Security Insider Lab I - Report 5

by

Subbulakshmi Thillairajan, Fabian Göttl, Mohamed Belkhechine

Exercise 1:

Exercise 1.1:

Firefox:

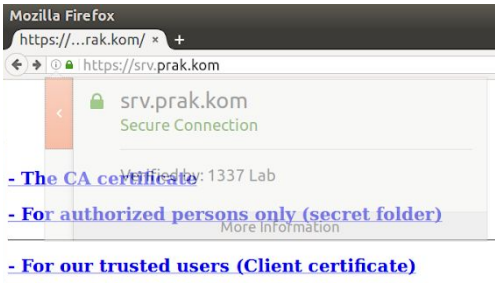
- Found the webserver IP 192.168.0.101 by nmap
- Goto page https://192.168.0.101
- Firefox shows a connection is not secure warning

Exercise 1.2:

- We can not access a encrypted version: https://192.168.0.101
- Connection can't be established because insecure/not trusted ERR: Unknown issuer
- Firefox is not trusting the certificate issuer as it was signed by a unknown CA
- But we can go to the unencrypted version: http://192.168.0.101
- Or: Access the site (temporary) by adding a Security exception to download the CA cert

Exercise 1.3:

HTTP: The connection is not secure.
HTTPS + Security Exception: The traffic is encrypted, but we can not trust it as we were downloading a CA cert, which may was manipulated by a man-in-the-middle.

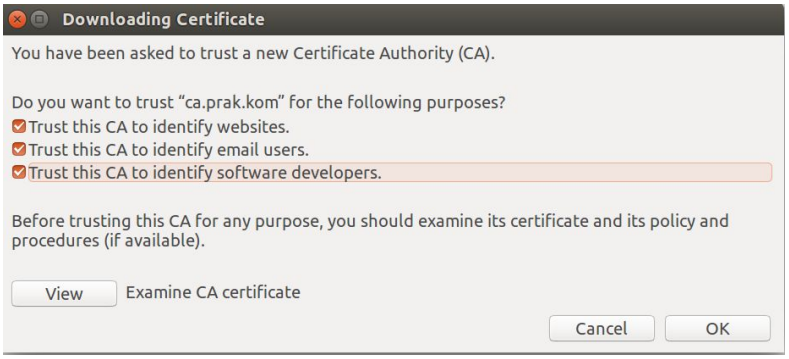


Exercise 1.4:

Server/Certificate details:
CN: ca.prak.kom
O: 1337 Lab
Serial Number: 01
Begins on: 01.01.2000
Expires on: 01.01.2101

Exercise 1.5:

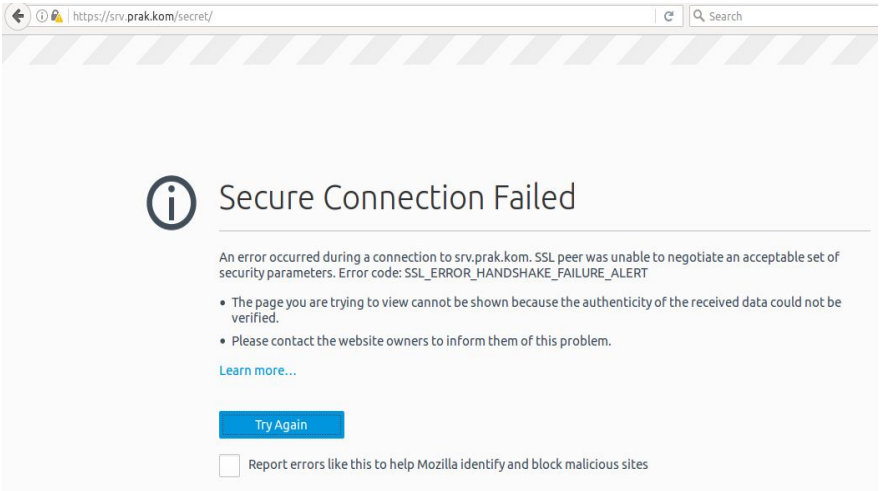
- Go to the encrypted version: https://192.168.0.101
- Download CA certificate



- Delete server cert in Settings->Advanced->View certificates->Servers
- Lab's CA is now trusted (which fixes the Unknown issuer error)
- Now: Connection ERR: INVALID_DOMAIN (because CN does not match the IP)
- Add IP and CN domain to /etc/hosts file: "192.168.0.101 srv.prak.kom"
- Goto https://srv.prak.kom (CN matches the domain)
- Secure connection established
- See Figure 1.4
- We can browse the / directory.
- The /secret is not accessable.

We can not access it in the first time as the certificate is not trusted. The reason is that the domain name (CN) is not matching the address that we are using to connect to the server. In our case this was the IP address 192.168.0.101. If we add "192.168.0.101 srv.prak.kom" to the /etc/hosts file, we are able to connect to https://srv.prak.kom.

Exercise 1.6:
We are not able to access the secret folder as we have the client certificate not installed yet.
Error: SSL_ERROR_HANDSHAKE_FAILURE_ALERT



Exercise 1.7:
Firefox: Settings->Advanced->View certificates
The lab's CA certificate can be found under Authorities. Its details are shown in the second image.

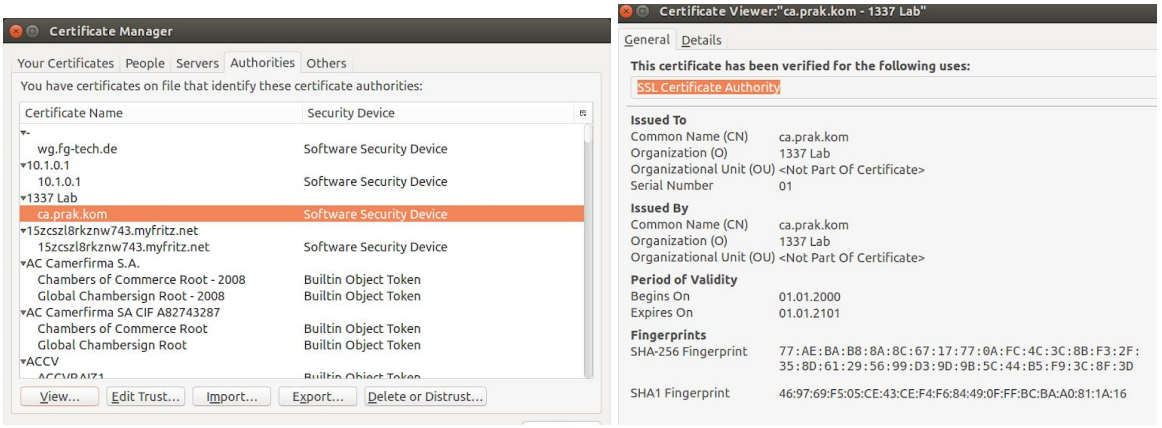


Figure: Lab's server certificate authority.

Exercise 1.8:
Export by clicking at Export in the *View certificates* list. Export options are following file formats: PEM, PEM with chain, DER, PKCS#7, PKCS#7 with chain.

Exercise 1.9:
The CA is now trusted and we installed the client certificate. We are authorised and can access the /secret folder.

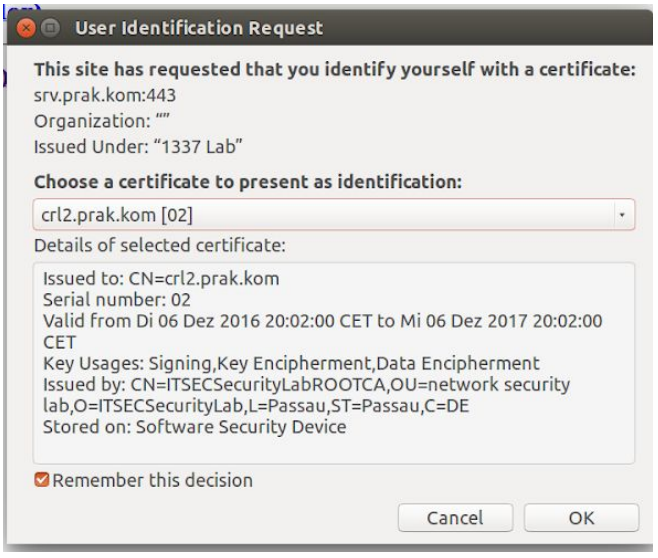


Figure: Details of the client certificate.

Exercise 1.10:

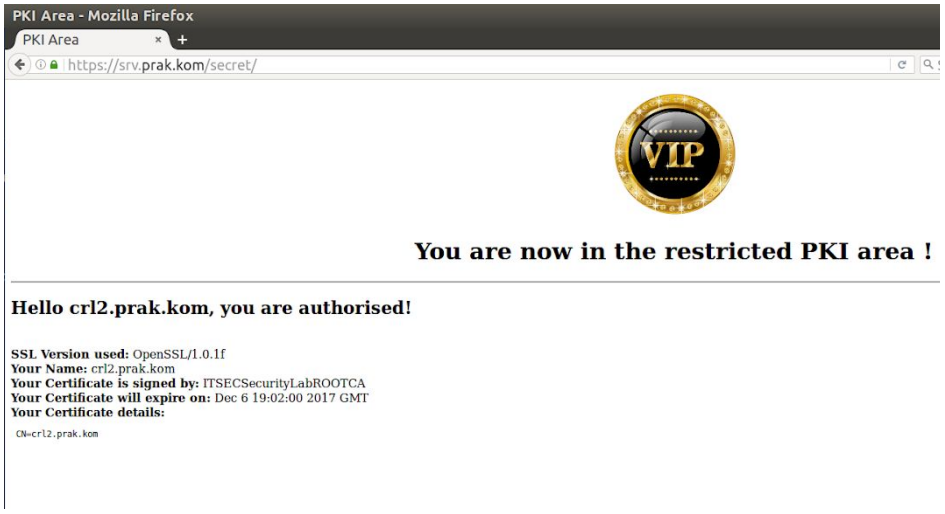


Figure: Successful access of the secret folder.

Exercise 2:

Exercise 2.1:

On host machine (CA) execute:
openssl genrsa -des3 -out cakey.pem 2048
less cakey.pem

Figure: Certificate Authority private key.

And then to generate the public key out of the private key we executed the following:
openssl rsa -in cakey.pem -outform PEM -pubout -out public.pem
less public.pem

Figure: Certificate Authority public key.

Exercise 2.2:

To generate the root self-signed certificate for the CA we executed the following command:
openssl req -new -x509 -extensions v3_ca -keyout cakey.pem -out cacert.pem -days 3650

In order to generate a private key for the machine server-hh we did the following on server-hh:
openssl genrsa -des3 -out server-hh-private.pem 2048

Next, to generate the CSR from the private key we run this command:
openssl req -key server-hh-private.pem -new -out server-hh-request.csr

Transfer server-hh-request.csr to CA:
scp server-hh-request.csr mbelkhechine@192.168.56.101:/home/server-hh

Before starting to sign our clients certificates we created the following folder structure in the CA:
sudo mkdir /etc/ssl/CA
sudo mkdir /etc/ssl/newcerts

Where CA folders will hold the CA certificate files. In order to keep track of the last serial number, we executed the following commands:

sudo sh -c "echo '01' > /etc/ssl/CA/serial"
sudo touch /etc/ssl/CA/index.txt

And then finally we edit the file /etc/ssl/openssl.cnf in a way to include our CA private key, the root self-signed certificate and file structure that we just created:

```
dir             = /etc/ssl              # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/CA/index.txt     # database index file.
#unique_subject = no                    # Set to 'no' to allow creation of
                                        # several ctificates with same subject.
new_certs_dir   = $dir/newcerts         # default place for new certs.

certificate     = $dir/certs/cacert.pem [redacted]# The CA certificate
serial          = $dir/CA/serial [redacted]# The current serial number
crlnumber       = $dir/crlnumber        # the current crl number
                                        # must be commented out to leave a V1 C$
crl             = $dir/crl.pem [redacted]# The current CRL
private_key     = $dir/private/cakey.pem# The private key
RANDFILE        = $dir/private/.rand    # private random number file
```

Figure: openssl.cnf file.

Now from the CA machine we execute the following command to sign our first client certificate request:
openssl ca -in server-hh-request.csr -config /etc/ssl/openssl.cnf

If we point our terminal to the folder /etc/ssl/newcerts/ we will find server-hh certificate. The certificate will have a unique serial number equal to 01.

Next, to send back the certificate to server-hh we executed the following command:
scp /etc/ssl/newcerts/01.pem server-hh@192.168.56.101:/etc/apache2/ssl/01.pem

Exercise 2.3:
Initial trust is created by a chain of authorities. The chain start at root certificates, which are initially trusted and are shipped with the browser's download package. These may direct trust to other intermediate CAs in the chain. The chain ends at the certificate with the server's CN.
Other initially trusted CAs: GeoTrust, RapidSSL, GlobalSign

Exercise 2.4:
Man in the middle attack of Google Mail in Iran's Internet: Attackers changed the DNS of mail.google.com to their webservers. In order to establish a "secure" SSL connection to clients, the attackers issued their own certificated by an eavesdropped private key of an Dutch CA. As the CA was trusted in all browsers, they accepted the certificate and showed a lock.

Exercise 3:

Exercise 3.1:
The ssl configuration is located in the file: /etc/apache2/sites-available/default-ssl.conf

Exercise 3.2:

While enabling Apache's SSL mod, a self-signed certificate is generated. The certificate is issued to the default hostname given in /etc/hostname.

Enable Apache's ssl module:
a2enmod ssl
a2ensite default-ssl

And then let's send the CA root self-signed certificate to the server-hh:
scp /etc/ssl/certs/cacert.pem server-hh@192.168.56.101:/etc/apache2/ssl/cacert.pem

Let's configure the file default-ssl.conf:

```
 GNU nano 2.5.3            File: default-ssl.conf

<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        ServerAdmin server-hh@127.0.0.1
        ServerName ca.group6serverhh.kom
        #ServerAlias www.your_domain.com
        DocumentRoot /var/www/html
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
        SSLEngine on
        SSLVerifyClient none
        SSLCertificateFile /etc/apache2/ssl/01.pem
        SSLCertificateKeyFile /etc/apache2/ssl/private.pem
        SSLCertificateChainFile /etc/apache2/ssl/cacert.pem
        SSLCACertificateFile /etc/apache2/ssl/cacert.pem
```

Figure: default-ssl.conf

Finally, we restart the Apache service:
service apache2 restart

In order to make apache2 restart without asking us for the rsa key certificate:

Add to /etc/apache/apache.conf:
SSLPassPhraseDialog exec:/etc/apache2/bin

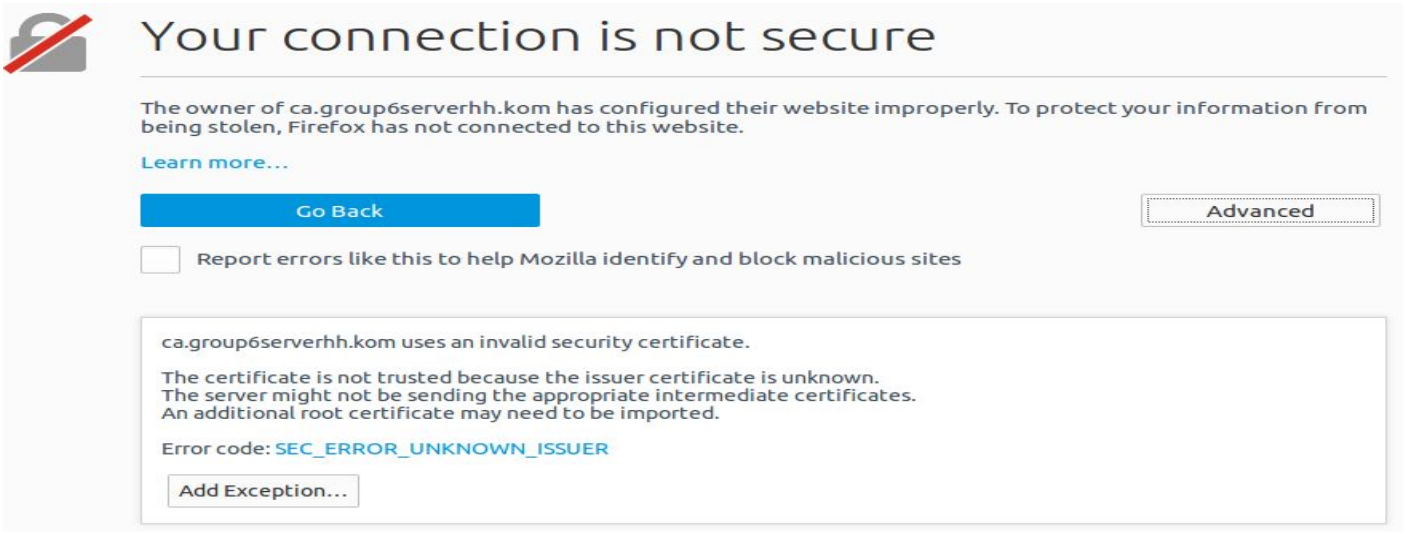
File /etc/apache2/bin:
#!/bin/sh
echo "Password"

Set file permissions as less as possible and required:
chmod 100 /etc/apache2/bin

Other method:
We decrypt the private key using the following command:
openssl rsa –in 01.pem -out 01.pem

And then we have to enter the passphrase.

Exercise 3.3:
At first when we make the ssl connection to the server we get the following warning:



Exercise 3.4:
In order to create a better user experience for our web-site, the user needs to install the CA root self-signed certificate by importing cacert.pem:

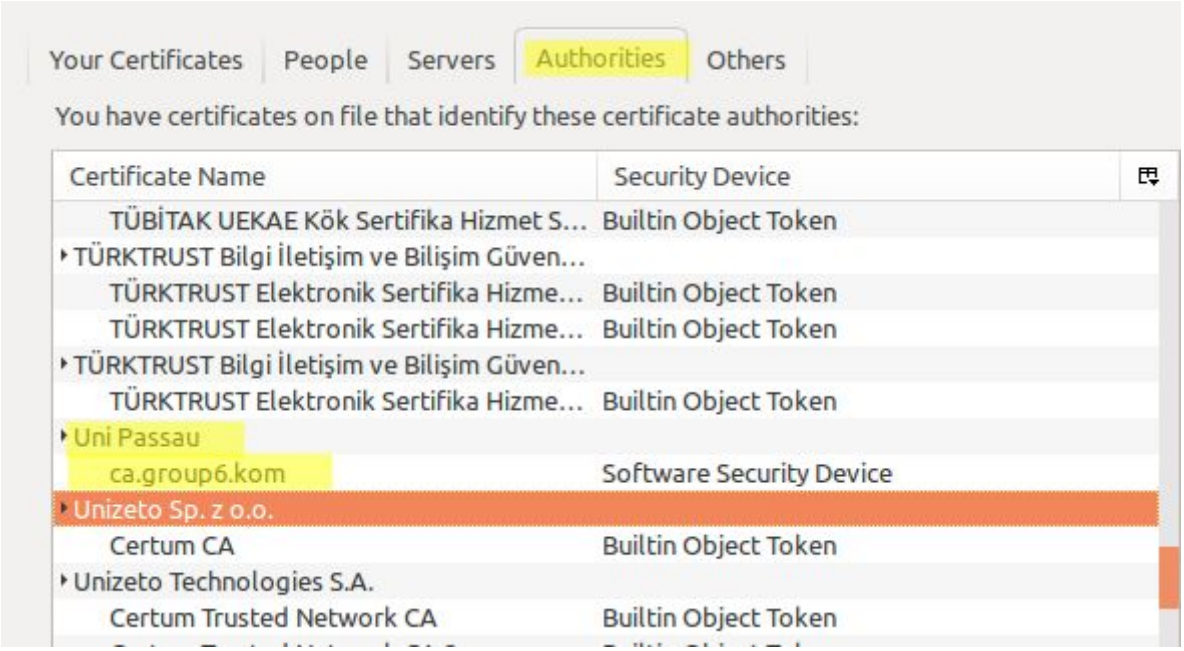


Figure: Installation of the authority.

And as a result our certificate gets accepted without UNKOWN_ISSUER error. The details of our certificate and the established connection (in the background) are shown in the following Figure:

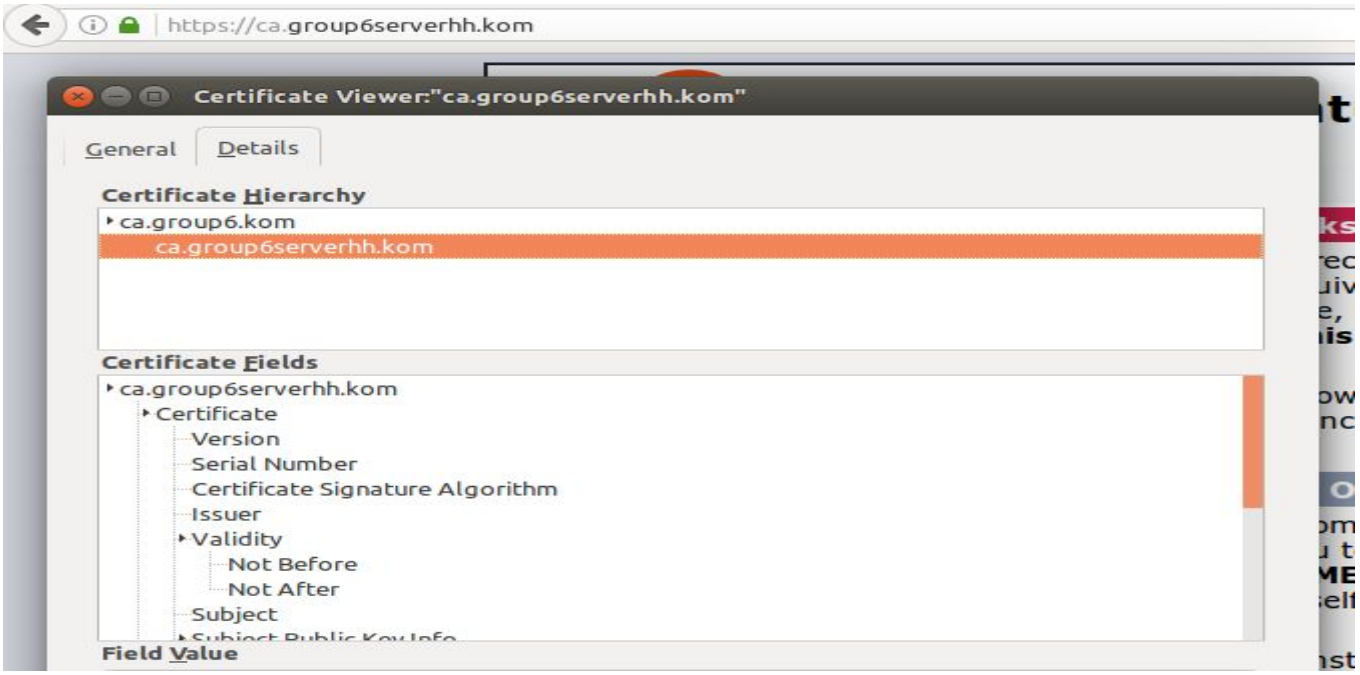


Figure: SSL connection to our web-site and displaying the certificate Hierarchy.

Exercise 4:

Prepare directory:
mkdir /etc/ssl/ca/intermediate

Create file/dir hierarchy:
cd /etc/ssl/ca/intermediate
mkdir certs crl csr newcerts private
chmod 700 private
touch index.txt
echo 1000 > serial
echo 1000 > /etc/ssl/ca/intermediate/crlnumber

Copy default config into intermediate folder:
cp /etc/ssl/openssl.cnf /etc/ssl/ca/intermediate

Change directory paths:
[CA_default]
dir = /root/ca/intermediate
private_key = $dir/private/intermediate.key.pem
certificate = $dir/certs/intermediate.cert.pem
crl = $dir/crl/intermediate.crl.pem
policy = policy_loose

Encrypt intermediate key with des3:
cd /root/ca
openssl genrsa -des3 -out intermediate/private/intermediate.key.pem 4096
chmod 400 intermediate/private/intermediate.key.pem

Create a CSR for signing our CA cert:
openssl req -config intermediate/openssl.cnf -new -des3 -key intermediate/private/intermediate.key.pem -out intermediate/csr/intermediate.csr.pem

When asked we provided the cert details:
Country code: DE
State: Bavaria
CN: ca.group6.kom

CSR was written to file *intermediate.csr.pem***.**

Open connection to Lab's Root CA by running:
telnet 192.168.0.101 9090

Copy paste content of file intermediate.csr.pem into telnet session.

After our CA cert was signed, we downloaded it at https://192.168.0.101/certs

As we did previously in exercise 3 we will generate a new request from the server-hh and then we will transfer to the subordinate CA to sign it:
Create new server request:
openssl req -key server-hh-private.pem -new -out server-hh-request.csr

Send server-hh-request.csr to CA by scp.

Sign the server request with the new certificate signed by our supervisors:
openssl ca -in server-hh-request.csr -config /etc/ssl/openssl.cnf

Send new created cert in /newcerts folder back to server-HH.

Created a chain file of Lab's Root CA and our group's CA:
cat intermediate/certs/ca.group6.crt ITSECSecurityLabROOTCA.crt > intermediate/certs/ca-chain.pem
chmod 444 intermediate/certs/ca-chain.cert.pem

Verify server certificate against chain by:
openssl verify -CAfile ca-chain.pem newcerts/SERVER_CERT_NAME.pem
Result: OK (see Image)

```
root@server-HH:/etc/ssl/CA# openssl verify -CAfile cachain.pem newcerts/02.pem
newcerts/02.pem: OK
root@server-HH:/etc/ssl/CA#
```

Exercise 4.1:
Firefox shows Unknown issuer error, because we have a new server certificate installed. Firefox is not trusting the new installed cert.

Exercise 4.2:
In order to fix the problem of connecting, we have to reconfigure Apache to know our CA chain.

Add chain file to Apache2 SSL vhosts file:
SSLCertificateFile /etc/apache2/ssl/ca.group6.kom.crt
SSLCertificateKeyFile /etc/apache2/ssl/private.pem
SSLCACertificateFile /etc/apache2/ssl/ca-chain.crt

We installed the CA lab root certificate ITSECSecurityLabROOTCA.crt in our browser.

Browsing to our page:
Firefox shows unknown issuer error. Cert details shows the working chain:

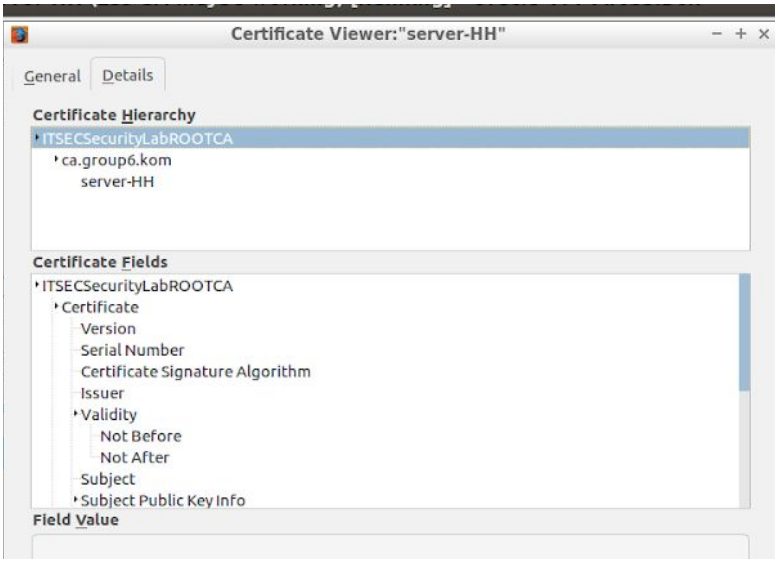


Figure: Certificate hierarchy.

We manually install our Group's CA by importing in the Firefox options. Further, we browse the server-HH page again. A connection was successful, although it should not be trusted, because the Root certificate of the chain has not been installed yet. We assume that Firefox trusts all imported CA's directly without validation of their root certificates.

Furthermore, we deleted our groups CA from Firefox and only installed the Lab's Root certificate. As desired, a connection was possible, even without our group's CA cert.

Exercise 5:

After creating the secret folder mkdir /var/www/html/secret and adding an index.html containing an image, we created the client certificate like the following:
First we created a folder in which we will store all the certificates and the keys of the users.
mkdir /etc/ssl/usercerts
Second we opened a terminal in that folder and we do the following:
Generating a user key:
openssl genrsa -des3 -out user.key 1024
Creating a user request:
openssl req -new -key user.key -out user.CSR
Sign it using our CA self-signed root certificate:
openssl ca -in user.CSR -out user.CRT
Finally we should convert it to P12 file, otherwise the browser won't recognise it and therefore it won't be imported sucessfully.
openssl pkcs12 -export -clcerts -in user.CRT -inkey user.KEY -out user.P12
Once this is done we go to server-hh and we add the what follows to the file /etc/apache/sites-enabled/default-ssl.conf:

```
<Location /secret>
        SSLVerifyClient require
        SSLVerifyDepth 2
</Location>
```

And Then from the client machine we install the user.P12 in mozilla firefox by: Preferences -> Privacy & Security -> Manage Certificates -> Your Certificates -> Import -> Choose file: user P12.

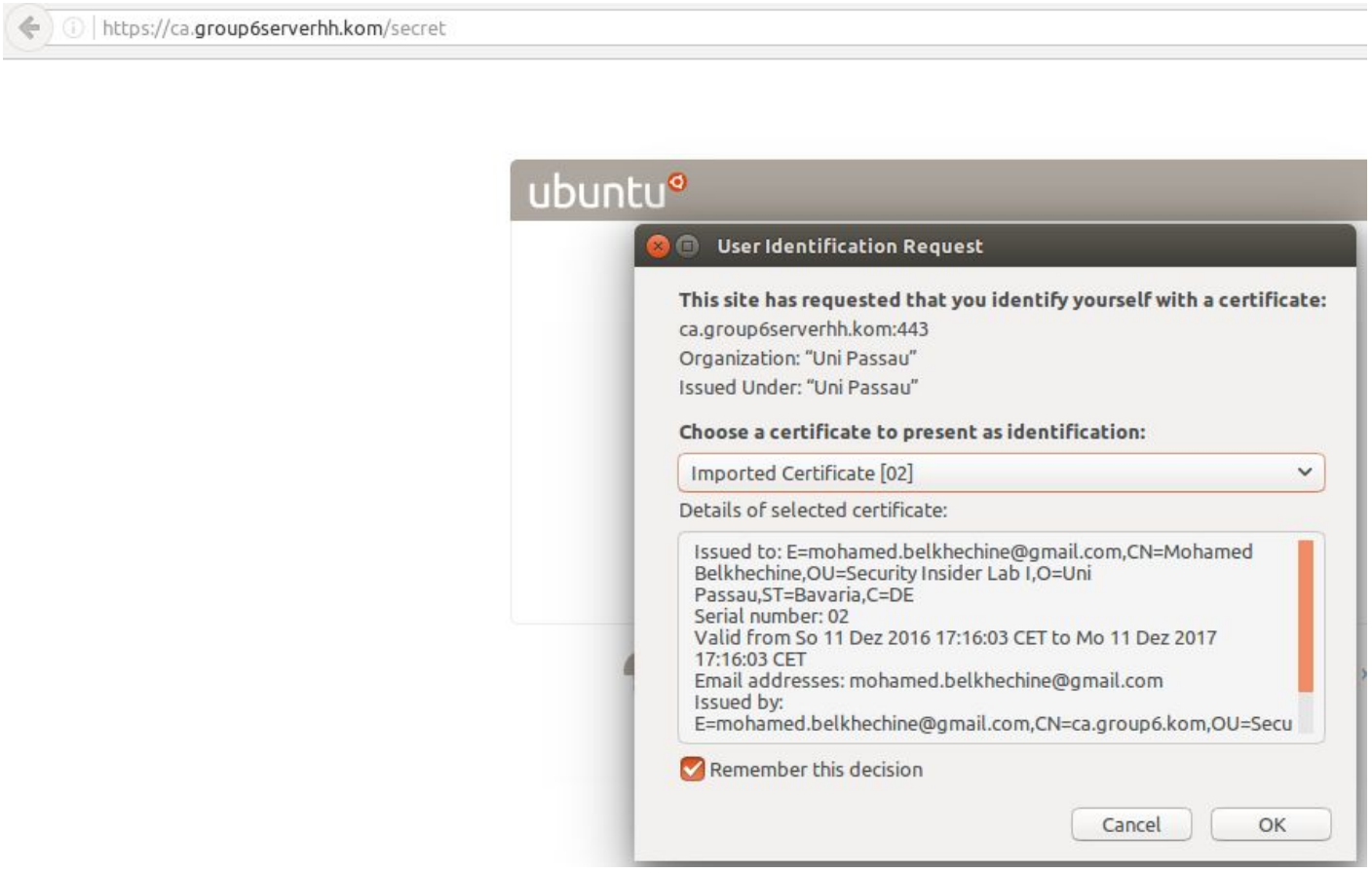


Figure: A popup shows up asking for the client certificate.

Once we have selected the right certificate, we will have access to the secret file.
We did the same steps with a group. The next figures shows how we dealt with signing their certificate.

```
# OpenSSL root CA configuration file.
# Copy to `/root/ca/openssl.cnf`.

[ ca ]
# `man ca`
default_ca = CA_default

[ CA_default ]
# Directory and file locations.
dir               = /root/CAdir
certs             = $dir
crl_dir           = $dir
new_certs_dir     = $dir/newcerts
database          = $dir/index.txt
serial            = $dir/serial
#RANDFILE          = $dir/private/.rand

# The root key and root certificate.
private_key       = $dir/caprivate.key
                    [ Read 133 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

Figure: Openssl configuration

```
root@kali:~/CAdir# openssl ca -config openssl.conf -in ~/test/aswin_05_kali1.csr
 -out ~/test/aswin_signed06.crt
Using configuration from openssl.conf
Enter pass phrase for /root/CAdir/caprivate.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 2 (0x2)
        Validity
            Not Before: Jan  9 20:37:52 2017 GMT
            Not After : Jan  7 20:37:52 2027 GMT
        Subject:
            countryName               = DE
            stateOrProvinceName       = BA
            organizationName          = UniPassau
            organizationalUnitName    = InsiderLab
            commonName                = ca.group05.kom
            emailAddress              = mail@ca.group05.kom
Certificate is to be certified until Jan  7 20:37:52 2027 GMT (3650 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
```

Figure: Signing other group's csr

```
root@kali:~/CAdir# cat index.txt
R       270107203752Z   170109205541Z   02      unknown /C=DE/ST=BA/O=UniPassau/
OU=InsiderLab/CN=ca.group05.kom/emailAddress=mail@ca.group05.kom
V       270109162533Z           03      unknown /C=DE/ST=BA/O=UniPassau/OU=Insid
erLab/CN=aswin@group05.com/emailAddress=mail@ca.group05.kom
root@kali:~/CAdir#
```

Figure: Checking if the certificate is valid or revoked.

Exercise 5.1:
Add a custom log entry for SSL queries to default_ssl.conf:
CustomLog "${APACHE_LOG_DIR}/ssl_request_log" "%t %h %{SSL_PROTOCOL}x%{SSL_CLIENT_M_SERIAL}x %{SSL_CLIENT_S_DN_CN}x \"%r\" %b"

```
  GNU nano 2.5.3                    File: /var/log/apache2/ssl_request_log

[14/Dec/2016:14:56:09 +0100] 132.231.164.207 TLSv1 04 Fabian Goettl "GET /secret/holder.min.js HTTP/1.1" 7973
[14/Dec/2016:14:56:09 +0100] 132.231.164.207 TLSv1 04 Fabian Goettl "GET /secret/js/bootstrap.min.js HTTP/1.1" 9833
[14/Dec/2016:14:56:09 +0100] 132.231.164.207 TLSv1 04 Fabian Goettl "GET /secret/js/jquery.min.js HTTP/1.1" 33760
```

Logs of the SSL protocol, certificate serial and CN.

Exercise 5.2:

Log handshake by following config:

```
<IfModule mod_ssl.c>
  ErrorLog /var/log/apache2/ssl_engine.log
  LogLevel debug
</IfModule>
```

Or we can access the file /var/log/apache2/error.log and check the lines that has [trace2].

E.g:
[Tue Jan 10 19:52:09.836037 2017] [ssl:trace3] [pid 20497] ssl_engine_kernel.c(1989): [client 192.168.179.32:38370] OpenSSL: Write: SSL negotiation finished successfully

Or we can fire up Wireshark at the moment of the handshake:

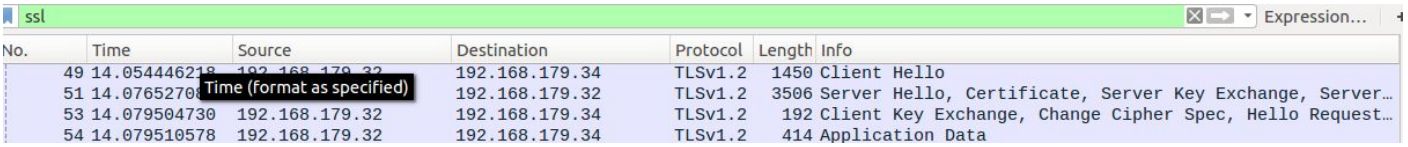


Figure: Handshake capture with Wireshark.

As per this article, the server presents its certificate first and then the client. In more details:

- Client sends CLIENT HELLO as described in the above image
- Upon receiving the CLIENT HELLO, if the server is configured for Client Certificate Authentication, it will send a list of Distinguished CA names & Client Certificate Request to the client as a part of the SERVER HELLO apart from other details depicted above.
- Upon receiving the Server Hello containing the Client Certificate request & list of Distinguished CA names, the client will perform the following steps:
    - The client uses the CA list available in the SERVER HELLO to determine the mutually trusted CA certificates.
    - The client will then determine the Client Certificates that have been issued by the mutually trusted Certification Authorities.
    - The client will then present the client certificate list to the user so that they can select a certificate to be sent to the user.

Exercise 6:

First of all we created a new folder in /etc/ssl having as name: crl and we executed the following commands to keep track of which user certificate has been revoked.
touch certindex
echo 01 > certserial
echo 01 > crlnumber

In order to revoke a user certificate we need to execute the following command:
openssl ca -config ../openssl.cnf -revoke ../usercerts/user.crt
Then we can generate the file crl.pem file using the following certificate:
openssl ca -config ../openssl.cnf -gencrl -out crl/crl.pem/

Once we have done that we edit the file **openssl.cnf,** CA_default section and we make sure that it points to our newly created crl folder crl/ and the crl.pem:
Crl_dir = $dir/crl
crl = $dir/crl/crl.pem

Finally, we need to edit the apache2 configuration file in server-hh. After transferring the file crl.pem to server-hh we add the following two lines:
SSLCARevocationFile /etc/apache2/ssl/crl.pem
SSLCARevocationCheck chain

Before revoking the certificate, the group had access to the secret folder.

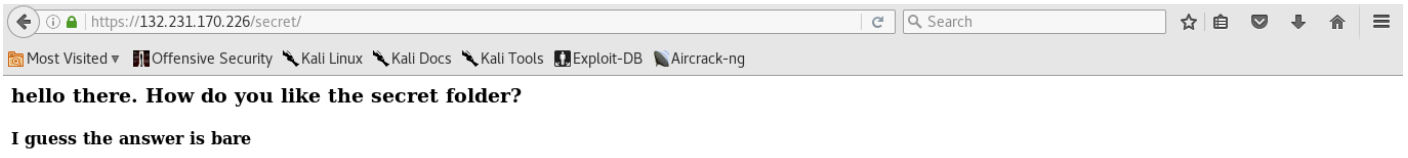


Figure: Secret folder.

When we try to access the secret folder hosted on server-hh and using the user.crt we get the following screen saying: **SSL_ERROR_REVOKED_CERT_ALERT**

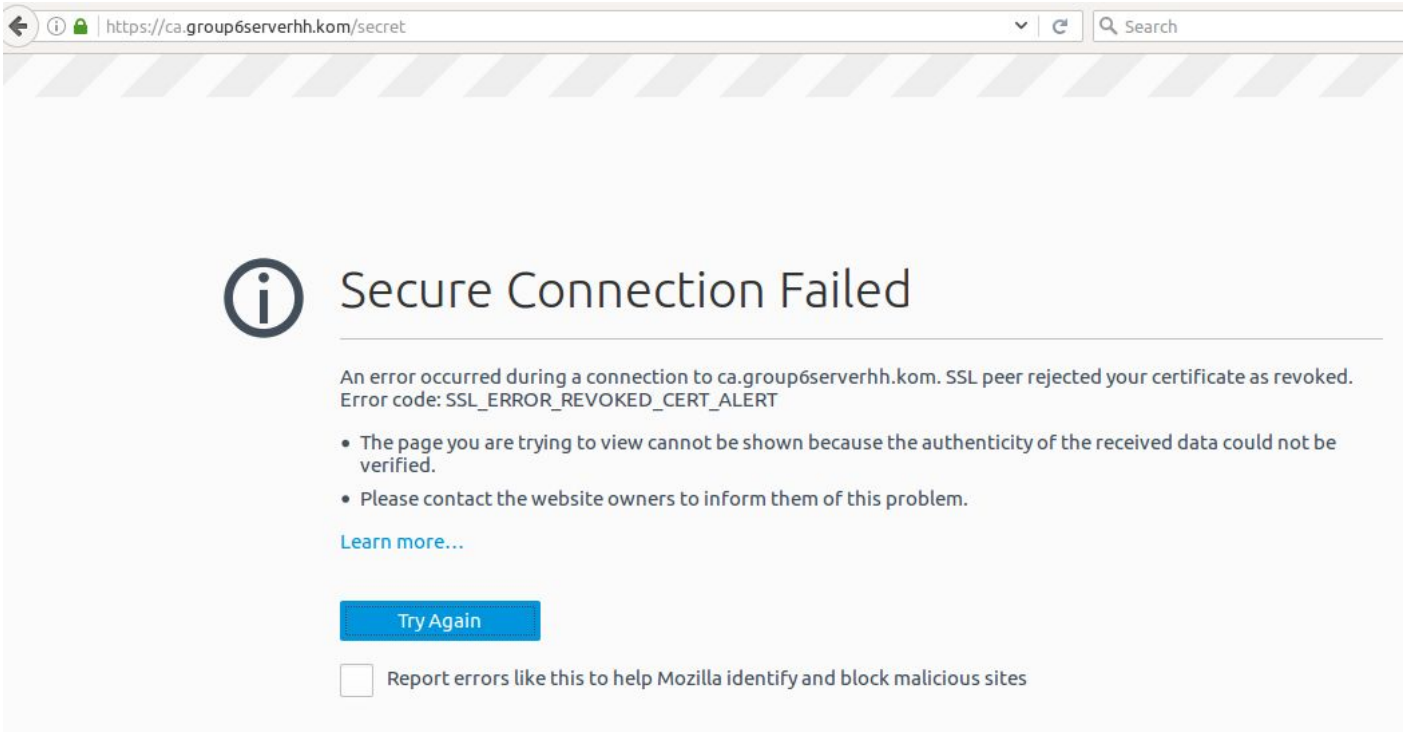


Figure: Denying access to clients having a revoked certificate

Exercise 6.1:
No, the checking is not done in real time. The apache server checks the validity period of the CRL. In order to show the basic details of a CRL such as its validity and and what certificate has been revoked we can make use of the following command:
openssl crl -in *crl_file* -noout -text

The server has to download the new revocation file from the CA machine. This can be automated by a script.

When a certificate status was changed, for example into *revoked*, the web server has to be reloaded:
service apache2 reload

Exercise 7:

Allow other CAs by creating a bundle of CA files:
cat ca.group6.crt ca.othergroup.crt > ca-bundle.crt

Add following line to Apache's SSL vhost config:
SSLCACertificateFile /etc/ssl/ca-bundle.crt
SSLVerifyDepth 2

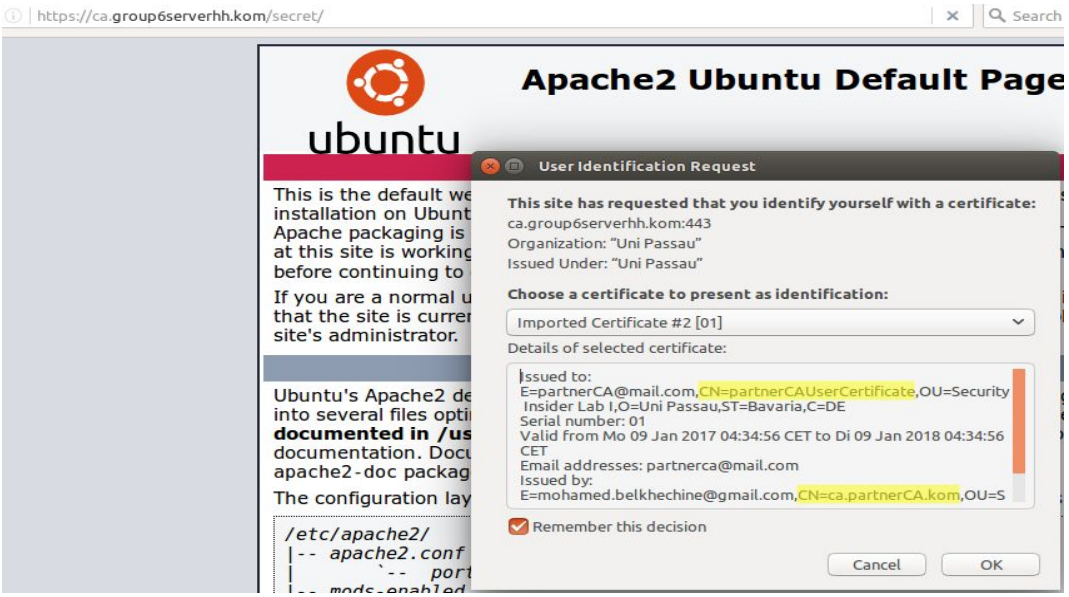


Figure: Requesting the access to the secret folder and providing the Partner CA certificate.

Exercise 7.1:
Our technique is not based on cross signing. Basically, instead of passing to the Apache server our CA root certificate only, we passed the partner root certificate with our CA root certificate.

Exercise 8:

First, we will issue an OCSP signing certificate to the OCSP server with the OCSP extension, otherwise signature verification will fail when a certificate is being checked:

openssl req -new -nodes -out auth.group6.kom.csr -keyout auth.group6.kom.key -extensions v3_OCSP
openssl ca -in auth.group6.kom.csr -out auth.group6.kom.crt -extensions v3_OCSP

Run OCSP server:
openssl ocsp -index /etc/ssl/CA/index.txt -port 8888 -rsigner auth.group6.kom.crt -rkey auth.group6.kom.key -CA /etc/ssl/CA/cacert.pem -text -out log.txt

Exercise 8.1:
Check if OCSP is working by creating and testing a dummy certificate:
openssl req -new -nodes -out dummy.group6.kom.csr -keyout dummy.group6.kom.key
openssl ca -in dummy.group6.kom.csr -out dummy2.group6.kom.crt

Check validity of dummy certificate by running a query:
openssl ocsp -CAfile /etc/ssl/CA/cacert.pem -issuer /etc/ssl/CA/cacert.pem -cert dummy.group6.kom.crt -url http://auth.group6.kom:8888 -resp_text

The server responded with:

```
1MEa1Fw8Abwq5LfsiZJW6U/WdVm8p93lDpKyigu7td6XGFWTJl8ILY0gSIWekI//
f20=
-----END CERTIFICATE-----
Response verify OK
dummy3.group6.kom.crt: good
        This Update: Jan 10 19:41:30 2017 GMT
root@server-HH:/etc/ssl#
```

Figure: Server Response.

Exercise 8.2:
Information that gets sent:

- Certificate: Public key
- Details of the certificate: CN, Mail, Address, Issuer, Serial Number,etc.
- Response verify OK or REVOKED

The additional information is sent, to check if we are really communication with the correct OCSP server and if we are verifying the correct certificate.

Exercise 8.3:
The OCSP request itself is not encrypted (http traffic), but we can check the validity of the server's certificate. This requires a CA certificate, which was downloaded from a trusted source. If the OCSP cert was issued by our trusted CA, we assume that we are communicating with the correct server.

Furthermore, the server has to sign the message in order to enable a verification on the client side.

Converting .crt to .pem:
openssl x509 -in dummy2.group6.kom.crt -out mycert.pem -outform PEM