEnable is the option to use the OCSP in checking the validity of client certificates.

We need to make a few changes in the OpenVPN server to make it use OCSP. We are going to use a script already provided by openvpn and make a few changes to make it work with our OCSP server:

```
cp /usr/share/doc/openvpn/contrib/OCSP_check/OCSP_check.sh /etc/openvpn/server
nano /etc/openvn/server/OCSP_check.sh
        ocsp_url="http://www.stiocsp.pt:81"
        issuer="/etc/pki/CA/ca.crt"
        verify="/etc/pki/CA/ca.crt"
```

And then we need to change the OpenVPN server configuration:

```
nano /etc/openvn/server/server.conf
        # Add the lines
        script-security 2
        tls-verify /etc/openvpn/server/OCSP_check.sh
```

And after that we need to give permission to use the OCSP_check.sh file:

```
chmod 777 /etc/openvpn/server/OCSP_check.sh
```

Now, he have to create again all certificates using the same commands we used in 4.1.1 and 4.2.1, in order to include the OCSP information in all certificates.

4.4 Two-Factor Authentication with Google

Both VPN establishment and HTTPS accesses are going to make use of one-time passwords that will be generated, in this case, by the Google Authenticator APP. This is another step to increase the security of our system.

4.4.1 2FA for Apache

To implement this, we need to download the source code of the module google-authenti
cator-apache-module from google and compile it:

```
# Download the source code
wget https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com
/google-authenticator-apache-module/GoogleAuthApache.src.r10.bz2

# decompress the file
tar xvf GoogleAuthApache.src.r10.bz2

cd google-authenticator-apache-module
# compile
make install
```

After this we need to change the Apache configuration file and add the configuration for 2FA and create a new directory for the 2FA users:

```
# 2FA users directory
cd /etc/httpd
mkdir google_auth_users
```

```
nano /etc/httpd/conf/httpd.conf
# apache configuration
        # at the end of the file
        LoadModule authn_google_module modules/mod_authn_google.so

        ¡Directory "var/www/html"¿
         Options FollowSymLinks ExecCGI
         AllowOverride All
         Order deny,allow
         Allow from all
         AuthType Basic
         AuthName "google auth"
         AuthBasicProvider "google_authenticator"
         Require valid-user
         GoogleAuthUserPath google_auth_users
         GoogleAuthCookieLife 3600
         GoogleAuthEntryWindow 2
        ¡/Directory¿
```

4.4.2 2FA for OpenVPN

In order to have 2FA working in OpenVPN we need to change the server configuration.

```
nano /etc/openvpn/server/server.conf
        # add at the end
        plugin /usr/lib64/openvpn/plugins/openvpn-plugin-auth-pam.so openvpn

nano /etc/pam.d/openvpn
        auth [user_unknown=ignore success=ok ignore=ignore default=bad] pam_securetty.so
        auth required /lib64/security/pam_google_authenticator.so secret=/etc/openvpn/
        google-authenticator/$USER user=gauth forward_pass
        auth include system-auth
        account include system-auth
        password include system-auth
```

Now we can add new users to use Google Authenticator:

```
# Add the user to run google-authenticator as and set the correct permissions
useradd gauth
mkdir /etc/openvpn/google-authenticator
chown gauth:gauth /etc/openvpn/google-authenticator
&& chmod 700 /etc/openvpn/google-authenticator

# create this script to make user creation easier
nano /root/create-gauth.sh
        #!/bin/sh
        # Parse arguments
        USERNAME="$1"
        if [ -z "$USERNAME" ]; then
            echo "Usage: $(basename $0) ¡username¿"
            exit 2
        fi
        # Set the label the user will see when importing the token:
        LABEL='OpenVPN Server'
        su -c "google-authenticator -t -d -r3 -R30 -W -f -l "${LABEL}" -s /etc/openvpn/google
-authenticator/${USERNAME}" - gauth
```

make it executable
chmod 700 /root/create-gauth.sh

create a new user
useradd -s /sbin/nologin sti
passwd sti
/root/create-gauth.sh sti

Running this program you should get an url that is a page with a qrcode that you should scan with your google authenticator app and then write the code on the console, and the user is created.

Now we copied the user to the apache user directory to have the same user for openvpn and apache:

cp /etc/openvpn/google-authenticator/sti /etc/httpd/google_auth_users/

And now the configuration should be completed.

5 Demo and Testing

In order to test our project we created two client certificates for apache and OpenVPN clients, one good and the other revoked. To revoke a certificate we used the following command:

openssl ca - revoke clientcert.crt - keyfile ca.key - cert ca.crt

In our case we created the certificates clientvpnbad (CN: VPN_CLIENT_BAD), client (CN: VPN_CLIENT), userbom (CN: APACHE_CLIENT_GOOD_2VM) e usermau (CN: APACHE_CLIENT_BAD_2VM)

Next, we started the OCSP responder and use OCSP to see if our certificates are actually revoked:

openssl ocsp -index index.txt -port 81 -rsigner ca.crt -rkey ca.key -CA ca.crt -text -out log.txt

openssl ocsp -CAfile ca.crt -issuer ca.crt -cert openvpnclient/clientvpnbad.crt -url http://www.stiocsp.pt:81 -resp_text

Figure 5: Client certificate for OpenVPN revoked

Figure 6: Client certificate for apache revoked

After this we will test our VPN connection starting by using the revoked client certificate.

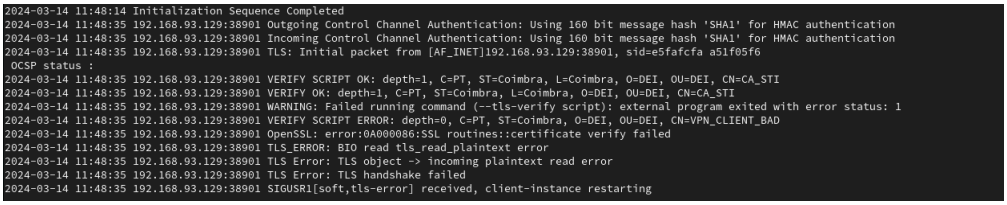

Figure 7: Vpn connection with revoked client certificate - server response

Figure 8: Vpn connection with revoked client certificate - client response

As we can see, the OCSP did not let the client connect as the certificate was revoked. Using the good certificate the client can connect successfully to the OpenVPN server and now have a tun0 interface for the vpn tunnel:

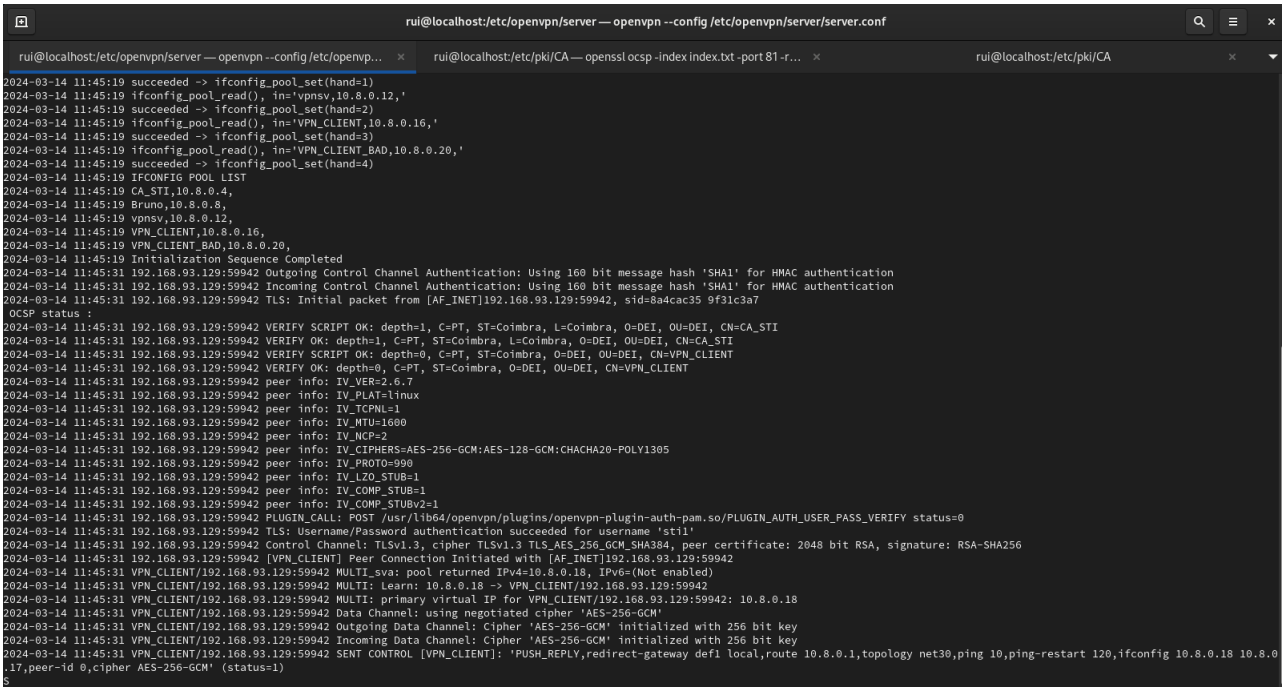

Figure 9: Vpn connection with good client certificate - server response

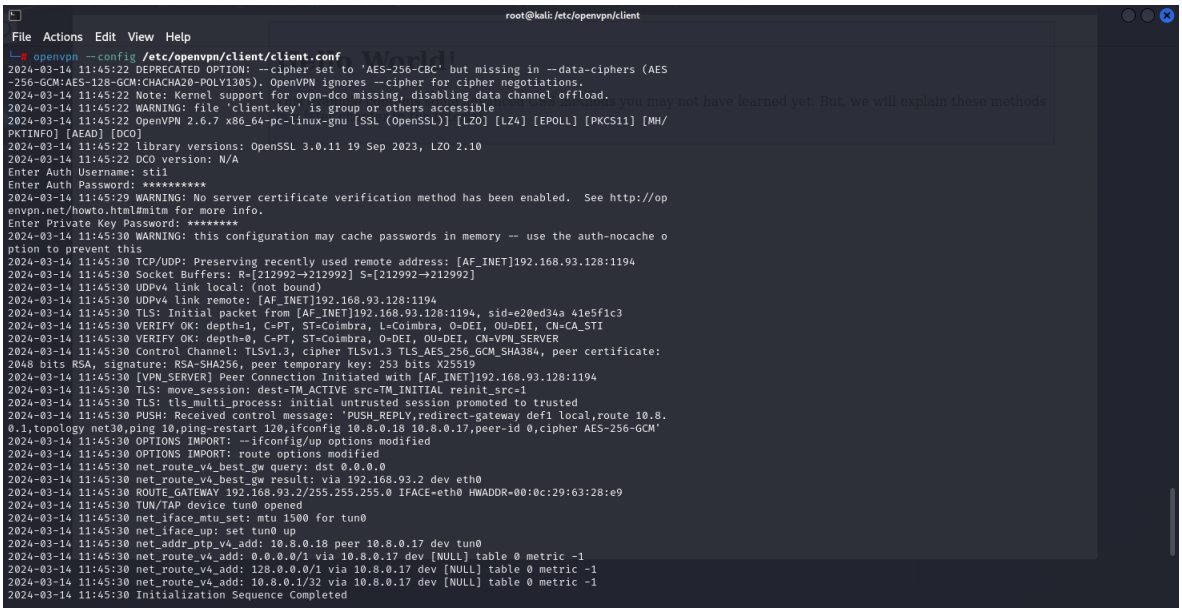

Figure 10: Vpn connection with good client certificate - client response

Figure 11: interface config for client

We can now use wireshark to test the connection and see if we have an HMAC signature to SSL/TLS handshake packets and see what happens when we ping the interface tun0 of the OpenVPN server.

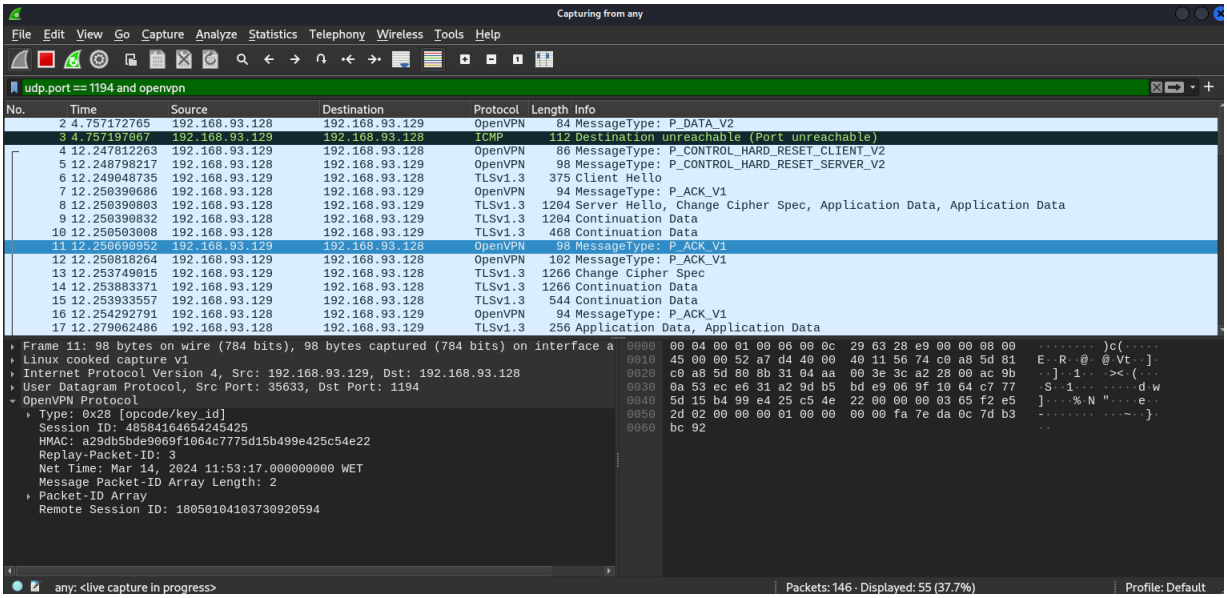

Figure 12: HMAC signature

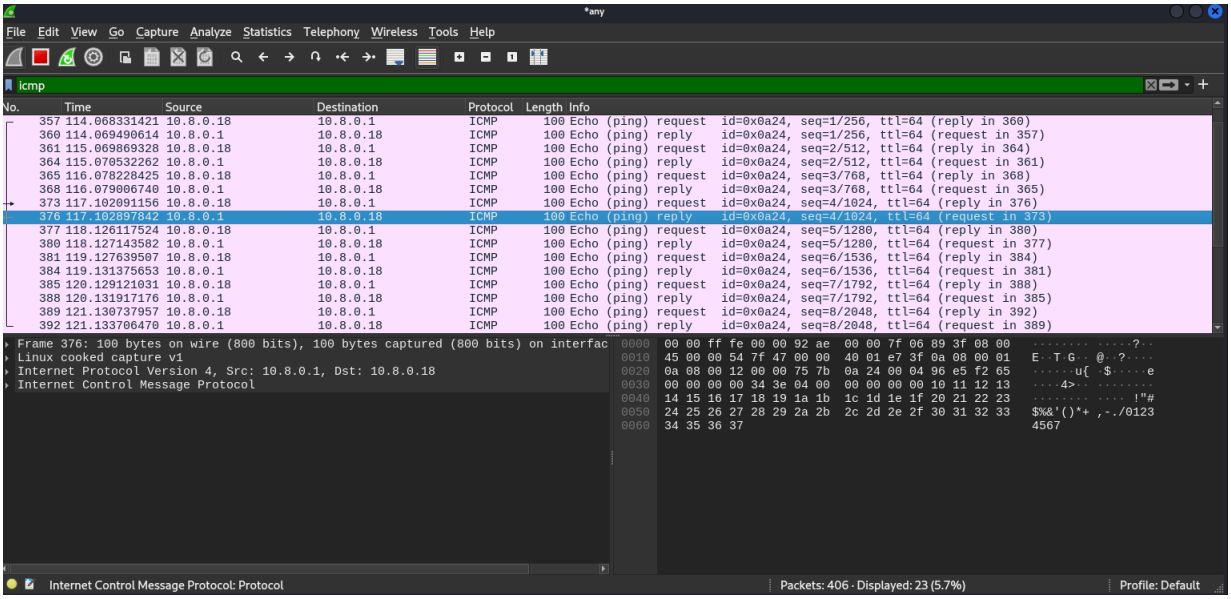

Figure 13: wireshark while ping to OpenVPN server

As we can see the server successfully replies to the client responses via vpn tunnel and we have the HMAC signature present as we wanted.

So now we can move on to the Apache connection. The way we did it, the client can only connect to the Apache server if connected to the VPN as the server is using the private network of the server. If we connect to our website without using the VPN we connect to a site that has the same address that we defined for ours. As we mentioned earlier, our URL is www.sti.pt

Figure 14: Connection to Apache server without VPN

Like we did with the VPN, we are going to test first with a revoked client certificate. The client

from now on will allways be connected to the VPN:

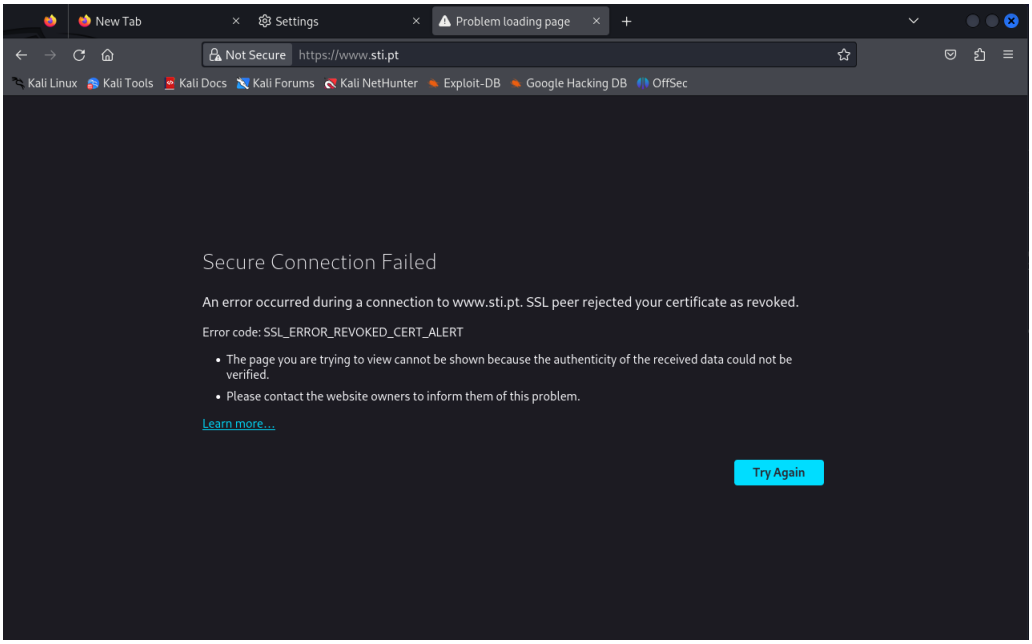

Figure 15: Connection to Apache server with revoked client certificate

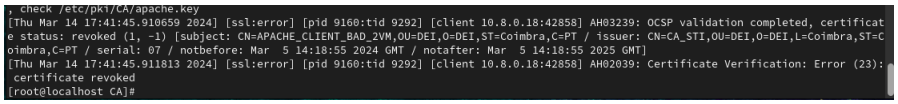

Figure 16: SSL error from httpd log

As we can see from the pictures above, the OCSP did not let the client connect to the Apache server with a revoked certificate. Now we are going to use a good certificate and everything should be good.

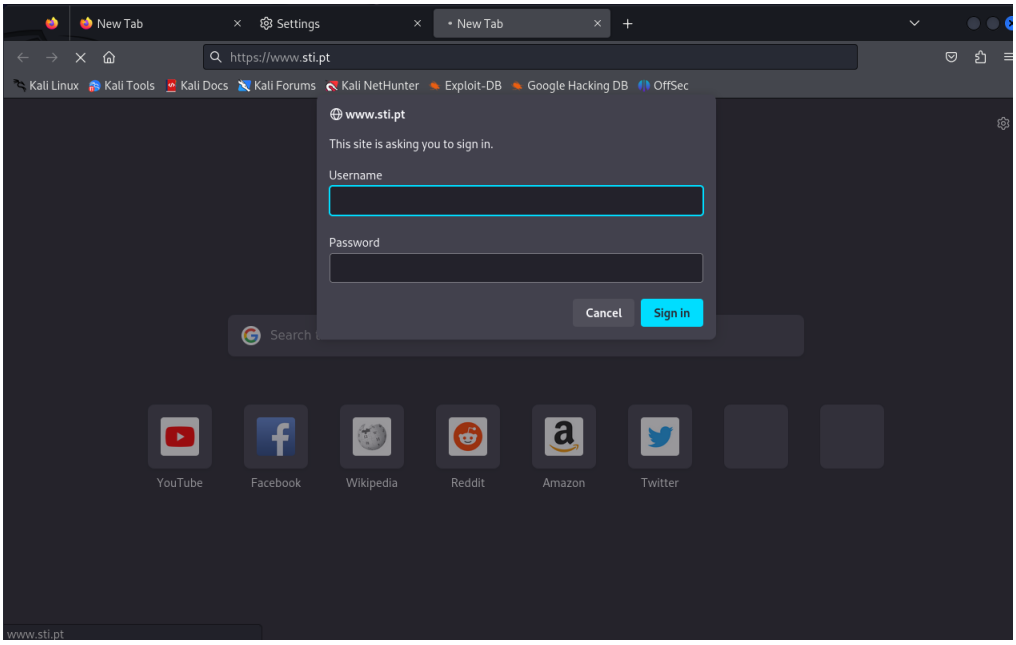

Figure 17: Asking for 2FA credentials

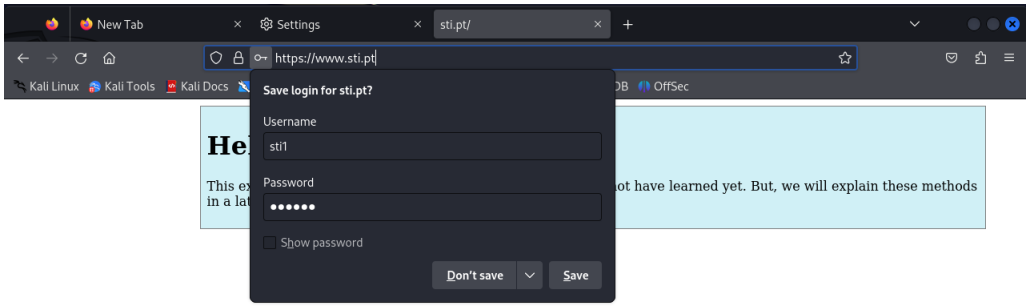

Figure 18: Successfully connection to the server

From the last two pictures we can see that we are asked to input the credentials to the 2FA authentication that we configured. Given those credentials, we were successfully connected to the server.

6 Conclusion

In conclusion, this practical assignment provided valuable hands-on experience in configuring and implementing a secure communication system using various technologies such as VPN, two-factor authentication (2FA), and Public Key Infrastructure (PKI). Through this project, we gained a deeper

understanding of network security principles and practical skills in setting up secure communication channels.

The use of OpenVPN allowed us to establish secure VPN tunnels between the client and server. Implementing two-factor authentication using the Google Authenticator added an extra layer of security by requiring users to provide a one-time password along with their credentials.

The creation and management of X.509 certificates using OpenSSL were integral to the PKI aspect of the project. We learned how to issue, revoke, and verify certificates, enabling mutual authentication between the client and server. Additionally, setting up an OCSP server enhanced the certificate validation process, ensuring the validity of certificates in real-time.

Overall, this assignment increased our knowledge of secure communication protocols, certificate management, and security.

References

[1] Information Tecnology Security Slides

[2] Jorge Granjal, Segurança Prática em Sistemas e Redes com Linux, FCA, 2017

[3] OpenVPN github, Retrieved March 7, 2024, from https://github.com/OpenVPN/openvpn/blob/master/contrib/OCSP_check/OCSP_check.sh

[4] Getting Google Authenticator and Apache to Cooperate on CentOS Retrieved March 8, 2024, from https://www.blogbyben.com/2012/02/getting-google-authenticator-and-apache.html

[5] Setup an OpenVPN server with certificate and two-factor authentication on CentOS 7 Retrieved March 8, 2023, from https://nethack.ch/2016/12/08/setup-an-openvpn-server-with-certificate-and-two-factor-authen
tication-on-centos-7/

[6] What is Traceroute & What is it For. Retrieved March 9, 2024, from https://www.thousandeyes.com/learning/glossary/traceroute