ING4 SI01

Network security

Authentication

Lab on PKI / HTTPS / PROXY

Author(s):
Gabriel PADIS
Anastasia DUCHESNE

Teacher(s):
Mr. Fayad

Paris - Semester 7 - March 5, 2019

Contents

1 Configuration

We have the following configuration :

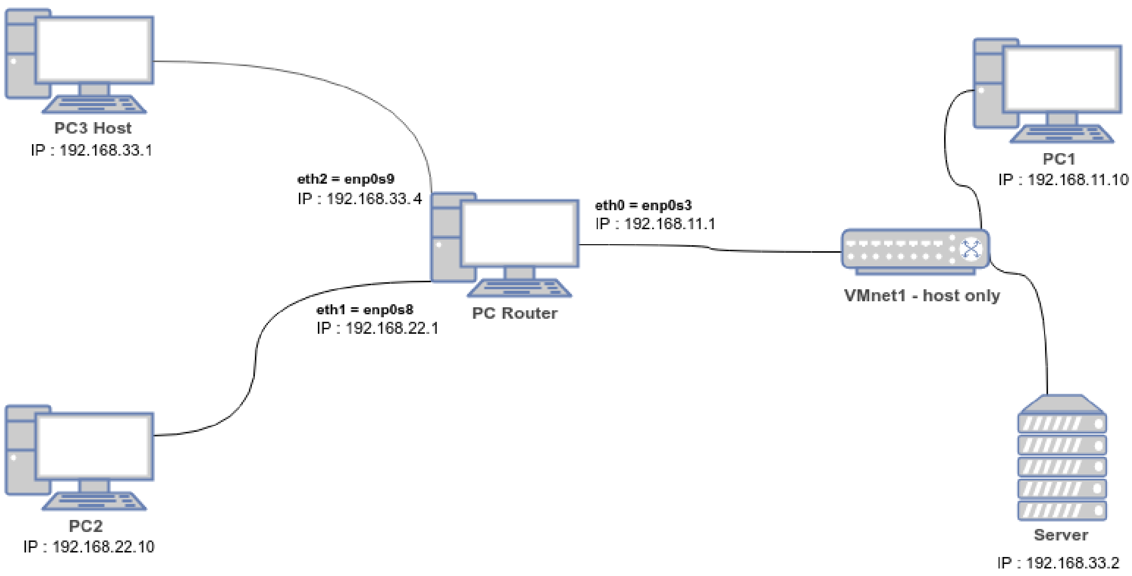


Figure 1: Network configuration

1.1 NAT configuration on PC Router

First, we create a NAT network, so we'll be able later to assign an IP address on the subnet 192.168.33.0/24 to our interface eth2 (enp0s9).

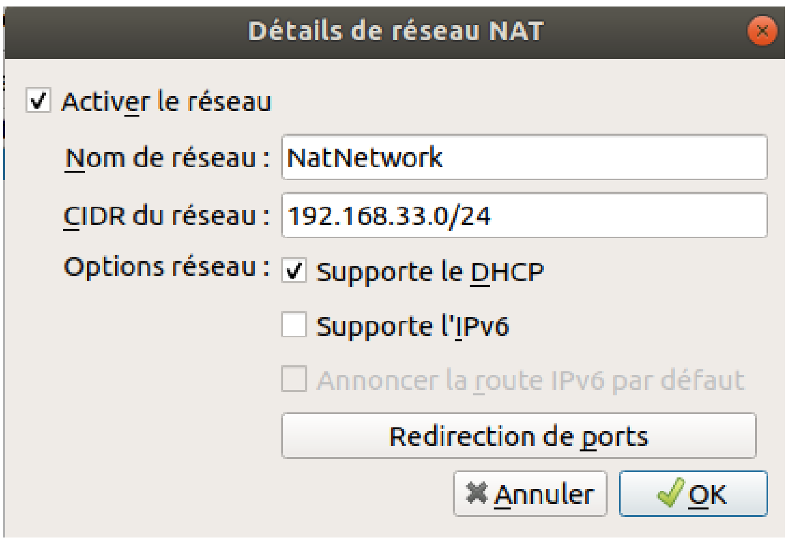


Figure 2: NAT network creation

Then, before turning on PC Router, we assign the right networks to the right adapters/interfaces. We have the following configuration :

- Adapter 1 (eth0) : Host-Only network, 192.168.11.0/24
- Adapter 2 (eth1) : Host-Only network, 192.168.22.0/24
- Adapter 3 (eth2) : NAT network, 192.168.33.0/24

On PC Router, we make changes in the file /etc/network/interfaces in order to have the following configuration for our interfaces :

```
#auto enp0s3
allow-hotplug enp0s3
iface enp0s3 inet static
address 192.168.11.1
netmask 255.255.255.0
up route add -net 192.168.22.0 netmask 255.255.255.0 gw 192.168.22.1

#auto enp0s8
allow-hotplug enp0s8
iface enp0s8 inet static
address 192.168.22.1
netmask 255.255.255.0
up route add -net 192.168.11.0 netmask 255.255.255.0 gw 192.168.11.1

auto enp0s9
allow-hotplug enp0s9
iface enp0s9 inet dhcp
```

Figure 3: Interfaces configuration on PC Router

Only enp0s9 (eth2) has to be on DHCP. When we up our interfaces, we see that the DHCP allocates an IP address on the subnet 192.168.33.0/24 to the interface enp0s9.

```
root@debian:/home/debian# ifup enp0s9
Internet Systems Consortium DHCP Client 4.3.5
Copyright 2004-2016 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/enp0s9/08:00:27:6d:06:4c
Sending on   LPF/enp0s9/08:00:27:6d:06:4c
Sending on   Socket/fallback
DHCPDISCOVER on enp0s9 to 255.255.255.255 port 67 interval 3
DHCPDISCOVER on enp0s9 to 255.255.255.255 port 67 interval 3
DHCPREQUEST of 192.168.33.4 on enp0s9 to 255.255.255.255 port 67
DHCPOFFER of 192.168.33.4 from 192.168.33.3
DHCPACK of 192.168.33.4 from 192.168.33.3
bound to 192.168.33.4 -- renewal in 534 seconds.
```

Figure 4: DHCP Request for enp0s9

We now have internet on our PC Router :

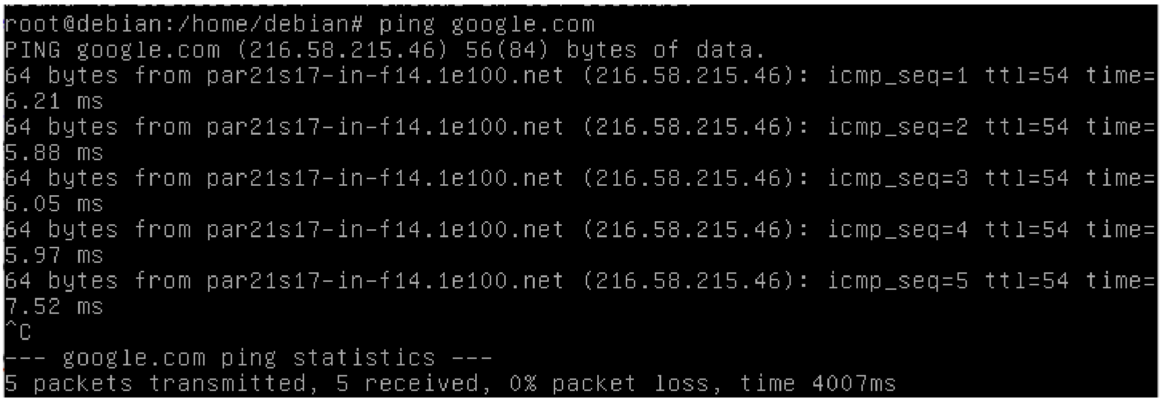


Figure 5: Pinging google.com

1.2 Configuring internet access on the other machines

Now that we have internet on the router, we're willing to have internet on our other machines. We want to do a source NAT. To do so, we're going to tell the firewall that our router is acting like a gateway. We're adding a new rule :

iptables -t nat -A POSTROUTING -o enp0s9 –j MASQUERADE

- **-t**: specifies the table that matches the packet. We use it with the table **nat**, since our goal is to transmit internet to the other machines. nat is the table that is consulted for the packets creating a new connection.
- **-A or –append** : appends one or more rules at the end of **POSTROUTING**, the chain we selected. When we do a source NAT, the change of the source addresses of the connections is done just before the packet goes out, that's why we're using POSTROUTING. Using POSTROUTING also means all the other functions will see our packet unmodified. POSTROUTING can only change the source address.
- **-o or –out-interface** : name of the interface through which the packet entering the POSTROUTING chain is going to be sent. In our case it is **enp0s9** since it's the one configured with a NAT network.
- **-j** : target specification. We use **MASQUERADE** because we have dynamically assigned IP connections, since we're using DHCP connections to connect to the network. It identifies an IP address for each packet that goes out and verifies the exit interface for each connection.

Finally, we intall **iptables-persistent**, and we save the NAT rule with **iptables-save** > **/etc/iptables/rules.v4** so it will be persistent in the firewall. This means that the rule will load each time we'll turn on our virtual machine, either way it would have been erased from the system.

```
root@debian:/home/debian# cat /etc/iptables/rules.v4
# Generated by iptables-save v1.6.0 on Fri Feb 15 10:54:12 2019
*nat
:PREROUTING ACCEPT [16:1248]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
-A POSTROUTING -o enp0s9 -j MASQUERADE
COMMIT
# Completed on Fri Feb 15 10:54:12 2019
# Generated by iptables-save v1.6.0 on Fri Feb 15 10:54:12 2019
*filter
:INPUT ACCEPT [1:76]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [1:76]
COMMIT
```

Figure 6: iptables persistent rules

We assigned a static IP address and used PC Router as a default gateway for all our machines.

VM	IP address
PC2	192.168.22.10
PC1	192.168.11.10
Server	192.168.11.11

For example, for our server in /etc/network/interfaces we have :

```
auto enp0s3
iface enp0s3 inet static
address 192.168.11.11
netmask 255.255.255.0
gateway 192.168.11.1
```

Figure 7: Interface configuration of server

Finally, we added 8.8.8.8 as a name server on PC1, PC2 and server.
When we try to ping google.com from our server, it works !

```
root@debian:/home/debian# ping google.com
PING google.com (216.58.215.46) 56(84) bytes of data.
64 bytes from par21s17-in-f14.1e100.net (216.58.215.46): icmp_seq=1 ttl=53 time=
7.46 ms
64 bytes from par21s17-in-f14.1e100.net (216.58.215.46): icmp_seq=2 ttl=53 time=
6.34 ms
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 8036ms
rtt min/avg/max/mdev = 6.344/6.903/7.462/0.559 ms
```

Figure 8: Pinging google from server

2 Part 1 : Certificates

2.1 CA ROOT

2.1.1 Generating a private key RSA

We generate a priate key RSA named **privcaroot.key**, in the private directory with the following criterias :

- lenght of 2048 bits
- password encrypted with the des3 algorithm

We used "*password*" as the **password** of our key.

Figure 9: Generating private key RSA

2.1.2 Creating a self-signed certificate

We're creating a root certificate authority. Its certificates are the top-most certificates of the tree of the certificates. The **root certificates** are self-signed with x509-based public key infrastructure. x509 certificates are used in internet protocols for example. Later, the private key of the root certificate will be used to sign other certificates.

We use the private key we generated in the question before in order to generate a self-signed certificate with the following command line :

openssl req -new -x509 -days 365 -key private/privcaroot.key -out certs/certcaroot.crt -config ./openssl.cnf -extensions CA_ROOT

We just generated a certificate with the following configuration :

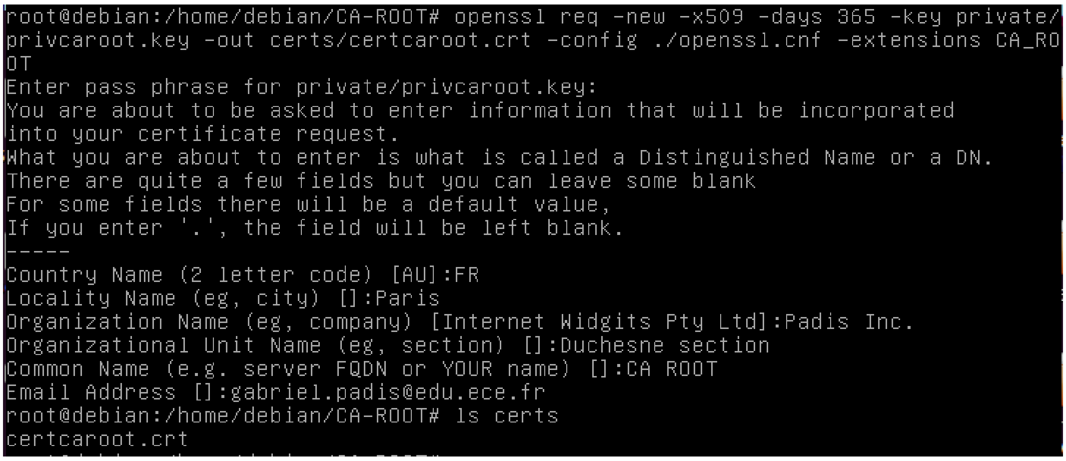
- Available for 365 days
- signed with the private key privcaroot.key
- uses the configuration from openssl.cnf file.
- uses the specified field for extensions for the certificate from the openssl.cnf file. Some of the extensions that are used are :

– **Basic Constraints** : specifies that it is a **certification authority (CA)** certificate. **Path length** gives the number of non-self issued or signed intermediate certificates that can be beneath our certificate in the certification tree. In our case, there can be only another one CA certificate in the path. We mark it as "**critical**" for a more secure SSL negotiation : if the certificate isn't issued by a CA, then the SSL negotiation will fail.

– **Subject key identifier** : creates a random identifier that is unique to the given public key. It is a hash value of the SSL certificate and permits to identify the certificates containing the given public key.

– **Authority key identifier :** identifies the public key corresponding to the private key with which the certificate was signed.

– **Key usage :** defines for what usage the certificate is made for. KeyCertSign enables to use the public key to verify signatures on public key certificates. cRLSign enables to use the public key to verify signatures on certificates revocation list.



```
root@debian:/home/debian/CA-ROOT# openssl req -new -x509 -days 365 -key private/
privcaroot.key -out certs/certcaroot.crt -config ./openssl.cnf -extensions CA_RO
OT
Enter pass phrase for private/privcaroot.key:
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.
-----
Country Name (2 letter code) [AU]:FR
Locality Name (eg, city) []:Paris
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Padis Inc.
Organizational Unit Name (eg, section) []:Duchesne section
Common Name (e.g. server FQDN or YOUR name) []:CA ROOT
Email Address []:gabriel.padis@edu.ece.fr
root@debian:/home/debian/CA-ROOT# ls certs
certcaroot.crt
```

Figure 10: Generating a self-signed root certificate

Copying the result of the resultant certificate, we can see several information :

- Data of the certificate, such as its version or serial number.
- Signature algorithm that were used for encryption
- All the information about the issuer that we entered while generating the certificate.
- The public key and its encryption algorithm
- All the information related to the extensions.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            99:d6:bc:b3:07:01:35:8a
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = FR, L = Paris, O = Padis Inc., OU = Duchesne section, CN = CA ROOT, emailAddress
= gabriel.padis@edu.ece.fr
        Validity
            Not Before: Feb 28 15:51:10 2019 GMT
            Not After : Feb 28 15:51:10 2020 GMT
        Subject: C = FR, L = Paris, O = Padis Inc., OU = Duchesne section, CN = CA ROOT, emailAddress
= gabriel.padis@edu.ece.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:bf:a1:24:08:52:b4:76:98:f8:68:a4:c5:4e:a4:
                    a8:df:ba:1f:27:6a:9f:ef:4f:2c:09:5c:cd:9b:7b:
                    cb:ae:16:b9:c0:17:00:f0:65:b3:c0:93:32:2e:26:
                    90:df:f8:bc:b4:38:ff:55:63:c5:2f:12:72:92:0e:
                    9b:08:64:5b:fa:61:58:5c:fb:75:35:cd:1b:b0:c2:
                    6e:9d:9f:e1:63:8a:c4:e6:0a:6b:f5:68:38:0f:33:
                    23:b0:16:8c:33:a7:3f:84:fe:26:b8:f0:f6:e1:3c:
                    cd:5e:13:5a:43:fb:69:96:d1:9b:a8:16:3f:e1:dc:
                    23:e0:0f:8e:7d:54:5f:8f:e0:e6:04:24:37:89:32:
                    78:6f:16:aa:66:9e:8c:42:bb:6a:7f:af:2e:ed:42:
                    da:80:fe:22:7d:ac:eb:79:f1:83:6a:b1:e2:28:ee:
                    f3:16:f6:31:26:64:ca:44:18:a4:52:c6:08:08:32:
                    13:42:11:89:9c:d9:78:a7:4c:67:c5:ee:b9:d2:2b:
                    63:85:14:5c:2b:bd:fb:fe:74:e8:8f:08:53:69:5e:
                    6a:20:80:e5:00:84:42:81:8e:f0:3f:9a:b2:1c:6f:
                    c5:4b:74:0b:90:28:a5:64:d7:72:32:ea:a3:40:02:
                    06:a0:01:90:d7:ad:63:10:06:bc:ad:e1:cf:69:db:
                    cd:d1
                Exponent: 65537 (0x10001)
        X509v3 extensions:
```

Figure 11: Certificate file

Finally, we update the configuration file openssl.conf with the certificate we generated and the private key we used to encrypt it.

```
[ ca ]^M
default_ca      = CA_default            # The default ca section^M
^M
####################################################################^M
[ CA_default ]^M
dir             = .                             # Where everything is kept^M
certs           = $dir/certs                    # Where the issued certs are kep
t^M
database        = $dir/index.txt                # database index file.^M
certificate     = $dir/certs/certcaroot.crt             # The CA certificate^M
new_certs_dir   = $dir/newcerts                 # default place for new certs.^M
serial          = $dir/serial                   # The current serial number^M
private_key     = $dir/private/privcaroot.key           # The private key^M
default_days    = 365^M
default_md      = sha256^M
preserve        = no^M
policy          = policy_match^M
^M
# For the CA policy^M
[ policy_match ]^M
countryName             = match^M
localityName            = match^M
organizationName        = match^M
"openssl.cnf" 72 lines, 2988 characters written
root@debian:/home/debian/CA-ROOT#
```

Figure 12: Updating openssl.conf

2.2 CA LAB

We're creating a second certification authority. We're going to create a certificate request for CA LAB, and send it to CA ROOT, so it can sign it. CA LAB will also be able to sign certificates.

First, we generate a private key RSA named privcalab.key, with a lenght of 2048 bits, and which password is "password".

```
root@debian:/home/debian/CA-LAB# openssl genrsa -des3 -out private/privcalab.key
 2048
Generating RSA private key, 2048 bit long modulus
.+++++
........................+++++
e is 65537 (0x010001)
Enter pass phrase for private/privcalab.key:
Verifying - Enter pass phrase for private/privcalab.key:
root@debian:/home/debian/CA-LAB# ls private/
privcalab.key
root@debian:/home/debian/CA-LAB# _
```

Figure 13: Generating a private key RSA

Then, we create a certificate request named certcalab.csr.

```
root@debian:/home/debian/CA-LAB# openssl req -new -key private/privcalab.key -ou
t certs/certcalab.csr -config ./openssl.cnf
Enter pass phrase for private/privcalab.key:
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.
-----
Country Name (2 letter code) [AU]:FR
Locality Name (eg, city) []:Paris
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Padis Inc.
Organizational Unit Name (eg, section) []:Duchesne section
Common Name (e.g. server FQDN or YOUR name) []:CA LAB
Email Address []:gabriel.padis@edu.ece.fr
root@debian:/home/debian/CA-LAB# ls certs
certcalab.csr
```

Figure 14: Generating a certificate request

- signed with the private key we just generated
- uses the configuration of the file openssl.conf

The request file has less information than the certificate one, there are only the issuer information, which later will have to match with the one enetered for CA ROOT certificate, but also the signature algorithms and public key.

```
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = FR, L = Paris, O = Padis Inc., OU = Duchesne section, CN = CA LAB, emailAddress
= gabriel.padis@edu.ece.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a3:42:b2:54:fd:3d:93:f3:55:71:06:84:40:ce:
                    6f:b6:89:80:69:3f:49:36:02:ea:cd:a9:ae:ab:99:
                    7e:d6:c4:c1:09:11:56:84:cc:e7:ca:28:5e:a1:d3:
                    e2:a5:e5:7d:25:ab:c9:01:47:57:46:de:75:c9:f6:
                    81:5f:0a:61:c3:24:da:6c:98:c9:23:5f:f0:35:0e:
                    f7:a9:4a:a4:ef:cd:f2:1f:da:33:0c:c5:74:75:9a:
                    24:72:25:38:b1:ce:57:df:99:25:a5:50:ed:c3:cf:
                    fc:d4:1f:9b:fd:90:c0:d5:2e:34:97:75:84:6b:bb:
                    5e:0c:ea:ff:84:ca:fc:c7:c2:73:27:02:af:04:dc:
                    bc:2e:de:f2:68:8b:25:ac:f3:81:10:03:6f:ef:a7:
                    e9:ad:c5:db:17:bf:8f:67:04:3d:ee:ae:a6:55:cd:
                    5d:8c:78:85:00:5f:e1:d6:b8:56:a0:5a:8f:dc:6a:
                    7e:a6:32:44:83:7a:6b:c2:d0:8d:ec:0d:05:09:b1:
                    a8:63:1b:e7:c8:dd:db:0d:ed:24:78:bc:77:d1:a5:
                    f7:6c:b9:b8:d3:2f:98:da:5c:5c:65:5b:df:8e:dc:
                    00:a8:72:4c:a5:28:16:a6:49:40:b4:05:d8:1d:1c:
                    40:e9:5c:9c:67:0b:c2:f1:af:6e:57:b5:2f:d4:0d:
                    35:47
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         07:a3:78:a7:10:36:1a:30:e4:a6:c8:c0:fd:53:9b:b3:6e:8e:
```

Figure 15: Output file of a certificate request

We send the request to PC2, so CA ROOT can sign the certificate. In order to do so, we use the following command line :

openssl ca –out certs/certcalab.crt -config ./openssl.cnf -extensions CA_LAB -infiles certs/certcalab.csr

The extensions that CA_LAB contains are similar to the CA_ROOT ones, and have the same purpose. There is only 2 new extensions.
In order to answer to the CA LAB request, CA ROOT, after verifying that the given information matches with the CA ones, and that there no existent certificates for it, generates and signs a new certificate. Then, we send the signed certificate back to CA LAB, to the PC1.

While looking at the output of the certcalab.crt generation, we see that there is no huge difference with the certcaroot.crt one. We added more information compared to the request output, mostly made of extensions.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = FR, L = Paris, O = Padis Inc., OU = Duchesne section, CN = CA ROOT, emailAddress
= gabriel.padis@edu.ece.fr
        Validity
            Not Before: Feb 28 16:07:20 2019 GMT
            Not After : Feb 28 16:07:20 2020 GMT
        Subject: C = FR, L = Paris, O = Padis Inc., OU = Duchesne section, CN = CA LAB, emailAddress
= gabriel.padis@edu.ece.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a3:42:b2:54:fd:3d:93:f3:55:71:06:84:40:ce:
                    6f:b6:89:80:69:3f:49:36:02:ea:cd:a9:ae:ab:99:
                    7e:d6:c4:c1:09:11:56:84:cc:e7:ca:28:5e:a1:d3:
                    e2:a5:e5:7d:25:ab:c9:01:47:57:46:de:75:c9:f6:
                    81:5f:0a:61:c3:24:da:6c:98:c9:23:5f:f0:35:0e:
                    f7:a9:4a:a4:ef:cd:f2:1f:da:33:0c:c5:74:75:9a:
                    24:72:25:38:b1:ce:57:df:99:25:a5:50:ed:c3:cf:
                    fc:d4:1f:9b:fd:90:c0:d5:2e:34:97:75:84:6b:bb:
                    5e:0c:ea:ff:84:ca:fc:c7:c2:73:27:02:af:04:dc:
                    bc:2e:de:f2:68:8b:25:ac:f3:81:10:03:6f:ef:a7:
                    e9:ad:c5:db:17:bf:8f:67:04:3d:ee:ae:a6:55:cd:
                    5d:8c:78:85:00:5f:e1:d6:b8:56:a0:5a:8f:dc:6a:
                    7e:a6:32:44:83:7a:6b:c2:d0:8d:ec:0d:05:09:b1:
                    a8:63:1b:e7:c8:dd:db:0d:ed:24:78:bc:77:d1:a5:
                    f7:6c:b9:b8:d3:2f:98:da:5c:5c:65:5b:df:8e:dc:
                    00:a8:72:4c:a5:28:16:a6:49:40:b4:05:d8:1d:1c:
                    40:e9:5c:9c:67:0b:c2:f1:af:6e:57:b5:2f:d4:0d:
                    35:47
                Exponent: 65537 (0x10001)
        X509v3 extensions:
```

Figure 16: certcalab.crt output

Once PC1 received the signed certificate, we can update the configuration in openssl.conf, by adding the path to the certificate and the private key to the default section.

```
####################################################################^M
[ CA_default ]^M
dir             = .                             # Where everything is kept^M
certs           = $dir/certs                    # Where the issued certs are kep
t^M
database        = $dir/index.txt                # database index file.^M
certificate     = $dir/certs/certcalab.crt      # The CA certificate^M
new_certs_dir   = $dir/newcerts                 # default place for new certs.^M
serial          = $dir/serial                   # The current serial number^M
private_key     = $dir/private/privcalab.key            # The private key^M
default_days    = 365^M
default_md      = sha256^M
preserve        = no^M
policy          = policy_match^M
^M
```

Figure 17: Updating openssl.conf

2.3 Server Certificate

On the server, we're not going to create any certification authorities.
It is the most bottom-level certificate on the tree.

First, we're generating a private key RSA without any password. To do so, we just had to remove **-des3** part of the command line.

```
root@debian:/home/debian# openssl genrsa -out privsrv.key 2048
Generating RSA private key, 2048 bit long modulus
......................+++++
.......+++++
e is 65537 (0x010001)
```

Figure 18: Generating private key

Once it's done, we're creating a certificate request for the server. It works the same way as for the CA LAB one. The only difference is that we don't need any configuration file, and that there is no password. This is not surpring since this certificate isn't for a CA.

```
root@debian:/home/debian# openssl req -new -key privsrv.key -out certsrv.csr
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.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:Paris
Locality Name (eg, city) []:Paris
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Padis Inc.
Organizational Unit Name (eg, section) []:Duchesne section
Common Name (e.g. server FQDN or YOUR name) []:DuchesnePadis
Email Address []:gabriel.padis@edu.ece.fr

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Figure 19: Generating a certificate request

The output of the request mainly contains the same information as before : issuers information, version number, and public key and signature algorithm.

```
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = FR, ST = Paris, L = Paris, O = Padis Inc., OU = Duchesne section, CN =
DuchesnePadis, emailAddress = gabriel.padis@edu.ece.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c9:c2:75:75:38:e0:13:48:5b:f3:b7:1b:03:dc:
                    92:57:d1:01:fe:0c:ee:85:02:03:4f:8e:1e:bf:2c:
                    9e:ed:c9:76:33:64:1d:d6:bc:6a:cf:24:73:27:1a:
                    f6:91:9a:a4:fd:7e:e9:2f:7f:4c:92:dd:4c:60:5d:
                    38:ba:77:07:f9:00:bc:ea:73:75:16:38:3f:60:c6:
                    19:23:08:59:63:20:bb:7e:5d:3f:b5:6b:72:9d:80:
                    46:b5:bd:8c:3c:13:84:6f:2a:bd:ba:01:9b:95:da:
                    b0:66:15:de:0b:0a:df:c9:2f:62:e4:02:59:c7:8a:
                    f0:c7:22:c5:63:6a:55:63:aa:0e:60:8c:cf:fa:e4:
                    eb:8e:4d:30:19:e7:6e:81:fc:60:0b:dc:65:b3:22:
                    f0:d1:bc:b8:fd:3d:75:8a:3c:ff:a9:da:fa:4f:72:
                    80:6e:44:90:30:dc:9f:12:3e:4b:b7:27:da:d2:e6:
                    ae:83:67:f6:c0:e0:86:42:ce:14:56:0b:a2:56:e1:
                    0f:e1:25:de:31:7c:f8:21:81:6d:9f:d7:3a:88:39:
                    21:a8:88:25:5d:bf:d1:bd:39:ad:1a:39:c7:b1:8e:
                    7c:03:43:68:19:1c:f7:84:38:76:15:79:dd:bc:66:
                    63:8f:4a:18:88:59:ce:a2:ac:ec:19:83:e0:2a:32:
                    6d:c5
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         8a:cb:78:a9:fb:db:ad:b9:31:a8:06:48:ad:d3:f8:54:a4:83:
```

Figure 20: Output of the certificate request

We send our request to PC1, so we can ask CA LAB to answer to it, and to generate a signed certificate for the server.
Before generating the signed certificate, we update the configuration file openssl.conf, in order to add the server IP address in the server section.

Finally, we sign a certificate with CA LAB, and we generate certsrv.crt, while using SERVER extensions.

```
Using configuration from ./openssl.cnf
Enter pass phrase for ./private/privcalab.key:
Can't open ./index.txt.attr for reading, No such file or directory
139668069565824:error:02001002:system library:fopen:No such file or directory:..
/crypto/bio/bss_file.c:74:fopen('./index.txt.attr','r')
139668069565824:error:2006D080:BIO routines:BIO_new_file:no such file:../crypto/
bio/bss_file.c:81:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :ASN.1 12:'Paris'
localityName          :ASN.1 12:'Paris'
organizationName      :ASN.1 12:'Padis Inc.'
organizationalUnitName:ASN.1 12:'Duchesne section'
commonName            :ASN.1 12:'DuchesnePadis'
emailAddress          :IA5STRING:'gabriel.padis@edu.ece.fr'
Certificate is to be certified until Feb 28 17:00:00 2020 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Figure 21: Generating a signed certificate for the server

Here again, even though if one or two extensions varies, they are pretty the same ones as CA_LAB ones. We see that the main difference are :

- KeyUsage :
- subjectAltName :

The certificate output contains the same information as per usual : the beginning and end date of the certificate, the issuer and subject information, encryption algorithms...

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = FR, L = Paris, O = Padis Inc., OU = Duchesne section, CN = CA LAB, emailAddress =
gabriel.padis@edu.ece.fr
        Validity
            Not Before: Feb 28 17:00:00 2019 GMT
            Not After : Feb 28 17:00:00 2020 GMT
        Subject: C = FR, L = Paris, O = Padis Inc., OU = Duchesne section, CN = DuchesnePadis,
emailAddress = gabriel.padis@edu.ece.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c9:c2:75:75:38:e0:13:48:5b:f3:b7:1b:03:dc:
                    92:57:d1:01:fe:0c:ee:85:02:03:4f:8e:1e:bf:2c:
                    9e:ed:c9:76:33:64:1d:d6:bc:6a:cf:24:73:27:1a:
                    f6:91:9a:a4:fd:7e:e9:2f:7f:4c:92:dd:4c:60:5d:
                    38:ba:77:07:f9:00:bc:ea:73:75:16:38:3f:60:c6:
                    19:23:08:59:63:20:bb:7e:5d:3f:b5:6b:72:9d:80:
                    46:b5:bd:8c:3c:13:84:6f:2a:bd:ba:01:9b:95:da:
                    b0:66:15:de:0b:0a:df:c9:2f:62:e4:02:59:c7:8a:
                    f0:c7:22:c5:63:6a:55:63:aa:0e:60:8c:cf:fa:e4:
                    eb:8e:4d:30:19:e7:6e:81:fc:60:0b:dc:65:b3:22:
                    f0:d1:bc:b8:fd:3d:75:8a:3c:ff:a9:da:fa:4f:72:
                    80:6e:44:90:30:dc:9f:12:3e:4b:b7:27:da:d2:e6:
                    ae:83:67:f6:c0:e0:86:42:ce:14:56:0b:a2:56:e1:
                    0f:e1:25:de:31:7c:f8:21:81:6d:9f:d7:3a:88:39:
                    21:a8:88:25:5d:bf:d1:bd:39:ad:1a:39:c7:b1:8e:
                    7c:03:43:68:19:1c:f7:84:38:76:15:79:dd:bc:66:
                    63:8f:4a:18:88:59:ce:a2:ac:ec:19:83:e0:2a:32:
                    6d:c5
                Exponent: 65537 (0x10001)
        X509v3 extensions:
```

Figure 22: Output of certsrv.crt

When we cat **index.txt** we see that it stores the issuer/subject information. By storing issuer/subject information, it ensures that when you're making a certificate, you're not going to make twice a certificate for the same request and organism.

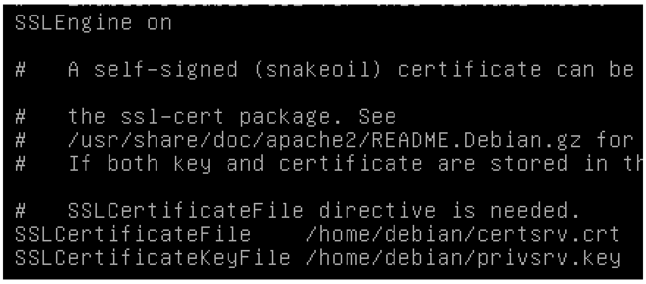
```
root@debian:/home/debian/CA-LAB# cat index.txt
V       200228170000Z           01      unknown /C=FR/L=Paris/O=Padis Inc./OU=Du
chesne section/CN=DuchesnePadis/emailAddress=gabriel.padis@edu.ece.fr
```

Figure 23: index.txt output

Signing a certificate update several files besides **index.txt**, such as the **serial number** for example, that increases each time.
Finally, we send back the signed certificate to the server.

2.4 HTTPS Server

First we activate the ssl module with the **"a2enmod ssl"** line command. Then, we restart apache2 service.
In order to configure our website with https, we have to add some few lines to the



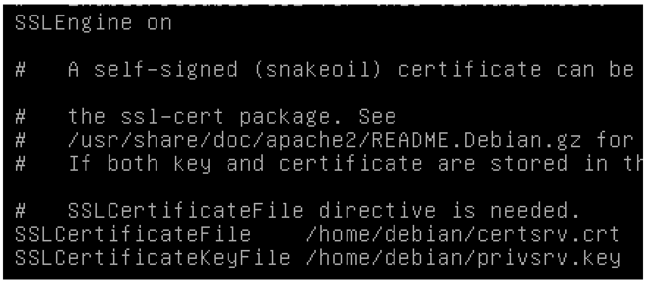
```
SSLEngine on

#   A self-signed (snakeoil) certificate can be

#   the ssl-cert package. See
#   /usr/share/doc/apache2/README.Debian.gz for
#   If both key and certificate are stored in th

#   SSLCertificateFile directive is needed.
SSLCertificateFile    /home/debian/certsrv.crt
SSLCertificateKeyFile /home/debian/privsrv.key
```

Figure 24: Configuring the default ssl configuration file

file. We're creating a new default page.

```
root@debian:/home/debian# cd /etc/apache2/sites-available/
root@debian:/etc/apache2/sites-available# ls
000-default.conf  default-ssl.conf
```

Figure 25: Creating a new default page



```
SSLEngine on

#   A self-signed (snakeoil) certificate can be

#   the ssl-cert package. See
#   /usr/share/doc/apache2/README.Debian.gz for
#   If both key and certificate are stored in th

#   SSLCertificateFile directive is needed.
SSLCertificateFile    /home/debian/certsrv.crt
SSLCertificateKeyFile /home/debian/privsrv.key
```

Figure 26: Configuring the default ssl configuration file

The important configurations are :

- **VirtualHost _default_:443** we're precising on which port we're listening, and who we are listening. 443 is a SSL port.
- **SSLEngine On** : enable the SSL on the page, in order to use a secure connection through certificates
- **SSLCertificateFile** : the certificate used for our secure connection
- **SSLCertificateKeyFile** : the key we used to encrypt our certificate.

Once our modifications are done, we have to enable our site by launching **a2ensite default-ssl** command line.

We check that in **/etc/apache2/ports.conf** the port 443 for SSL and 80 for HTTP are enabled. Either way, we append them to the file.

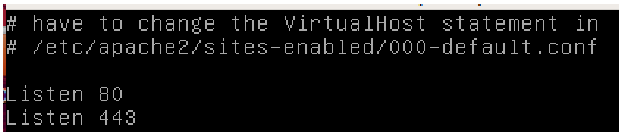


Figure 27: Checking the ports configuration

Finally, we restart the apache services, and we're ready to use our server new page we just created !

When we try to connect to the server from PC3-Host using a secure connection, we see an error :

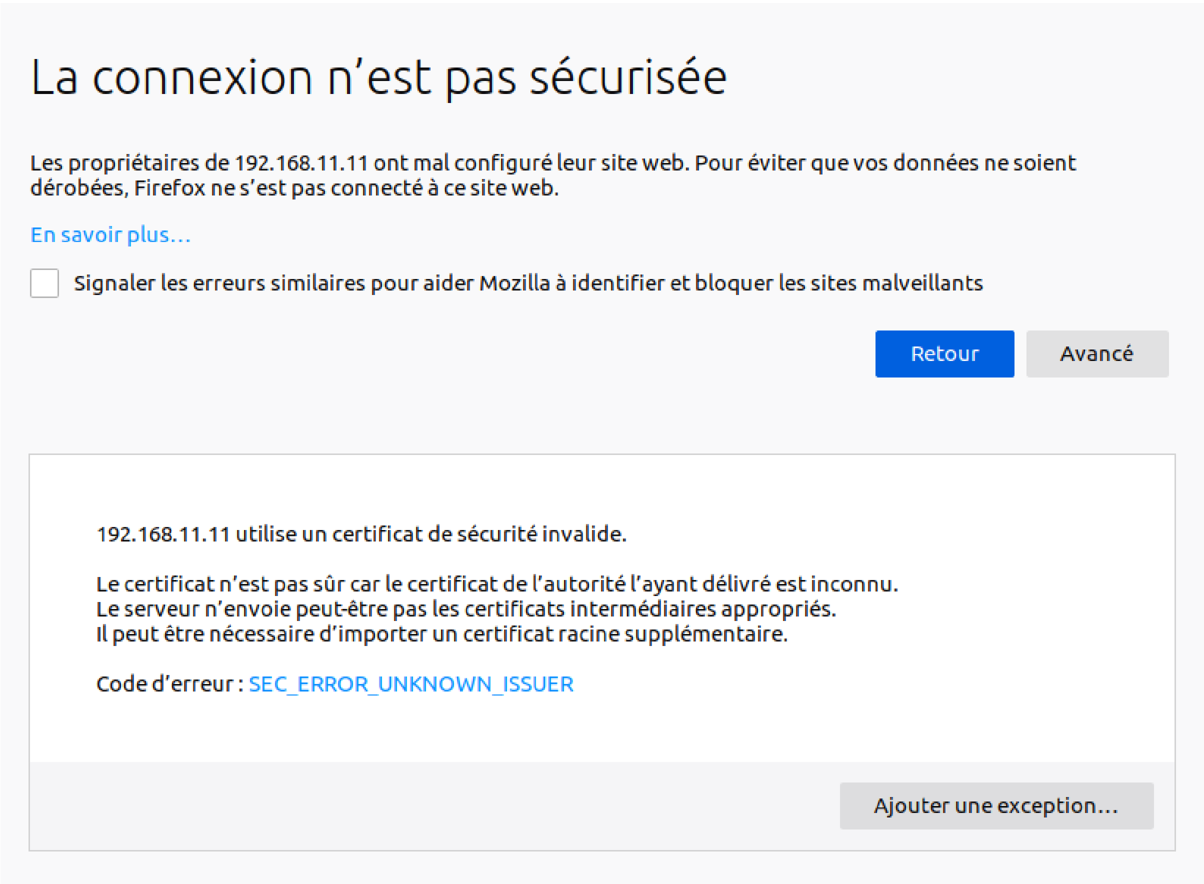


Figure 28: Error

There is an error saying that the CA that delivered our certificate is unknown. It is because we didn't update our browser with the CA certificates. Without the CA specification, our certificate has no value, because nothing proves that it is safe. It is also possible to access to our server's certificate.

In order to fix this problem, we have to create the CA related to our Organization, by importing CA ROOT and CA LAB certificates into our browser.

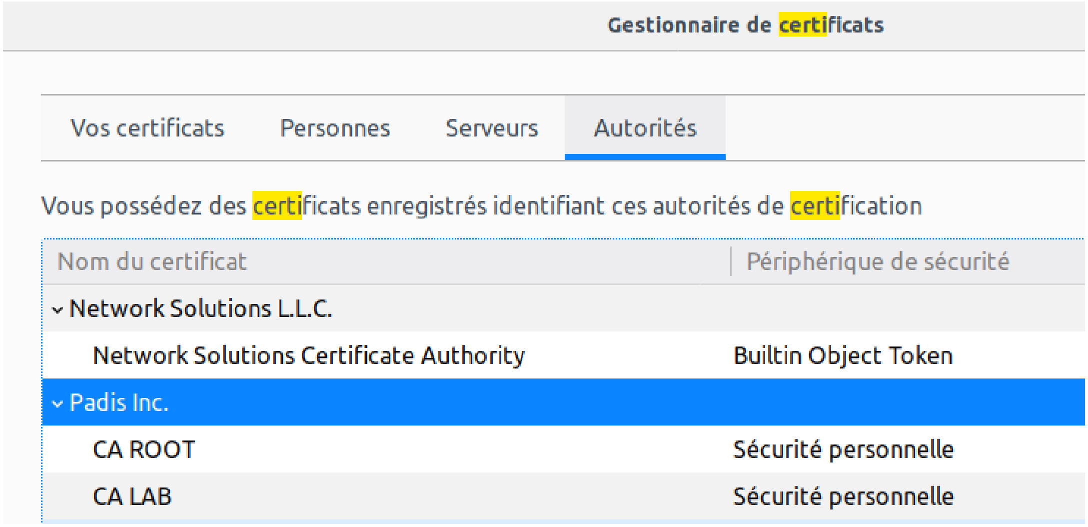


Figure 29: Importing the needed certificates

We do so, then we restart the browser and apache2 services. The secure connection to our server works fine now.

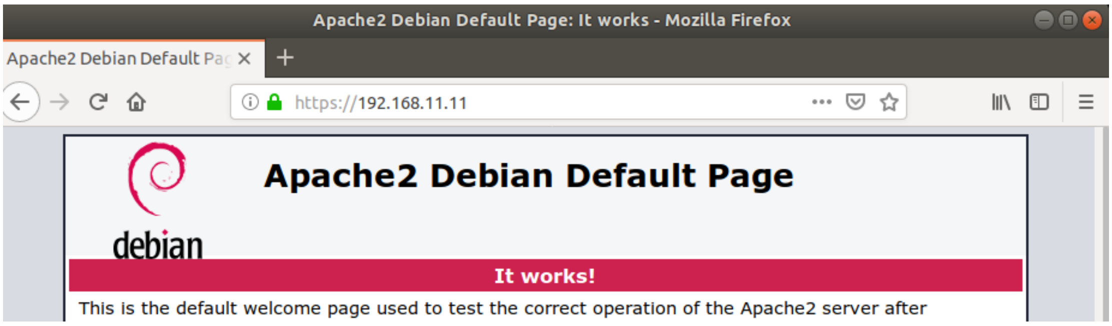
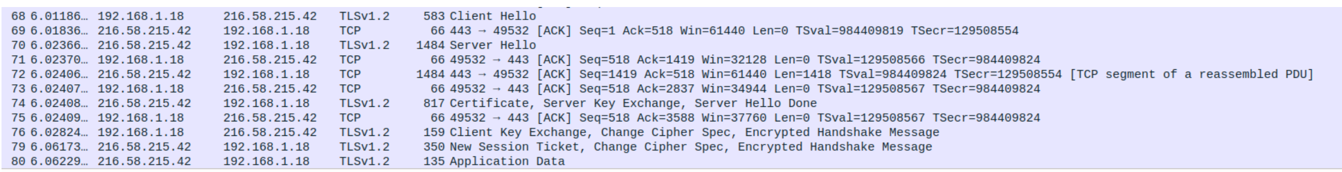


Figure 30: https://192.168.11.11 connection

The main difference with the question above is that, now that we imported our CA, the server certificate authenticity can be verified. This way, we know that it is a safe certificate. The connection is now considered as secure.

Let's capture the traffic with Wireshark :



68	6.01186…	192.168.1.18	216.58.215.42	TLSv1.2	583	Client Hello
69	6.01836…	216.58.215.42	192.168.1.18	TCP	66	443 → 49532 [ACK] Seq=1 Ack=518 Win=61440 Len=0 TSval=984409819 TSecr=129508554
70	6.02366…	216.58.215.42	192.168.1.18	TLSv1.2	1484	Server Hello
71	6.02370…	192.168.1.18	216.58.215.42	TCP	66	49532 → 443 [ACK] Seq=518 Ack=1419 Win=32128 Len=0 TSval=129508566 TSecr=984409824
72	6.02406…	216.58.215.42	192.168.1.18	TCP	1484	443 → 49532 [ACK] Seq=1419 Ack=518 Win=61440 Len=1418 TSval=984409824 TSecr=129508554 [TCP segment of a reassembled PDU]
73	6.02407…	192.168.1.18	216.58.215.42	TCP	66	49532 → 443 [ACK] Seq=518 Ack=2837 Win=34944 Len=0 TSval=129508567 TSecr=984409824
74	6.02408…	216.58.215.42	192.168.1.18	TLSv1.2	817	Certificate, Server Key Exchange, Server Hello Done
75	6.02409…	192.168.1.18	216.58.215.42	TCP	66	49532 → 443 [ACK] Seq=518 Ack=3588 Win=37760 Len=0 TSval=129508567 TSecr=984409824
76	6.02824…	192.168.1.18	216.58.215.42	TLSv1.2	159	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
79	6.06173…	216.58.215.42	192.168.1.18	TLSv1.2	350	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
80	6.06229…	216.58.215.42	192.168.1.18	TLSv1.2	135	Application Data

Figure 31: Wireshark capture of the Handshake protocol

We saw in class the following sequence diagram for the Handshake protocol :

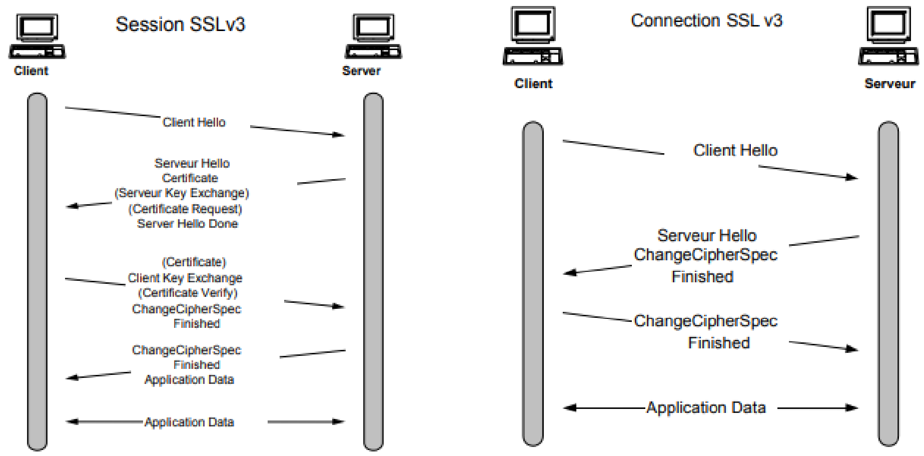


Figure 32: Handshake protocol sequence diagram seen in class

We clearly see the different steps presented in class, on our Wireshark capture :

- Server and Client acknowledge each other

```
192.168.1.18     216.58.215.42    TLSv1.2     583 Client Hello
216.58.215.42    192.168.1.18     TCP          66 443 → 49532 [ACK]
216.58.215.42    192.168.1.18     TLSv1.2    1484 Server Hello
```

- Server and Client exchange security parameters. Server authenticates.

```
192.168.1.18     TLSv1.2     817 Certificate, Server Key Exchange, Server Hello Done
```

- Client authentication. Data encryption

```
216.58.215.42    TLSv1.2     159 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
192.168.1.18     TLSv1.2     350 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
```

3 Part 2 : Proxy

Proxy is another way to secure our connection. It works in the same way as a firewall. It allows or denies connections to some website that are either on the white or black list.

3.1 Configuring Squid

First, we configure visible_hostname, and the cache with the default values.

```
visible_hostname proxy.lab.com
```

Figure 33: visible_hostname

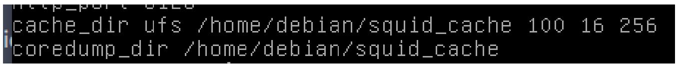


Figure 34: Configuring cache with the default values

In order to configure the cache, we have to be sure that squid has the right permissions to access it. To do so, we change the access rights of the file with **chown 777 /home/debian/squid_cache**.

Then, we configure the proxy on our browser. We have to precise that our proxy server address is the **eth0** interface's address : **192.168.11.1**. We also have to precise the port we're listening on. Squid port is always 3128. We configure the proxy for HTTP and SSL (HTTPS) connections.

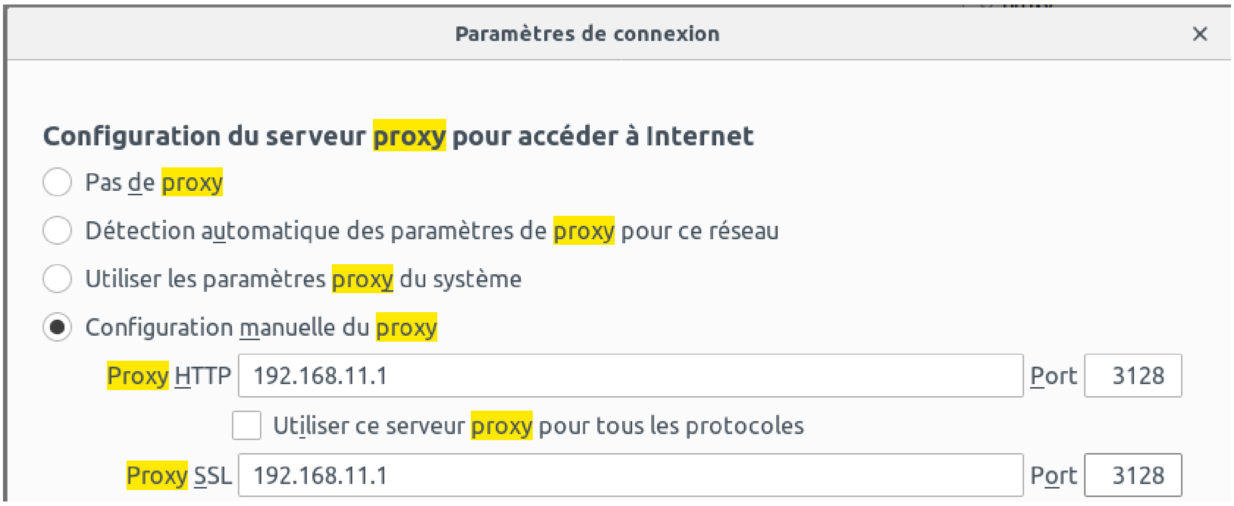


Figure 35: Proxy configuration on firefox

When we try to connect to our server using http, we have an error.

Figure 36: Error

It is normal since there is no rule allowing the access from our server.
When we look at our access logs, we see the result code **TCP_DENIED/403**.

```
1551653113.822      5 192.168.11.254 TCP_DENIED/403 2365 GET http://192.168.11.11/ - HIER_NONE/- text/html
1551653113.867      0 192.168.11.254 TCP_DENIED/403 2360 GET http://192.168.11.11/favicon.ico - HIER_NONE/- text/html
```

Figure 37: Error

TCP_DENIED means that the request was denied by the access controls. **403** means that it was a denied because it was "**Forbidden**".

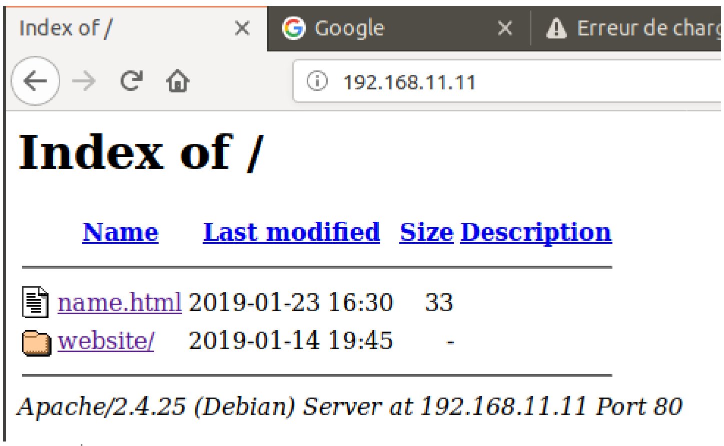
In order to solve this problem, we're going to create an access list labnet, to allow our LAN **192.168.11.0/24** to use proxy squid.

```
acl labnet src 192.168.11.0/24
http_access allow labnet
```

Figure 38: Creating access list and allowing access

3.2 Accessing to services through proxy

We can now access to our server :




```
1551653255.027   4969 192.168.11.254 TCP_TUNNEL/200 721 CONNECT 192.168.11.11:443 - HIER_DIRECT/192.168.11.11 -
1551653255.027   4969 192.168.11.254 TCP_TUNNEL/200 725 CONNECT 192.168.11.11:443 - HIER_DIRECT/192.168.11.11 -
1551653255.028   5007 192.168.11.254 TCP_TUNNEL/200 3743 CONNECT 192.168.11.11:443 - HIER_DIRECT/192.168.11.11 -
```

Figure 39: Accessing to our server with http

TCP_REFRESH_MODIFIED means that the revalidation produced a new modified object. **200** means **OK**

TCP_HIT means that the response delivered was taken from the cache directories, the disk.

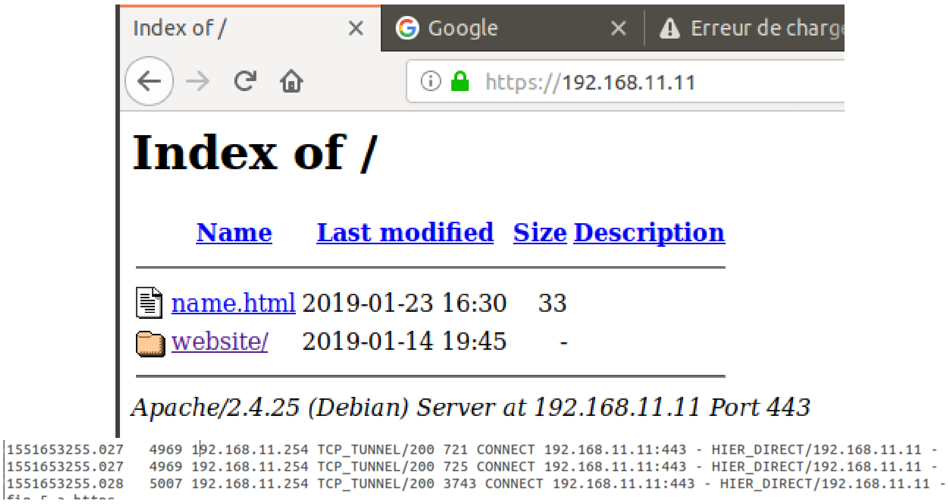


Figure 40: Accessing to our server with https

TCP_TUNNEL means that the protocol was tunneled. It means that the connection was secured and encapsulated, since it is an SSL conection. **200** means **OK.**

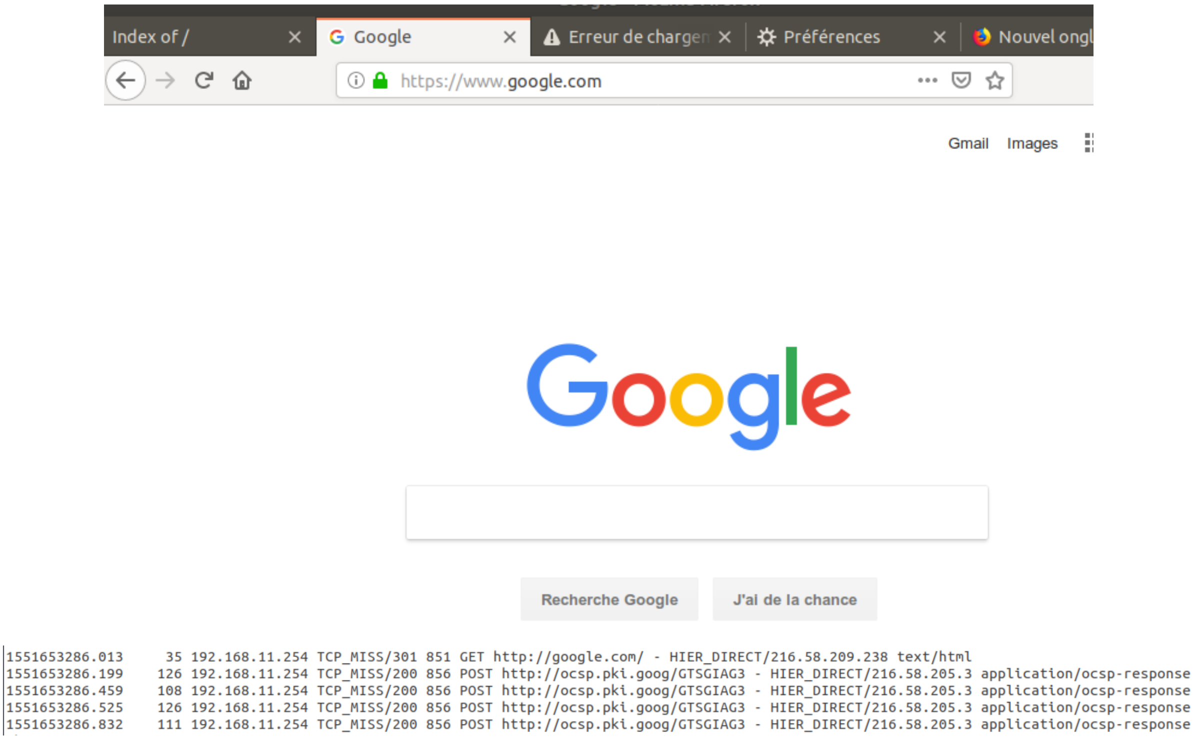


Figure 41: Accessing to Google

TCP_MISS means that, unlike TCP_HIT, the object delivered was not in the cache, it is a network response object. **301** means **moved permananty**. **200** means ok.

3.3 Clearing cache

```
1551653392.483      8 192.168.11.254 TCP_REFRESH_MODIFIED/200 786 GET http://192.168.11.11/ - HIER_DIRECT/192.168.11.11 text/html
1551653392.529      0 192.168.11.254 TCP_MEM_HIT/200 529 GET http://192.168.11.11/icons/blank.gif - HIER_NONE/- image/gif
1551653392.530      0 192.168.11.254 TCP_MEM_HIT/200 606 GET http://192.168.11.11/icons/folder.gif - HIER_NONE/- image/gif
1551653392.530      0 192.168.11.254 TCP_MEM_HIT/200 610 GET http://192.168.11.11/icons/text.gif - HIER_NONE/- image/gif
1551653392.533      0 192.168.11.254 TCP_MISS/404 592 GET http://192.168.11.11/favicon.ico - HIER_DIRECT/192.168.11.11 text/html
```

Figure 42: Logs after cleaning cache

TCP_MEM_HIT means that the response delivered was taken from the memory, the RAM. This is normal since we just cleared the cache, it had to retrieve the response from somewhere else.

3.4 Configuring basic NCSA

NCSA is used to permit us to authenticate to our proxy with a username and a password.

First, we have to create a file in which will be stored the usernames with their corresponding passwords. To do this, we use htpasswd.

- -c option creates a new file
- -m option encrypts the passwords with MD5 algorithm.

```
root@debian:/etc/squid# htpasswd -c -m /etc/squid/users user1
New password:
Re-type new password:
Adding password for user user1
root@debian:/etc/squid# chmod o+r /etc/squid/users
```

Figure 43: NCSA first step

Then, we change the rights of the file so we sure that squid can have access to it. **o+r** means everyone ("**others**") can **read** it.

Then we do two checks. First, we search for the path of the ncsa auth file.

```
root@debian:/etc/squid# dpkg -L squid | grep ncsa_auth
/usr/lib/squid/basic_ncsa_auth
/usr/share/man/man8/basic_ncsa_auth.8.gz
```

Figure 44: Checking for the path

Then, we check that everything works fine :

```
root@debian:/etc/squid# /usr/lib/squid/basic_ncsa_auth /etc/squid/users
user1 password
OK
```

Figure 45: Files working fine

Finally, the last thing to do is to update **squid.conf** file. At the beginning, we have to uncomment the authentication parameters :

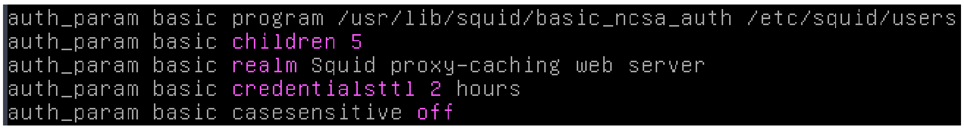


Figure 46: auth$_{param}$

- **program :** specifies where are stored the passwords and the helper program location.
- **children 5** : the number of authenticator processes to spawn. It means that 5 users can connect to it.
- **realm Squid proxy-caching web server:** the text that will appear in the connection window.
- **credentialsttl 2 hours:** how long the authentication lasts. Our authentication lasts 2 jours.

Then we have to modify our access list so users will have to authenticate and be on the access list in order to go through the proxy:

Figure 47: acl

Figure 48: http_access

We have to restart squid. Then, the authentication works !

Figure 49: http_access

3.5 Blocking Facebook

We want our proxy to block the connection to Facebook. To do so, we just have to change the access list, and to deny the access to the **destination domain** "Facebook".

```
acl fb dstdomain www.facebook.com
```

Figure 50: Referencing Facebook in the access list

```
http_access deny fb
http_reply_access deny fb
#http_access deny CONNECT fb
```

Figure 51: Denying all connections from Facebook

We denied :

- **http_access :** access to the destination domain
- **http_reply_access :** replies to the client requests.
- **http_access :** https connections

```
http_access deny fb
http_reply_access deny fb
#http_access deny CONNECT fb
```

Figure 52: Denying all connections from Facebook

When we try to connect to Facebook, the connection is refused by the proxy server.

Figure 53: Connection refused

```
1551653450.363      0 192.168.11.254 TCP_DENIED/403 2233 CONNECT www.facebook.com:443 - HIER_NONE/- text/html
```

Figure 54: access logs

As you can see, the code is **TCP_DENIED** for our attempt to connect to Facebook. 403 means "forbidden".

3.6 Proxy Auto-configuration

A .pac file contains a set of rules that redirects web traffic. It decides if it goes directly to the destination, or if has to pass through the proxy. It is coded in JavaScript.

We want our PAC to do the following things :

- if we're trying to connect to our server, it must be direct.
- else, we have to go through the proxy.

This is our proxy.pac file :

```
function FindProxyForURL(url, host) {

    if(shExpMatch(host, "192.168.11.11"))
```

```
        {
                return "DIRECT";
        }
        else
        {
                return "PROXY 192.168.11.1:3128; DIRECT";
        }
}
```

We have to change the settings in our browser too :

Figure 55: Using a pac file

https://netfilter.org/documentation/HOWTO/fr/NAT-HOWTO-6.html http://www.admin6.fr/2012/03/r
de-routage-simple-avec-iptables/ https://en.wikipedia.org/wiki/Root_certificate