Cybersecurity MSc: Applied Cryptography 2023-24

Practical Exercises: Certificate issuing, exploitation and revocation

September 14, 2023 Due date: no date

Changelog

- v1.0 - Initial version.

1 Introduction

This laboratory guide will focus in the setup of a Certification Authority (CA), on the issuing of a certificates for Web servers and on their revocation. Furthermore, we will show the relevance of the provisioning of complete certification chains for helping certificate validators to get complete chains.

For the execution of this guide we will use Linux and the following packages: `xca` and `apache2`.

2 Creation of a new certification hierarchy

Certification Authorities are vital for authentication of services across the Internet. They are considered to be trusted and this trust is inherited by the certificates they issue (sign). Although CAs commonly have a global reach and are to be globally trusted, this is not really a requirement and custom CAs can be created. If clients install the custom CA and trust it, all certificates issued by the CA will also be trusted, just like any other commercial, widely deployed root CA. This is useful for services which have a limited number of users accessing it.

In this step we will create a custom CA in order to generate signed certificates for a personal Web server. For this purpose we will rely on the `xca` software.

The first step is to launch the `xca` application and create a new database. Do not forget to specify a password!

2.1 Creation of the root CA

Afterwards, generate a new private key for the CA root key pair, named `CA root`. Usually, CA keys are considerably stronger that those of server certificates. Consider 4096 bits if possible.

Then create a certificate for the CA's root key pair by selecting *Certificates→New Certificate*. Do not forget to select the key you just created, as well as the *[default] CA* template (you must select it and then choose *Apply all*. Also, you must fill the identification data for the CA, define the key usage extension as critical and define the validity of the CA (a few years). Use as *Internal name* for the certificate the same name of its private key, `CA root`.

2.2 Creation of an intermediate CA

After having the root of the CA properly configured, we can generate other certificates of its hierarchy; we will start by an intermediate CA.

In a normal scenario, a key pair owner would create the CSR for geting a certificate to its public component, a Registration Authority (RA) would validate the CSR (namely, validate the identification items on it and the requested key usages and certificate usage restrictions), and a CA would issue the certificate upon and accepted CSR.

In this experiment we will use the same `xca` instance (database) to hold the credentials of the intermediate CA. Thus, we will we use it to create a new key pair, create a CSR for it, approve the CSR (this step is just ignored) and issue the certificate (signed by the root private key).

Go to the tab *Certificate Signing Requests* and click *New Request*. Apply again the *[Default] CA* template and fill the remaining fields. Do not forget to generate a new key pair in the *Subject* tab (call it `Sub CA`) and define the key usage extension as critical. Also use the key name in the *Internal name* field of the subject identification.

With the CSR, the certificate can be finally generated by signing the CSR with the `CA root` key. Right-click on the CSR and select *sign*. Please note that the CSR signing credentials are defined in the initial tab, where you select the certificate that corresponds to the private key that will be used to issue the new certificate. The pinpointing of the signing certificate is required to extract the identification of the issuing entity.

Verify in the *Certificates* tab that the new certificate is hierarchically below the `CA root` one.

3 Certificate issuing for a Web server

After having the intermediate CA up and running, we can generate a certificate for our server. For that, we need first to create a Certificate Signing Requests (CSR), and sign that request with the intermediate CA just created (`Sub CA`). Go to the tab *Certificate Signing Requests* and click *New Request*. This time, apply the *TLS_server* template and fill the remaining fields. Use `localhost` as the common name, generate a new key and define the key usage extension as critical. You can select the *TLS Web Server Authentication extended key usage* (also as critical).

With the CSR, the certificate can be finally generated by signing the CSR with the `Sub CA` certificate. Right-click on the CSR and select *sign*. Please make sure you sign the CSR with the `Sub CA` certificate!

Verify in the *Certificates* tab that the new certificate is hierarchically below the `Sub CA` one.

4 Export of keys and certificates

The keys and certificates produced with `xca` need to be used by other tools, namely an `apache2` HTTP server and client browsers. For that, we will export selected keys and certificates produced with `xca`.

There are many formats to export keys and certificates.

For certificates, the most convenient one is PEM, because it allows sets of certificates to be concatenated in a single file for being uploaded by some tool, which can be more convenient that using directories with independent files (though in a single file an observer of the certificates' encoded data looses all information about which refers to whom).

For private keys, they can be exported isolated or bundled with a certificate, or a certificate chain. They can also be exported in cleartext or protected by an encryption layer (with a key derived from a password).

In this experiment, the private key that needs to be exported is the one of the HTTPS server, and the `apache2` configuration requires private keys to be provided unprotected. Regarding the certificates, we will do some experiments with certification chains, thus we want all of them to be individually exported to independent files.

The location of the files created with the exports depends on their use. Since we are going to use them in an `apache2` server, we can store all of them, except the root certificate, in the `/etc/apache2/ssl.crt` directory (it should exist only after the installation of the `apache2` package; if nonexistent, create it). The root certificate can be exported to the directory were Linux usually keeps the system's trusted certification roots, the `/etc/ssl/certs` directory.

Therefore, go to the *Private Keys* tab, select the HTTPS server private key and export it as an unencrypted private key in text format (*PEM private (*.pem)*). Then, go to the *Certificates* tab, and for each certificate, select it, and export them in PEM text format with headers (*PEM (*.crt)*).

5 Import of certificates

Certificate validation processes are grounded by the trust is small sets of public keys, which define (trusted) root certification authorities. Since we have created a new one, and it was used to build a certification chain down to a server's certificate, we need the server's clients to trust on that root in order to trust on the server's certificate.

In this experiment, the browser that will contact the HTTPS server will have to install our self-certified root certificate as a trusted authority. Thus, install the certificate of our root CA as a trusted authority in the browser's list of authorities, and do not forget to allow trusting on it to authenticate Web servers. You can do this with Firefox, but other browsers should likewise allow it. In Firefox, go to *Preferences*, choose the *Privacy & Security* tab and close to the bottom select *View Certificates*. The list of trusted authorities is presented in the *Authorities* tab.

6 Setup of an HTTPS server

For the server we will use the `apache2`, which must also be installed. Afterwards, the `SSL` module must be enabled by issuing:

```
a2enmod ssl
```

In the folder `/etc/apache2/sites-available` there is a file named `default-ssl.conf`. Copy this file to `/etc/apache2/sites-enabled` as `experiment.conf`. This file defines the SSL configuration that the server will use, and must be edited in order to consider the cryptographic material just created. Be sure to modify the following variables:

- `SSLCertificateFile`: This should refer a PEM file containing the server certificate.
- `SSLCertificateKeyFile`: This should refer a PEM file containing the server private key.
- `SSLCertificateChainFile`: This file should contain the certification hierarchy of the server's certificate, excluding the root certificate.

  In a first assessment, set this variable with its correct value but leave it commented (line starting with `#`). We will see a difference upon its activation.

Edit all required variables and restart the server by executing (as administrator):

```
service apache2 restart
```

With a Firefox browser access the server by typing the URL `https://localhost`. You should get an error, since the server's certificate cannot be validated by the browser. This happens because the intermediate CA certificate is not known by the browser, and is not also being provided buy the HTTPS server. Therefore, the browser cannot build a trust path from the server's certificate up to a public key belonging to a trusted authority.

Uncomment the variable `SSLCertificateChainFile` in the `apache2` configuration file and restart the server. After doing this, you should be able to access the URL above referred. Now everything is correct, there should be no warning or error, and the page should be secure. The practical result is that the server is authenticated and the connection is secure. The browser accepts the server's certificate because it was validated up to a trusted but public key. If the `common-name` field of the certificate is different from `localhost`, the browser would show a warning message.

Go to the browsers' certificates and edit the trust on our root certificate. In particular, remove the trust on it relatively to Web servers. Restart the server (to delete cached TLS sessions) and repeat the access to the same URL. This time, you should get an error, as the browser not longer can trust the server's certificate.

7 Revocation of certificates

With `xca` create a new certificate for the `Sub CA` (you can reuse the previous CSR), but this time add *X509v3 CRL Distribution Points* in the *Extensions* tab. Usually CRL distribution points are given by URLs that belong to CA's Web servers, but for the purpose of demonstration we can use and Web server. Therefore, add an URL such as `http://localhost/RootCA_CRL.pem` for distributing the `CA root` CRL using our Web server.

Export the new certificate, install it in the `apache2` server and restart it. Refresh the browser access to the HTTPS Web page and confirm that the intermediate certificate of the server's certification chain contains the CRL distribution point. See what happens with the fact that the browser cannot access the referred CRL. Try to force the browser to verify CRLs (or OCSP servers).

Now go to the *Certificates* tab, select the `Sub CA` certificate and revoke it, selecting *CA Compromise* as reason. Then, go to the *Revocation lists* tab, select *New CRL* and select the `CA root` (the revocation of the `Sub CA` certificate must be published by its issuer). You can use the default values provided for the CRL. Export the CRL with the name given above, and store it in the directory /var/www/html.

Restart the browser and repeat the access to the HTTPS Web page. You should get an error due to the revocation referred in the now-available CRL

8 References

- XCA, `http://sourceforge.net/projects/xca`
- Apache2 ModSSL, `http://httpd.apache.org/docs/2.4/mod/mod_ssl.html`