SIO Lab
Transport Layer Security
version 1.0

Authors: João Paulo Barraca, Hélder Gomes, André Zúquete, Alfredo Matos

Version log:

- 1.0: Initial version

1 Introduction

Secure access to services is frequently (but not always!) conditioned to an enrollment process. The provider identity must be asserted to protect the user, typically done through public key cryptography and certificates. The user identity must also be determined, before resources can be used. There are many ways of determining the identity of a user, ranging from simple usernames and passwords (shared secrets), to biometrics and X.509 certificates (to name a few).

In this context, after ensuring are contacting the intended server through public key certificates, smartcards can be used for remote authentication of their owners.

This laboratory guide will explore how a server can be configured to tunnel HTTP requests inside a secure session, making use of the Transport Layer Security (TLS) protocol. This will be achieved by creating a Certification Authority, which will be used to create a certified public key, that can be used by web servers, and verified by clients.

It will also explore how an HTTP server can be configured to request users to authenticate themselves through the use of a smartcard, such as the Portuguese Citizen Card (although any other PKCS#11 compatible certificate would work), as means to provide mutual authentication.

Note: It is highly recommended the use of the virtual machine that was previously provided, alongside the created containers.

2 Creating a Secure Server

2.1 Create a webserver

```
# Create the server container
host:~$ lxc launch images:debian/bookworm webserver
```

" This server should be reachable by a hostname, which can be configure in hosts file: `/etc/hosts`. First, find the ip address of the container by reading it from `lxc list` or `lxc info server`.

Then, add it to the hosts file `/etc/hosts`:

```
$ sudo nano /etc/hosts

  SERVER_IP webserver
```

To make sure it is resolving properly issue a ping command:

```
ping webserver
```

After the webserver is configured, the container will be reachable by opening a browser and pointing it to `http://webserver` or `https://webserver`.

2.2 Creating a Server Cerficate

2.2.1 Creating a Certification Authority

Certification Authorities (CA) are vital for the authentication of services across the Internet. They can be considered as trusted roots if they are preinstalled in browsers or in the operating systems, seing their trust being propagated to the certificates that they sign. While many CAs are trusted at a global scale, this requirement is not globally valid in all circumstances. The existence of CAs valid for a specific scope, shared by a set of services and clients is perfectly plausible and may be secure.

From a trust perspective, the only requirement is that clients trust the policies of a CA and its PKI. If a small group of clients trusts a specific CA, locally managed, they can install the CA as a trusted root, and they will also trust the certificates it emits (signs).

In this stage the objective is to create a local CA, with the purpose of certifying a local set of services, and in particular the HTTP server to be created. For that matter we will use an application named XCA, which is commonly available in the package sources of mainstream applications

```
$ apt install xca
```

After starting the application, the first step is to create a new database, which should be protected by a password. Using a password allows to secure the database by encrypting its content. In particular, it will encrypt the CA key material, which includes the CA private key.

Generate a new key to be used by the CA when signing new certificates and name it `CAKey`. It is usual to assume that CAs are long lived (e.g., 20 years), therefore keys to be used by CAs should be robust to future attacks, and have high complexity. In practice this means that CA keys should be of 4096 bits.

Then create a certificate for the CA, using the key that was previously created. Select the options *Certificates* → *New certificate*. Apply the template for a CA (*CA template*) by selecting it and pressing *Apply*. You should also fill in the information related to the CA, which includes its *Subject* and the definition of its validity. Normal service certificates have a validity of a few years (e.g., 1 to 3). CA certificates have extended validities in order of 10 to 20 years.

After creating the CA certificate, it is possible to emit new certificates for services. This is achieved by creating a Certificate Signing Request (CSR), which is signed by the CA, resulting in the emission of a certificate. Each certificate should have a different key, but they must be signed by the CA private key.

2.2.2 Creation of a certificate for an HTTPS server

HTTPS servers process standard HTTP requests, and emit responses using this protocol, but they encapsulate messages in secure tunnels by making use of the TLS specification.

For this purpose it is required the use of a technology that enables clients to validate the identity of a given server, correlating the server with the URL that is presented in a browser. That is: it is important to assure the identity of a server, so that users are sure that they are exchanging messages with the correct system, and not with a rogue server deployed by an attacker. The role of the CA is to validate the CSR, but authenticating that the URL associated with the certificate (through the *Common Name* field) is actually owned by the entity requesting the certificate.

Access the tab *Certificate signing requests* and select *New Request*. In this situation, choose the template named *TLS_server* for the new certificate.

In the tab *Subject* fill the identification information of the server. Take care with the value of the *Common Name* as it is used to match the certificate with the server URL. For the purpose of our guide, it should be `webserver` as the server will be accessed through the URL `http://webserver`.

In this tab, specify that you want to use a new pair of asymmetric keys for thee certificate (select *Generate a new key*).

After all fields have the necessary values, the Certificate Signing Request can be issued. Afterwards sign the CSR using the CA certificate. **Take care of signing the CSR with the correct key.**

2.2.3 Exporting the server certificate

HTTPS servers use cryptographic material that is stored in PEM files, somewhere in the filesystem. In this case it consists of the private key and the X.509 certificate. If the server certificate was signed by an Intermediate CA, the file containing the server certificate should also contain the certificate of the intermediate CAs.

Export the **Server** Certificate to a file named `cert.pem` and the private key to a file named `privkey.pem`. When exporting the private key, remember that keys should be protected by some mechanism such as a password. Setting a password will effectively cipher the private key, preventing it from being exposed to other users.

Move the files to the path `/etc/ssl/private` in the newly created container.

```
# Create the server container
vm:~$ lxc push cert.pem /etc/ssl/private/
vm:~$ lxc push privkey.pem /etc/ssl/private/
```

2.3 HTTPS server setup

To implement a secure HTTPS server we will rely on the `nginx` software, which is commonly available in the Linux distributions package repositories.

```
# Log into the server
vm:~$ lxc shell server
# install nginx
server:~$ apt install nginx
```

The configuration of this server can be found in the `/etc/nginx` folder, and the configuration of the domain to be exposed by HTTP in the path `/etc/nginx/sites-available`.

There are several configurations, inside the `server` block, that must be changed in order to implement our purpose. The following snippet contains the relevant lines:

```
  listen 443 ssl;

  server_name _;
  ssl_certificate /etc/ssl/private/cert.pem;
  ssl_certificate_key /etc/ssl/private/privkey.pem;
  ssl_protocols TLSv1.2;
```

After these changes, put the file in `/etc/nginx/sites-enabled`, validate the configuration by running `nginx -t`, and then restart the server by running `service nginx restart`.

If everything is properly configured, you should be able to access https://webserver. It is expected that you are presented with an error stating that the identity of the server cannot be determined.

3 Configuring the client

The problem found in the previous section is due to the fact that the browser receives a certificate signed by an unknown certification authority. This happens because the certificate from our CA was not imported to the database of trusted roots.

Therefore we must import a custom CA into the browser. To do this, start by accessing XCA, and export the **CA** Certificate to a file named `ca.pem`. **DO NOT** export the private key.

Start an Internet Browser and drag the file into the browser. This should prompt a dialog to confirm the import of the certificate as a trusted root.

After these steps, access https://webserver and then https://webserver. Can you explain the results obtained?

Take notice that after this step, users can validate the identity of the server. However, the server is unable to authenticate users.

4 Authentication of Citizens through the Citizen Card

In this section, the objective is to implement mutual authentication, so that both the server and the client properly authenticate the identity of each other.

The first step is to obtain all intermediate certificates used by a citizen card, in PEM format. They are available in the course web page, and can also be downloaded directly from the PKI webpage[1].

Once the certificates are obtained, use `cat` to put all certificates in a single file (`PTEID.pem`), and place this file in `/etc/ssl/certs`.

Then, we can instruct the web server to require users to present a valid certificate, signed by a specific CA. Edit the `nginx` server definition file and add the following directives:

```
ssl_client_certificate /etc/ssl/certs/PTEID.pem;
ssl_verify_client on;
ssl_verify_depth 10;
```

Restart the `nginx` server and access https://webserver using the web browser. Can you explain what happens?

The result is that the web server will require users to present a valid certificate, signed by the a CA present in the `PTEID.pem` file. However, the browser has no way of knowing what certificate to provide, or how to do it. Hence, the access is denied.

You can use `Wireshark` to observe the dialog between the browser and the server, and check the certificates being provided (and the ones not actually being provided).

4.1 Add the PT Citizen Card to the browser

In order for the browser to use the Portuguese Citizen Card, it is required to add the secure module to the browser. This is achieved by loading the proper library that exposes the appropriate PKCS#11 interface.

The instructions are specific for each browser. In `Firefox`, access *Preferences->Privacy & Security->Security Devices*, and then choose `Load`. In the popup that is presented, fill in the following information:

- **Module Name**: `PTEID`
- **Module filename**: `/usr/local/lib/libpteidpkcs11.so`

Then press `OK`. If everything is working properly, you should see a new module loaded. Also, in the previous configuration panel, accessing `View Certificates`, two new certificates should be present in the tab `Your Certificates`.

Repeat accessing https://webserver with a Citizen Card inserted. You should be presented with a popup requesting the **AUTHENTICATION PIN**, and if the correct value is presented the server will allow access to the page.

Once again, you can use `Wireshark` to inspect the traffic exchanged between the browser and the web server. In particular, you should be able to see the message requesting the user to be authenticated and the certificates being exchanged.

[1] https://pki.cartaodecidadao.pt/publico/certificado/cc_ec_cidadao_autenticacao/EC de Autenticacao do Cartao de Cidadao XXXX.cer Where the part `XXXX` should have a value in the form `0001`, `0002`, etc...

4.2 Exploiting authentication data in applications

It is important to mandate users to be authenticated when they access a server. But it is also important to exploit this fact and enforce an access control policy. That is: while the current configuration requires users to present a valid certificate, **ANY** Authentication Certificate within its validity will be allowed to access.

Because, authentication information is exposed to applications, custom code can validate the information present and further restrict access to a much more restricted set of citizens.

In this example we will be using PHP. Go to `/var/www/html` and create a PHP file, named `index.php` with the following content:

```
<html>
<head>
    <meta charset="UTF-8">
</head>
<body>
    <pre>
    <?php print_r($_SERVER); ?>
    </pre>
</body>
</html>
```

Add support for PHP by installing the package `php-fpm`, and adding the following structure to the `nginx` server configuration file:

```
index index.php;
root /var/www/html;

location ~* \.php$ {
    fastcgi_pass unix:/run/php/php7.2-fpm.sock;
    include         fastcgi_params;

    fastcgi_param   USER_CERTIFICATE $ssl_client_escaped_cert;
    fastcgi_param   DN $ssl_client_s_dn;
    fastcgi_param   VERIFIED $ssl_client_verify;

    fastcgi_param   SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param   SCRIPT_NAME $fastcgi_script_name;
}
```

Restart both `nginx` and the `php7.2-fpm` service.

Once again, access https://webserver and check the result obtained. You should see many server variables, which will include the indication that the user was authenticated, as well as the authentication data provided.

Change the example and restrict the access to the page so that only one user is able to access it. For the remaining users (even if correctly authenticated), present an error message.

5 Bibliography

- PT Citizen Card
- EC Authentication Certificates
- EC Citizen Card Certificates (Test)
- EC Citizen Card Certificates
- EC State
- NGINX SSL
- NGINX PHP
- XCA