Le 19/06/2020

Rapport de TP n°3

Club privé avec connexion https

UTBM – Département Informatique

MI44 Semestre P20

COMPOSITION DU GROUPE DE PROJET :

ABDELAZIZ Hassan

Mohamed OUEIDAT

Chargé du cours :

Abdeljalil **ABBAS-TURKI**

Sommaire

Introduction

Notre projet porte sur la sécurisation http en https à l'aide des méthodes vues en cours. Notre projet a été fait en python. Pour ce faire, le projet consiste à :

1- Créer une autorité de certification (ca) détenant une paire de clés (publique, privée) et un certificat (le tiers de confiance)
2- Générer un certificat du serveur signé par la ca. Ceci se déroule en deux étapes :
   a. Générer le certificat de la requête de certification (CSR : Certificate Signing Request)
   b. Faire signer le certificat par la ca.
3- Remplacer la connexion http par la connexion https

Objectif

Nous supposons un club privé qui distribue un mot de passe d'entrée aux adhérents une fois par mois. Ils utilisent ce mot de passe pour accéder au club. Les adhérents obtiennent le mot de passe en se connectant sur un lien URL secret. Le lien leur est communiqué lors de leur dernière rencontre physique. Actuellement, il s'agit simplement d'une connexion http, ce qui permet à toute personne observant le trafic de lire le mot de passe du club. L'objectif du projet est de remplacer la connexion http par une connexion https.

Qu'est-ce que OpenSSL ?

SSL a été à l'origine développé par Netscape. OpenSSL est un version libre du protocole SSL (Secure Sockets Layer version 1 et 2) et TLS (Transport Layer Security = SSL version 3). Il permet de crypter toutes les données échangées entre le client et le serveur de façon à ce que seul le serveur puisse décrypter ce qui vient du client et inversement. Un éventuel pirate ne peut pas, dans un temps raisonnable, décrypter les informations.

SSL sert de support :

- à SSH pour donner un telnet sécurisé et faire des tunnels cryptés simplement
- à HTTP pour sécurisé les sites de Web marchands
- à POP ou IMAP pour sécurisé la récupération de mail
- à tout autre protocole en clair afin de le sécuriser

SSL a trois buts :

- confidentialité des échanges grâce au cryptage symétrique
- intégrité des données grâce au fonction de hachage
- authentification des entités communiquantes grâce aux certificats

CA : Certificate Authorities : autorité de certification

Les autorités de certifications sont des organisations qui émettent les certificats moyennant paiement. Elles garantissent les certificats par leur signature ce qui permet d'être sûr du serveur auquel on se connecte (vente en ligne, banque...).

Contenu d'un certificat

Un certificat contient (suivant la normalisation X.509):

des informations sur le certificat :

- la version de la norme X.509
- le numéro de série du certificat (information purement administrative)
- l'algorithme de cryptage utilisé pour la signature de celui-ci
- la date de début de validité
- la date de fin de validité

des informations sur le demandeur/propriétaire :

- le nom (DN : Distinguished Name) du propriétaire qui comprend :
  - Nom du serveur (Common Name = CN) : nom du serveur sécurisé
  - Organisation ou Entreprise (Organization or Company = O) : nom de l'entreprise de la personne qui demande le certificat
  - Service (Organizational Unit = OU) : nom du service ou du département de l'organistation qui demande le certificat.
  - Ville (City/Locality = L) : nom de la ville dans lequel se trouve l'organisation
  - Département/Région/Etat (State/Province = ST) : nom de la province dans laquelle se trouve l'organisation
  - Pays (Country = C) : code ISO du pays dans lequel se trouve l'organisation (FR, BE, ...)
  - adresse email de l'administrateur

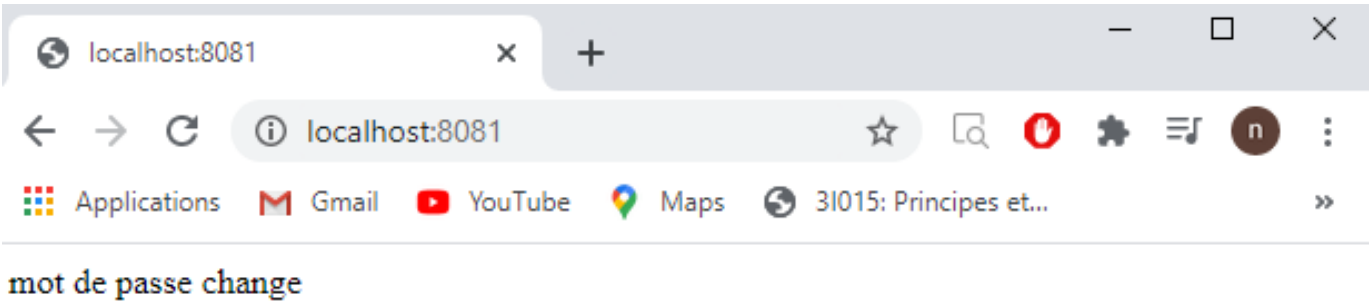
Les étapes de configuration :

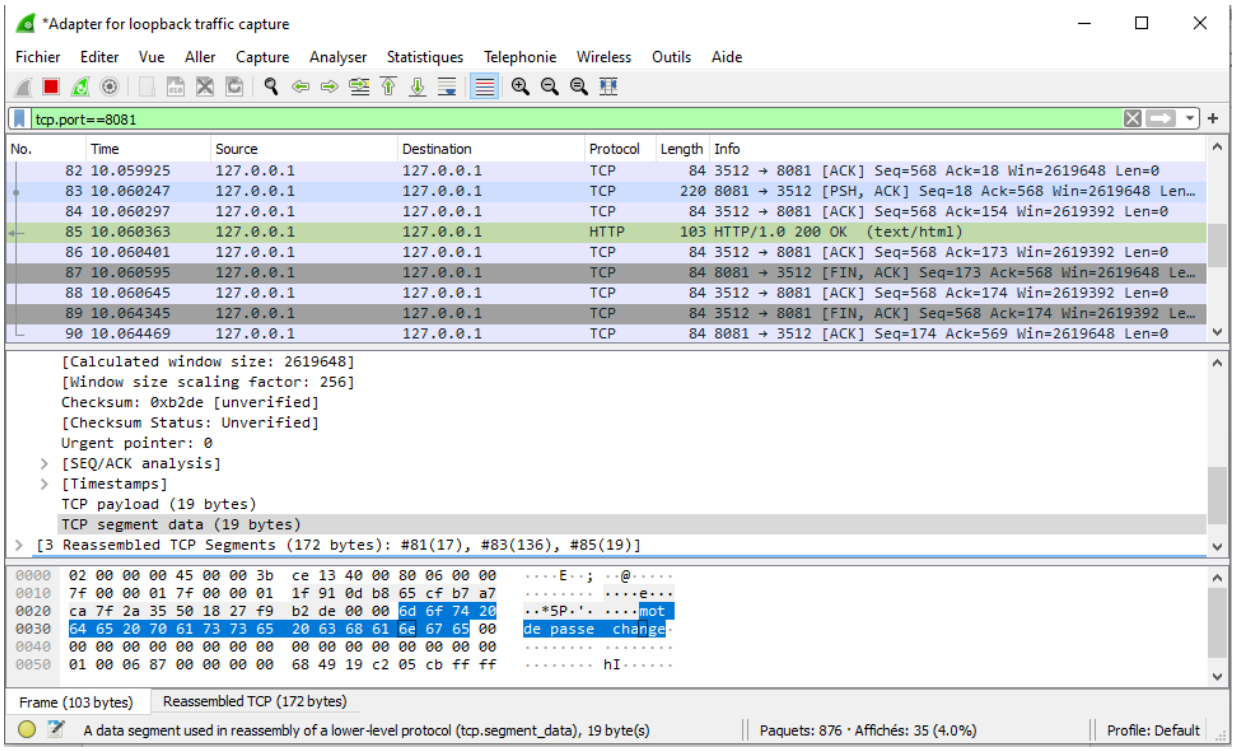
- ## Etape 1 : Vérification du serveur http

Pour cette étape nous allons utiliser le logiciel Wireshark pour faire cette vérification.

Le logiciel Wireshark permet la capture et l'analyse de trames sur Ethernet. Son utilité est indéniable pour contrôler le bon fonctionnement de réseau ou vérifier les trames transitant sur une interface d'un commutateur ou analyser les trafics inutiles ou ceux impactant les performances du réseau.

Dans le code initial du serveur qui imprime le message secret tant qu'il aura le SECRET_Url ensemble de variables d'environnements. Dans ce cas, le SECREt_URL est http://27.0.0.1 :8081. Donc, notre plan est donner a chaque membre du club l'URL secrète et de leur dire de la garder secrète et sûre. Notre message secret ne serait toujours pas sécurisé. Pour montrer pourquoi vous devez en savoir un peu plus sur la surveillance du trafic réseau. Pour ce faire, nous allons utiliser **Wireshark**.





La façon dont notre serveur actuel s'exécute n’est **pas sécurisé**. HTTP enverra tout en clair pour que tout le monde puisse le voir. Cela signifie que même si quelqu'un n'a pas votre SECRET_URL, ils peuvent toujours voir tout ce que vous faites tant qu'ils peuvent surveiller le trafic sur *tout* entre vous et le serveur. On peut voir ici clairement le mot de secret **mot de passe change.**

- **Etape 2 : Génération du certificat de l’autorité de certification**

Grâce à ces deux fonctions, nous pouvons générer votre paire de clés privée et publique assez rapidement en Python :

```python
from PKI_utile import generate_private_key, generate_public_key
private_key = generate_private_key("ca-cle-privee.pem" ,"security")

generate_public_key(

    private_key,
    nomdefichier="ca-cle-publique.pem",
    country="FR",
    state="Bourgogne Franche-Comte",
    locality="Belfort",
    org= "Societe SA",
    alt_names=["localhost"],
    hostname="mon-ca.com",
    )
```

Après avoir importé vos fonctions d'aide depuis PKI_utile, nous devons d'abord générer notre clé privée et l'enregistrer dans le fichier **ca-cle-privee.pem**. Nous passons ensuite cette clé privée dans **generate_public_key ()** pour générer votre clé publique. Dans notre répertoire, nous avons maintenant avoir deux fichiers :

```
ca-cle-privee.pem

ca-cle-publique.pem
```

Pour créer une CSR, nous avons d'abord besoin d'une clé privée. Heureusement, nous pouvons utiliser la même **generate_private_key ()** que lorsque nous avons créé la clé privée de notre CA. En utilisant la fonction ci-dessus et les méthodes définies précédemment, nous pouvons faire ce qui suit :

```python
from PKI_utile import generate_csr, generate_private_key
cle_privee_serveur = generate_private_key("serveur-cle-privee.pem", "Pass")

generate_csr(

    cle_privee_serveur,
    nomdefichier="serveur_csr.pem",
    country="FR",
    state="Bourgogne Franche-Comte",
    locality="Belfort",
    org= "Societe SA",
    alt_names=["localhost"],
    hostname="mon-site.com",
    )

```

Après avoir exécuté ces étapes dans une console, vous devriez vous retrouver avec deux nouveaux fichiers :

`serveur-cle-privee.pem`

`serveur_csr.pem`

- **Le code d'affichage de CA**

```python
import sys
import os
import ssl
from pprint import pprint as pp


def main():
    cert_file_name = os.path.join(os.path.dirname(__file__), "serveur-cle-publique.pem")
    try:
        cert_dict = ssl._ssl._test_decode_cert(cert_file_name)
        pp(cert_dict)

    except Exception as e:
        print("Error decoding certificate: {:}".format(e))


if __name__ == "__main__":
    print("Python {:s} on {:s}\n".format(sys.version, sys.platform))
    main()
```

- **Résultat attendu**

```
In [5]: runfile('C:/Users/AZIZOU One/Documents/Projet MI44/Aficher_certificat.py',
wdir='C:/Users/AZIZOU One/Documents/Projet MI44')
Reloaded modules: PKI_utile
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)] on win32

{'issuer': ((('countryName', 'FR'),),
            (('stateOrProvinceName', 'Bourgogne Franche-Comte'),),
            (('localityName', 'Belfort'),),
            (('organizationName', 'Societe SA'),),
            (('commonName', 'mon-ca.com'),)),
 'notAfter': 'Aug 17 20:37:35 2020 GMT',
 'notBefore': 'Jun 18 20:37:35 2020 GMT',
 'serialNumber': '69F5A425BD789DECC443A6DD5B2FD00CA152ADC0',
 'subject': ((('countryName', 'FR'),),
             (('stateOrProvinceName', 'Bourgogne Franche-Comte'),),
             (('localityName', 'Belfort'),),
             (('organizationName', 'Societe SA'),),
             (('commonName', 'mon-site.com'),)),
 'subjectAltName': (('DNS', 'localhost'),),
 'version': 3}

In [6]:
```

- **Etape 3 : Génération du certificat du serveur**

```
9   from PKI_utile import sign_csr
10  from cryptography import x509
11  from cryptography.hazmat.backends import default_backend
12  from cryptography.hazmat.primitives import serialization
13  from getpass import getpass
14
15
16  csr_file = open("serveur_csr.pem", "rb")
17  csr = x509.load_pem_x509_csr(csr_file.read(), default_backend())
18  print(csr)
19
20  ca_public_key_file = open("ca-cle-publique.pem", "rb")
21  ca_public_key = x509.load_pem_x509_certificate(
22      ca_public_key_file.read(),
23      default_backend()
24      )
25  print(ca_public_key)
26
27
28  ca_private_key_file = open("ca-cle-privee.pem", "rb")
29  ca_private_key = serialization.load_pem_private_key(
30      ca_private_key_file.read(),
31      getpass().encode("utf-8"),
32      default_backend(),
33      )
34
35  sign_csr(csr, ca_public_key, ca_private_key, "serveur-cle-publique.pem")
36
37
```

Cette étape consiste à créer un certificat pour cela il faut lancer la console Python et à utiliser **sign_csr ()**. Nous devons charger notre CSR et la clé privée et publique de votre CA. On commence par charger notre CSR :

- **Ligne 16-18 :** Dans cette section du code, nous allons ouvrir notre fichier **serveur-csr.pem** et utiliser **x509.load_pem_x509_csr()** pour créer votre objet **csr**. Ensuite, on doit charger la clé publique de notre CA :

- **Ligne 20-25 :** Une fois de plus, nous avons créé un objet **ca_cle_publique.pm** qui peut être utilisé par sign_csr(). Le module x509 avait le pratique load_pem_x509_certificate() pour vous aider. La dernière étape consiste à charger la clé privée de notre CA :

- Ce code chargera notre clé privée. Notre clé privée a été cryptée avec le mot de passe que nous avons spécifié. Avec ces trois éléments, nous pouvons maintenant signer votre CSR et générer une clé publique vérifiée :

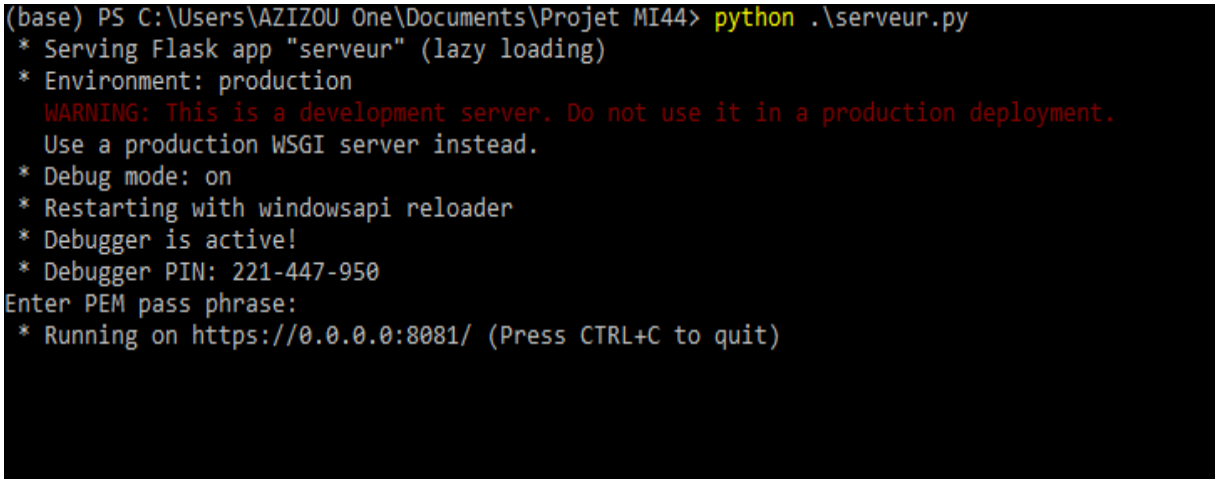
```
sign_csr(csr, ca_cle_publique, ca_cle_privee, "server-cle-publique.pem")
```

- **Etape 4 : Connexion HTTPS**

En utilisant notre fichier original server.py, exécutez la commande suivante pour lancer votre toute nouvelle application Python HTTPS :

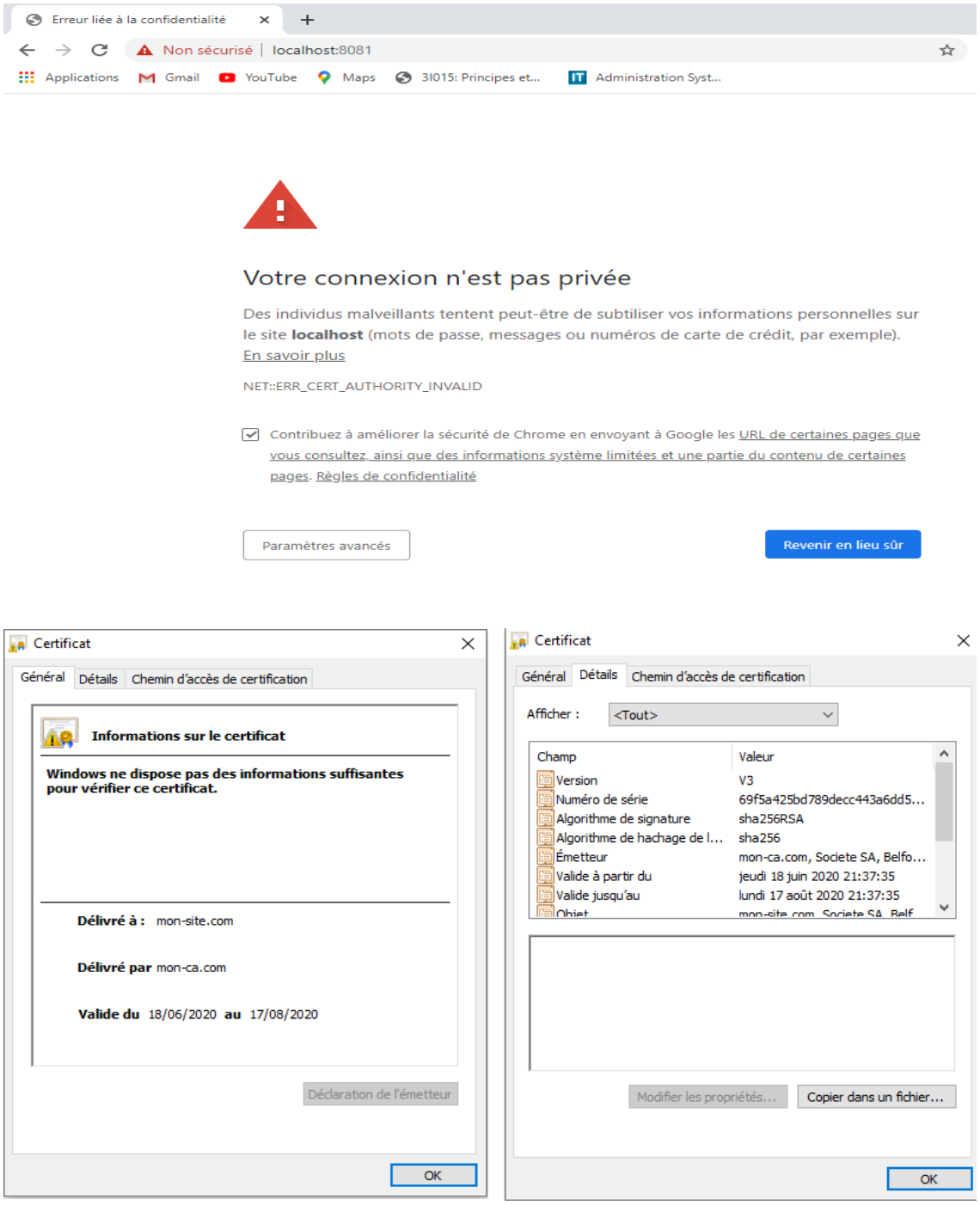
```
8   from flask import Flask
9   # définir le message secret
10  MESSAGE_SECRET="le message secert que je vous envoies"
11  app=Flask(__name__)
12  @app.route("/")
13
14  def get_secret_message():
15      return MESSAGE_SECRET
16
17  if __name__=="__main__":
18      app.run(
19              debug=True,
20              host="0.0.0.0",
21              port=8081,
22              ssl_context=('serveur-cle-publique.pem','serveur-cle-privee.pem')
23              )
```

Nous avons maintenant un serveur Python HTTPS fonctionnant avec notre propre paire de clés privées-publiques, qui a été signée par notre propre autorité de certification !



https://localhost:8081 nous a donné un certificat. Nous avons aussi vérifié l'émetteur du certificat qu'il nous a donné, et d'après toutes les autorités de certification que je connais, cet émetteur n'en fait pas partie.

Si nous essayons de naviguer sur votre site web avec votre navigateur, vous obtiendrez ce message, ainsi qu'on peut voir aussi l'autorité du certificat qui n'est connue.


Erreur liée à la confidentialité
Non sécurisé | localhost:8081
Applications
Gmail
YouTube
Maps
3I015: Principes et...
Administration Syst...
Votre connexion n'est pas privée
Des individus malveillants tentent peut-être de subtiliser vos informations personnelles sur le site localhost (mots de passe, messages ou numéros de carte de crédit, par exemple).
En savoir plus
NET::ERR_CERT_AUTHORITY_INVALID
Contribuez à améliorer la sécurité de Chrome en envoyant à Google les URL de certaines pages que vous consultez, ainsi que des informations système limitées et une partie du contenu de certaines pages. Règles de confidentialité
Paramètres avancés
Revenir en lieu sûr
Certificat
Général
Détails
Chemin d'accès de certification
Informations sur le certificat
Windows ne dispose pas des informations suffisantes pour vérifier ce certificat.
Délivré à : mon-site.com
Délivré par mon-ca.com
Valide du 18/06/2020 au 17/08/2020
Déclaration de l'émetteur
OK
Certificat
Général
Détails
Chemin d'accès de certification
Afficher : <Tout>
Champ
Valeur
Version
V3
Numéro de série
69f5a425bd789decc443a6dd5...
Algorithme de signature
sha256RSA
Algorithme de hachage de l...
sha256
Émetteur
mon-ca.com, Societe SA, Belfo...
Valide à partir du
jeudi 18 juin 2020 21:37:35
Valide jusqu'au
lundi 17 août 2020 21:37:35
Modifier les propriétés...
Copier dans un fichier...
OK

- **Etape 5 : Quels que améliorations**

- Pour faire des améliorations, il y a un autre aspect de l'équation d'authentification Python HTTPS, et c'est le serveur. Il est possible de mettre en place une vérification de certificat pour un certificat de serveur également. Cependant, l'authentification du serveur peut être un outil très puissant.
- L'authentification consiste à s'assurer qu'une personne dit la vérité sur un ou plusieurs de ses attributs. Dans le contexte de la communication entre 2 parties, un attribut d'une partie pourrait être son nom, sa raison sociale, ou tout autre élément de ce qu'elle est ce qu'elle sait, ou ce qu'elle possède, qui pourrait assurer l'autre partie de l'identité de sa partenaire.
- Par exemple lors de l'accès à un site web de vente en ligne, le consommateur voudrait, au moment du paiement, être sur qu'il donne son argent à la boutique en ligne dont il connait la réputation, et non à un escroc qui encaissera l'argent sans suite. C'est pour cela qu'il a besoin d'authentification : pour être sur que le site web est bien la boutique, d'après son nom de domaine. On a donc souvent besoin d'identification du serveur web, mais plus rarement celle du client. SSL permet l'authentification des deux parties.

Conclusion :

Dans ce TP, nous avons appris certains des fondements des communications sécurisées sur l'internet aujourd'hui. Tout au long de ce TP, nous avons acquis une compréhension de plusieurs sujets :

- Cryptographie
- HTTPS et TLS
- Infrastructure à clé publique
- Certificats