Rapport TP2 RS40

Alexandre BARTHELME

Mai 2023

Sommaire

1 Introduction

Ce deuxième TP est orienté sur la sécurité web et sur le fait que le flux réseau soit crypté. Il consiste à passer un serveur web d'une connexion HTTP à HTTPS. Pour le réaliser, nous devons configurer le serveur mais surtout la génération des différents fichiers .pem permettant de créer un certificat, signé par un autorité de certification "factice".

2 Avant modification

Au début, le serveur se lance et fonctionne uniquement en HTTP. Par conséquent le mot de passe défini dans le fichier build.py se retrouve visible non seulement sur la page web, mais également en clair dans le traffic réseau.
Celui ci est visible via Wireshark, lorsqu'on observe notre carte locale (loopback) on peut observer le traffic HTTP et constaté la requête de la page, ainsi que la réponse (Requête 'OK') qui contient dans l'encapsulation HTTP le text en clair et donc le mot de passe :" Mot de passe personnel modifié"

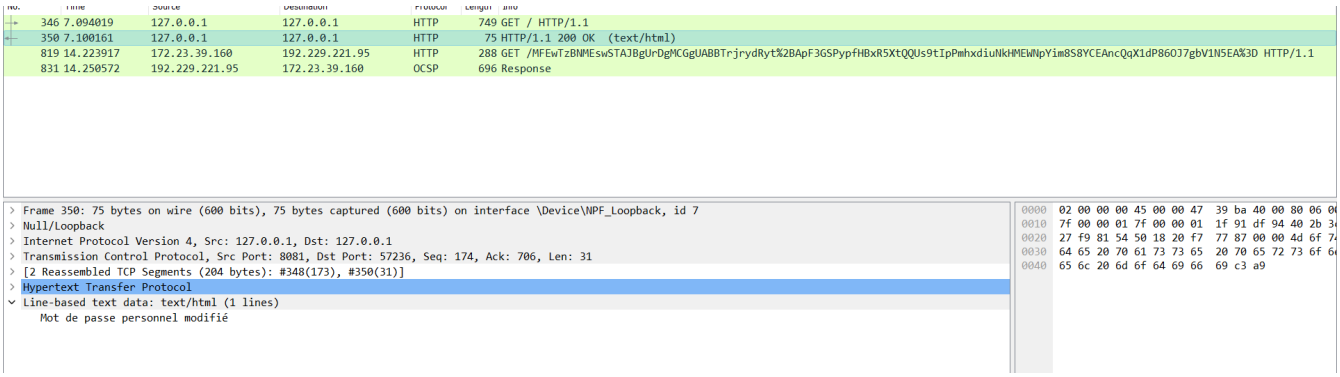

FIGURE 1 – Flux HTTP visible

3 Authorité de Certification

3.1 Principe

La CA est celle qui va éditer et signer un certificat qui par la suite va être reconnu par un navigateur web. Ce certificat sera considéré comme "sûr" si la CA est reconnue. Dans notre cas, la CA dispose de clé privé et public et d'une fonction qui permet de signer les requêtes de signature qu'elle reçoit. Elle a également son propre mot de passe pour la clé privée.

3.2 Code fichier ca/core.py

Les différentes fonctions à completer demandes les informations suivantes :
- *generate_private_key* : le fichier et le mot de passe CA

- *generate_public_key* : le fichier , le fichier de clé public et la config En ce basant sur les informations fournies et sur les variables déjà existantes, on peut coder les fonctions suivantes :

```
    self._private_key = generate_private_key(private_key_filename, password)

    self._public_key = generate_public_key(self._private_key,
   public_key_filename, config)
```

Les autres informations sur l'utilisation de la CA seront données dans la partie sur le fichier "build.py"

4 Génération des certificats

4.1 Serveur

Pour la création des certificats, on modifie également le fichier core du serveur, qui contient un fonction importante : $generate_csr$ qui constitue une "requête" pour la génération des certificats. Cette fonction demande la clé privé du serveur, nécessaire ensuite pour l'autorité de certification, pour emettre les certificats. Concrètement , voici les deux fonctions (dont l'une similaire à la CA) :

```
    self._private_key = generate_private_key(private_key_filename, password)
    self._csr = generate_csr(self._private_key,csr_filename,config)
```

4.2 Génération build.py

Enfin, pour générer nos certificats et toute "l'infrastructure", notre programme dispose d'un fichier *build.py* incluant toutes les fonctions d'instantiations des éléments : - Une CA - Un serveur - Un certificat voici la configuration personnalisée du build :

```

RESOURCES_DIR = "resources/"
CA_PRIVATE_KEY_FILENAME = RESOURCES_DIR + "ca-private-key.pem"
CA_PUBLIC_KEY_FILENAME = RESOURCES_DIR + "ca-public-key.pem"
SERVER_PRIVATE_KEY_FILENAME = RESOURCES_DIR + "server-private-key.pem"
SERVER_CSR_FILENAME = RESOURCES_DIR + "server-csr.pem"
SERVER_PUBLIC_KEY_FILENAME = RESOURCES_DIR + "server-public-key.pem"
CA_PASSWORD = "passwordca"
SERVER_PASSWORD = "passwordsrv"

CA_CONFIGURATION = Configuration("FR", "Territoire de Belfort", "Sevenans", "
   UTBM_CA", "localhost")
SERVER_CONFIGURATION = Configuration("FR", "Territoire de Belfort", "Sevenans",
    "UTBM_SER", "localhost")

# Cr ation de l'autorit  de certification
certificate_authority = CertificateAuthority(CA_CONFIGURATION,CA_PASSWORD,
   CA_PRIVATE_KEY_FILENAME, CA_PUBLIC_KEY_FILENAME)
    # regardez en haut et ca/core.py

# Cr ation du server
server =Server(SERVER_CONFIGURATION, SERVER_PASSWORD,
   SERVER_PRIVATE_KEY_FILENAME, SERVER_CSR_FILENAME)
    # regardez en haut et server/core.py

```

```
# Signature du certificat par l'autorit  de certification
signed_certificate = certificate_authority.sign(server.get_csr(),
    SERVER_PUBLIC_KEY_FILENAME)


#impression des certificats    compl ter regardez #print_pems

print("impression des certificats ...")
print("CA public key filename :")
ppems.print_pems(CA_PUBLIC_KEY_FILENAME)
```

Lors de l'éxecution du build, il va générer les différents fichiers .PEM, dont le "ca-public-key.pem" demandé : Les autres informations de la CA se trouvant juste au dessus.

```
-----BEGIN CERTIFICATE-----
MIIDbTCCAlWgAwIBAgIUaSJ6xEbCB/fOQfYWA4tQ589aI2gwDQYJKoZIhvcNAQEL
BQAwZjELMAkGA1UEBhMCRlIxHjAcBgNVBAgMFVRlcnJpdG9pcmUgZGUgQmVsZm9y
dDERMA8GA1UEBwwIU2V2ZW5hbnMxEDAOBgNVBAoMB1VUQk1fQ0ExEjAQBgNVBAMM
CWxvY2FsaG9zdDAeFw0yMzA1MDkxNTU1MDdaFw0yMzA3MDgxNTU1MDdaMGYxCzAJ
BgNVBAYTAkZSMR4wHAYDVQQIDBVUZXJyaXRvaXJlIGRlIEJlbGZvcnQxETAPBgNV
BAcMCFNldmVuYW5zMRAwDgYDVQQKDAdVVEJNX0NBMRIwEAYDVQQDDAlsb2NhbGhv
c3QwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDURoc7r0d52b/WiJFr
lTWSZdC4zycKcvALbnOMal3mr9jqXmyGFrIqjyuCDot9pN4jOThXE7StUac8ikbG
+9S1rv8aKo6CGZteW+LxZB9DfZvA8nuCSvopETkB9fKWUNsvFiT1MAHoCcASBdS1
K8zQrC5j/Fw4u99JvmqqoGb31N9nrsST58EKnHLbSp+x+d4POEUVURO29Wez8Bzn
L9mvselK58rW0Jgc3OFHbaYXkrCErxRzWLgaPfqWlaqjKG4L/XVA4GF0wVGgl2hs
XPM+wXxrnRdUDXRnNxT0CkhfkZCNx9FrnZgBYd+i/5xPsKWcDifm/W+UDCuPWRsr
ATKtAgMBAAGjEzARMA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEB
AL6pga9/iGB3EUXLU9LR797bcgSF0WI3qVWwm/zrV9zW1Aws39Tqrsxy66MjZqy2
d/SycsghF8oMi95SfTZRMxlfVkyeiA1ZnsY1LxFmn4kY4FP55mET3A3ThPVsjYMJ
GK9V6wvTq2DfUUvjEgjb7sbZY9Cx+w+Cp5REvoFJB6ZJ1JJYpRZcC/KB5TM5KyLl
p4qOyaVa6OgApkXW8vYKaT5H677mRmJe3rJ0vSi/uSJMXwROFerNtIk7z+WyM9Kd
zLfHc/67aNJS7Wi3jLrbQHZyPcyZnWe/E6r1h8RkZlt3HqM++SW1XsGHWPRaMgNM
LK+wKc1SYhz7LYC+uxyfsf8=
-----END CERTIFICATE-----
```

5 Lancement du serveur en HTTPS

Pour finir, il faut juste demander au serveur de se lancer en HTTPS, en incluant un "contexte SSL" qui va utiliser les fichiers "server-public-key.pem" et "server-private-key.pem" :

```
        # HTTP version
    #app.run(debug=True, host="0.0.0.0", port=8081)
    # HTTPS version
    # A compl ter  : n cessit  de d placer les bons fichiers vers ce
   r pertoire
    context = ("server-public-key.pem", "server-private-key.pem")
    app.run(debug=True, host="0.0.0.0", port=8081, ssl_context=context)
```

En terme d'amélioration, on pourrait inclure une phase de connexion pour accéder à l'information ; ce qui nécessiterai une base de donnée et la création de compte par une interface php et/ou un formulaire HTML.

Finalement, en lançant le serveur et en accédant à la page, le navigateur nous avertira que la connexion n'est pas sécurisé, car il ne s'agit pas d'une autorité de certification reconnue par le navigateur. Si l'on souhaite accéder à la page il faudrait ajouter le certificat dans le navigateur et lui indiquer qu'il s'agit d'une autorité de "confiance".

Autre option déconseillée : forcer l'accès depuis un navigateur proposant cette option comme Firefox. Sinon, pour les navigateurs comme Chrome ou Edge, il sera impossible d'accéder à la page.

On peut consulter les détails du certificats dans les options de la page :

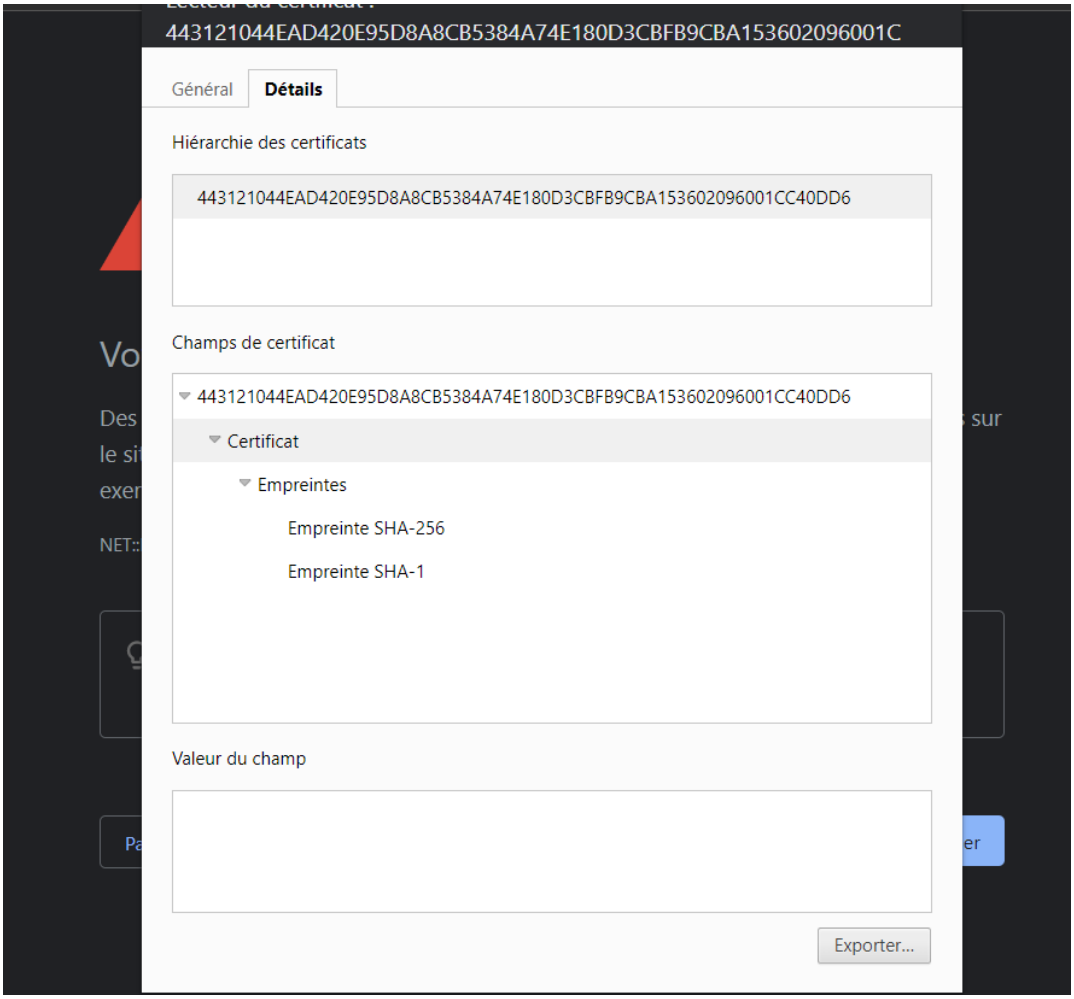

FIGURE 2 – Caption

6 Amélioration web

Par rapport à l'améliorer cité ci-dessus, j'ai donc rajouté deux pages HTML (une page "home" et "login") permettant à un utilisateur de se connecter avant de pouvoir voir le mot de passe affiché sur le site par défaut. Pour faciliter les choses et éviter une base de donnée, le mot de passe et le nom d'utilisateur sont codés en clair dans le code puis le mot de passe est hashé et modifié dans le fichier *run_serveur.py*.

6.1 Page HTML

La page "home" indique le mot de passe à l'utilisateur connecté :

```
    <!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1>Welcome, {{ id }}!</h1>
    <p> here is your secret message : {{ message }}</p>
```

```
</body>
</html>
```

La page de login est uniquement constitué d'un formulaire pour demander nom d'utilisateur et mot de passe :

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>

    {% if error %}
        <p>{{ error }}</p>
    {% endif %}

    <form method="POST" action="/">
        <label for="id">ID:</label>
        <input type="text" id="id" name="id" required><br><br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required><br><br>

        <input type="submit" value="Login">
    </form>
</body>
</html>
```

6.2 Fichier *run_server.py* et hashage

Pour le login, on défini le mot de passe en brut dans le fichier, puis on le hash un certain nombre de fois en ajoutant un "sel" avant le hashage. Pour que la connexion se fasse, on doit retrouver le même hash avec le mot de passe qu'entre l'utilisateur dans le formulaire. Enfin, si la connexion réussi, le site redirige vers la page "home" avec le mot de passe secret :

```

from flask import Flask, render_template, request
import hashlib

# d finir le message secret
SECRET_MESSAGE = "MDPnul" # A modifier #fait1
app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        # Si le formulaire est envoye
        id = request.form['id']
        password = request.form['password']
        #faire le hash + sel
        password = salt +password
        password = hash_password(password)
        # Verifier les informations
        if id == 'admin' and password == hashsalt:
```

```
            # connexion reussite, redirection vers home.html avec le "secret
    message"
            return render_template('home.html', id=id, message= SECRET_MESSAGE)
        else:
            # echec de la connexion
            error = 'Identifiants invalides'
            return render_template('login.html', error=error)
    else:
        # Afficher le formulaire s'il n'a pas   t   envoy
        return render_template('login.html')

def hash_password(password):
    #Hasher le mot de passe avec SHA-256 25x
    for i in range(25):
        password = hashlib.sha256(password.encode()).hexdigest()
    return password

#definir le mot de passe et le hash
password = "pass"
salt = "salt"
password = salt+password
hashsalt = hash_password(password)


if __name__ == "__main__":
    # HTTP version
    #app.run(debug=True, host="0.0.0.0", port=8081)
    # HTTPS version
    # A compl ter  : n cessit  de d placer les bons fichiers vers ce
    r pertoire
    context = ("server-public-key.pem", "server-private-key.pem")
    app.run(debug=True, host="0.0.0.0", port=8081, ssl_context=context)
```

7 Conclusion

Maintenant, le site (via Firefox uniquement) nous permet de se connecter pour voir un mot de passe secret en passant par une version HTTPS de la page. Nous avons configurer les différentes parties prenantes de la sécurité web (CA, serveur , page web) et vu les notions de bases des certificats.