Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Sécurité des usages TIC

Mini Projet

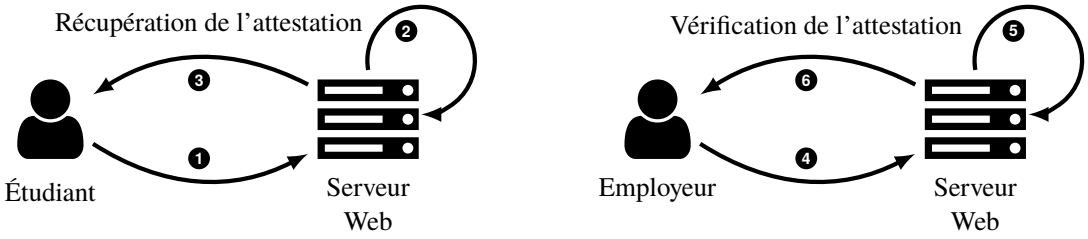
Authentification, Web sécurisé & Stéganographie

Une société de certification **CertifPlus** vous a contacté pour mettre en œuvre leur procédé de **diffusion électronique sécurisée d'attestation de réussite** aux certifications qu'elle délivre.

Il faut garantir **l'authenticité** de l'attestation délivrée de manière électronique sous forme d'image :

- ▷ l'image contient une information **visible** :
  - ◇ le **nom de la personne** recevant l'attestation de réussite ;
  - ◇ le **nom de la certification** réussie ;
  - ◇ un QRcode contenant la **signature de ces informations** ;
- ▷ l'image contient une information **dissimulée** :
  - ◇ une information infalsifiable est dissimulée par **stéganographie** dans l'image. Cette information reprend les informations visibles de l'attestation ainsi que la date de délivrance garantie par un « *timestamp* » signé par une autorité d'horodatage « `www.freetsa.org` ».

La société conçoit ce procédé sous forme d'un WebService ou Application Web, c-à-d un serveur TCP utilisant le protocole HTTP, supportant différentes requêtes :



1. la requête de récupération de l'attestation :
   - ◇ `http://localhost:8080/certificat`
     - □ requête POST ❶ contenant les informations de l'étudiant ;
     - □ une image en réponse ❸ ;
   - ◇ le WebService traite la demande de délivrance d'attestation ❷ :
     - ⋆ récupération du nom, prénom, intitulé de la certification ;
     - ⋆ réalise signature de ces informations avec la date de demande ;
     - ⋆ obtention d'un « *timestamp* », ou estampille temporelle auprès du tiers horodateur ;
     - ⋆ dissimulation de cette estampille par stéganographie dans l'image et intégration de la signature dans le QRcode ;
     - ⋆ retourne l'image en réponse ❸ ;
2. la requête de vérification de l'attestation :
   - ◇ `http://localhost:8080/verification`
     - □ requête POST ❹ contenant l'image de l'attestation ;
     - □ une réponse « *certifié* » ou « *attestation erronée* » ❺ ;
   - ◇ le WebService traite la demande de véfication de l'attestation :
     - ⋆ extrait et l'estampille dissimulée dans l'image par stéganographie ;
     - ⋆ vérifie la signature codée dans le QRcode.

La société CertifPlus, va se comporter en tant qu'« Autorité de Certification » pour se délivrer des certificats suivant les usages dont elle aura besoin pour réaliser son service.

Le web service

Pour la réalisation du WebService, nous utiliserons le « *micro-framework* » `bottle` disponible sous forme d'une bibliothèque Python 3 :

```
xterm
~/ $ python3 -m pip install bottle
```

Le programme Python réalisant la gestion des différente requêtes demandées :

```
#!/usr/bin/python3
from bottle import route, run, template, request, response

@route('/creation', method='POST')
def création_attestation():
    contenu_identité = request.forms.get('identite')
    contenu_intitulé_certification = request.forms.get('intitule_certif')
    print('nom prénom :', contenu_identité, ' intitulé de la certification :',
       contenu_intitulé_certification)
    response.set_header('Content-type', 'text/plain')

    return "ok!"

@route('/verification', method='POST')
def vérification_attestation():
    contenu_image = request.files.get('image')
    contenu_image.save('attestation_a_verifier.png',overwrite=True)
    response.set_header('Content-type', 'text/plain')

    return "ok!"

@route('/fond')
def récupérer_fond():
    response.set_header('Content-type', 'image/png')
    descripteur_fichier = open('fond_attestation.png','rb')
    contenu_fichier = descripteur_fichier.read()
    descripteur_fichier.close()

    return contenu_fichier

run(host='0.0.0.0',port=8080,debug=True)
```

le type MIME d'une réponse au format texte

la requête « /fond » montre comment renvoyer une image

Vous pouvez avoir un warning de la part du programme⟹ignorer

Pour faire les requêtes, nous utiliserons l'outil `curl` :

- pour l'envoi des données de l'étudiant :

```
xterm
~/ $ curl -X POST -d 'identite=toto' -d 'intitule_certif=SecuTIC' \
http://localhost:8080/creation
```

- pour la requête de vérification et l'envoi de l'image :

```
xterm
~/ $ curl -v -F image=@fond_attestation.png http://localhost:8080/verification
```

- pour une requête récupérant en réponse une image :

```
xterm
~/ $ curl -v -o mon_image.png http://localhost:8080/fond
```

Pendant ce temps sur le WebService :

```
xterm
~/ $ python3 serveur_web.py
Bottle v0.12.19 server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.

nom prénom : toto  intitulé de la certification : SecuTIC
127.0.0.1 - - [30/Mar/2021 13:48:22] "POST /creation HTTP/1.1" 200 3
```

pour la requête :

```
xterm
~/ $ curl -X POST -d 'identite=toto' -d 'intitule_certif=SecuTIC' \
http://localhost:8080/creation
```

Attention

- *Vous adapterez et combinerez ces exemples dans le cas de votre projet.*
- *Vous écrirez des scripts pour faciliter la manipulation des commandes curls et afin de les intégrer dans l'archive finale de votre projet.*

Le code à ajouter dans le code du WebService :

- ⋆ dans la fonction `vérification_attestation()` gérant la route « */verification* » :
  - ◇ lit l'image soumise et récupère le contenu dissimulé par stéganographie :

    Nom Prénom ‖ Intitulé certification ‖ timestamp

    Où l'opérateur ‖ désigne la concaténation.
    Le nom et prénom, ainsi que l'intitulé de la certification devront être complétés afin d'obtenir une chaîne de 64 caractères.
    Le « timestamp *» a une taille fixe de n octets, dépendant de la taille de clé utilisée par le service d'horodatage.*
  - ◇ découpe le contenu en deux parties : informations sur 64 octets et estampille sur *n* octets ;
  - ◇ vérifie le « *timestamp* » par rapport à l'AC d'horodatage ;
  - ◇ récupère la partie QRcode de l'image pour récupérer la signature des données ;
  - ◇ vérifie la signature de la partie informations avec la clé publique de CertifPlus ;
  - ◇ retourne le résultat de cette vérification.
- ⋆ dans la fonction `création_certificat` associée à la route `/creations` :
  - ◇ récupération du nom et prénom de l'étudiant, ainsi que de l'intitulé de la certification, .
  - ◇ création de l'image :
    - ⋆ utilisation d'un fond contenant un tracé infalsifiable obtenu à partir d'un SpirographEnigma2022 : `http://p-fb.net/fileadmin/fond_attestation.png`
    - ⋆ intégration d'un QRCode contenant la **signature** que vous aurez converties au format ASCII ;
    - ⋆ intégration du texte Nom Prénom et intitulé de la certification.

  - ◇ construction du bloc d'informations (Nom Prénom, Intitulé certification)
  - ◇ dissimulation par stéganographie du bloc d'information et du « *timestamp* » dans l'image ;
  - ◇ ajout du QRcode contenant la signature de ce bloc avec la clé privée de CertifPlus,
  - ◇ fourniture de l'image en réponse.

Travail à réaliser

- ☐ créer une AC avec une configuration bien choisie délivrant des cartificats utilisant les **Courbes Elliptiques** (voir le travail réalisé dans la fiche de TP n°5) ;
- ☐ créer un certificat permettant la signature ;
- ☐ choisir un algorithme de signature et déduire la taille *n* de la signature ;
- ☐ ajouter le code chargé de traiter les données soumises par l'utilisateur dans la fonction `création_certificat` :
  - ◇ construction du bloc d'informations à insérer dans l'image (concaténations et signature) ;
  - ◇ insertion de ce bloc par stéganographie et envoi sécurisé de l'image obtenue ;
- ☐ ajouter le code d'extraction de preuve permettant de vérifier l'attestation dans la fonction `vérification_certificat` :
  - ◇ récupération des données dissimulées par stéganographie ;
  - ◇ récupération du QRCode et vérification de la signature ;
  - ◇ renvoi du résultat.

Opérations graphiques

Pour réaliser :

- la génération du texte à combiner dans l'image finale à l'aide de l'outil ImageMagick :

```
xterm
$ convert -size 1000x600 -gravity center -pointsize 56 label:"Attestation de
réussite\ndélivrée à P-F.B" -transparent white texte.png
```

- pour la création du QRcode vous utiliserez les bibliothèques suivantes :

```
xterm
$ sudo apt install python3-pip
$ python3 -m pip install qrcode numpy Pillow
$ python3 -m pip opencv-python
```

Pour l'utiliser vous utiliserez la méthode suivante :

```
import qrcode

data = "https://p-fb.net/"
nom_fichier = "qrcode.png"
qr = qrcode.make(data)
qr.save(nom_fichier, scale=2)
```

Un fichier « `qrcode.png` » doit avoir été créé dans votre répertoire courant.

- la combinaison des images en l'image finale avec ImageMagick :

```
xterm
composite -gravity center texte.png fond_attestation.png combinaison.png
composite -geometry +1418+934 qrcode.png combinaison.png attestation.png
```

- pour récupérer la partie de l'image contenant le QRcode :

```
from PIL import Image

attestation = Image.open(fileName)
qrImage = attestation.crop((1418,934,1418+210,934+210))
qrImage.save("qrcoderecupere.png", "PNG")
```

- pour récupérer les données contenues dans le QRcode, il faudra installer les bibliothèques suivantes :

```
xterm
$ sudo apt install libzbar0 libzbar-dev
$ python3 -m pip install zbarlight
```

Puis pour l'utiliser :

```
import zbarlight
from PIL import Image

image = Image.open("qrcode_maousse.png")
data = zbarlight.scan_codes(['qrcode'], image)
```

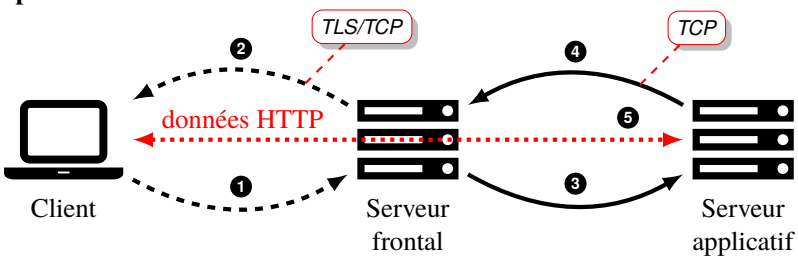
Stéganographie

Le programme suivant dissimule ou récupère les données en modifiant les bits de poids faible de la composante rouge de l'image.

```
#!/usr/bin/python3

from PIL import Image

def vers_8bit(c):
        chaine_binaire = bin(ord(c))[2:]
        return "0"*(8-len(chaine_binaire))+chaine_binaire

def modifier_pixel(pixel, bit):
        # on modifie que la composante rouge
        r_val = pixel[0]
        rep_binaire = bin(r_val)[2:]
        rep_bin_mod = rep_binaire[:-1] + bit
        r_val = int(rep_bin_mod, 2)
        return tuple([r_val] + list(pixel[1:]))

def recuperer_bit_pfaible(pixel):
    r_val = pixel[0]
    return bin(r_val)[-1]

def cacher(image,message):
        dimX,dimY = image.size
        im = image.load()
        message_binaire = ''.join([vers_8bit(c) for c in message])
        posx_pixel = 0
        posy_pixel = 0
        for bit in message_binaire:
           im[posx_pixel,posy_pixel] = modifier_pixel(im[posx_pixel,posy_pixel],bit)
           posx_pixel += 1
           if (posx_pixel == dimX):
                   posx_pixel = 0
                   posy_pixel += 1
           assert(posy_pixel < dimY)

def recuperer(image,taille):
        message = ""
        dimX,dimY = image.size
        im = image.load()
        posx_pixel = 0
        posy_pixel = 0
        for rang_car in range(0,taille):
           rep_binaire = ""
           for rang_bit in range(0,8):
                   rep_binaire += recuperer_bit_pfaible(im[posx_pixel,posy_pixel])
                   posx_pixel +=1
                   if (posx_pixel == dimX):
                           posx_pixel = 0
                           posy_pixel += 1
           message += chr(int(rep_binaire, 2))
        return message


# Valeurs par defaut
nom_defaut = "image_test.png"
message_defaut = "Hello world"
choix_defaut = 1

# programme de demonstration
saisie = input("Entrez l'operation 1) cacher 2) retrouver [%d]"%choix_defaut)
choix = saisie or choix_defaut

if choix == 1:
        saisie = input("Entrez le nom du fichier [%s]"%nom_defaut)
        nom_fichier = saisie or nom_defaut
        saisie = input("Entrez le message [%s]"%message_defaut)
        message_a_traiter = saisie or message_defaut
        print ("Longueur message : ",len(message_a_traiter))
        mon_image = Image.open(nom_fichier)
        cacher(mon_image, message_a_traiter)
        mon_image.save("stegano_"+nom_fichier)
else :
        saisie = input("Entrez le nom du fichier [%s]"%nom_defaut)
        nom_fichier = saisie or nom_defaut
        saisie = input("Entrez la taille du message ")
        message_a_traiter = int(saisie)
        mon_image = Image.open(nom_fichier)
        message_retrouve = recuperer(mon_image, message_a_traiter)
        print (message_retrouve)
```

Vous adapterez ce code à votre programme en utilisant la taille du bloc d'informations + celle du timestamp pour paramétrer la récupération automatique des données.

Pour la mise en place de la connexion TLS



En **exploitation** dans le réseau d'une entreprise comme au sein du « *Cloud* », le serveur applicatif qui s'occupe de traiter les requêtes est rarement « *exposé* » directement sur Internet :

- ▷ un serveur **frontal** est utilisé pour gérer la connexion avec le client ;
- ▷ ce serveur est accessible par Internet :
  - ◇ il reçoit la connexion du client sur le nom DNS et l'adresse IP connue par ces clients ❶ ;
  - ◇ il utilise un certificat pour établir une connexion sécurisée TLS sur TCP ❷ ;

  ⟹ le client authentifie le serveur avec l'AC utilisée pour le certificat du serveur.
- ▷ une fois la connexion établie, la requête HTTP est transmise au serveur applicatif qui la traite :
  - ◇ le serveur applicatif est dans le réseau de l'entreprise ou dans le « *cloud* » ;
  - ◇ une connexion TCP s'établie entre le serveur **frontal** et ce serveur applicatif ❸ ;
  - ◇ le serveur frontal relaie la requête HTTP et les données reçue par TLS/TCP depuis le client, dans la connexion étable avec le serveur applicatif ❺ ;
  - ◇ la réponse du serveur applicatif est ensuite relayée jusqu'au client au travers de la connexion TLS par le serveur frontal ❺.

Le serveur **frontal** :

- ▷ sert de « *reverse proxy* » : il peut analyser le contenu pour faire de la protection contre des attaques sur HTTP et les serveurs applicatifs ;
- ▷ sert de « *load balancer* » : il peut rediriger vers différents serveurs applicatifs dont le nombre peut augmenter en fonction de l'afflux de clients pour traiter tous ces clients ;
- ▷ gère la partie cryptographique liée à TLS :
  - ◇ il décharge le serveur applicatif de cet effort et bénéficie d'éléments de qualité (entropie, paramètres des algorithmes cryptographiques) ;
  - ◇ il peut être mis à jour dès qu'une faille de sécurité est découverte ;
  - ◇ il peut bénéficier de protection contre les « *DoS* », les attaques « *brute force* » ;
  - ◇ sa sécurité est la responsabilité de l'hébergeur ou « *cloud provider* » indépendamment de l'application Web.

⟹ **Attention** : le serveur frontal n'est pas suffisant pour protéger l'application Web qui peut souffrir de vulnérabilités propres.

En pratique, les attaques sur l'infrastructure du Cloud sont plus rares que celles sur les applications Web.

Dans notre cas nous utiliserons un simple relais « *TLS/TCP* » ⟺ « *TCP* » avec `socat` :

xterm
```
~/ $ socat \
openssl-listen:9000,fork,cert=bundle_serveur.pem,cafile=ecc.ca.cert.pem,verify=0 \
tcp:127.0.0.1:8080
```

- □ le paramètre `cert=` est un fichier texte contenant à la fois la clé privée et le certificat du serveur :

  xterm
  ```
  ~/ $ cat ecc.serveur.key.pem ecc.serveur.pem > bundle_serveur.pem
  ```
- □ l'accès au WebService ou application Web se fait maintenant en TLS et sur le port 9000 :

  xterm
  ```
  ~/ $ curl -v -X POST -d 'identite=toto' -d 'intitule_certif=SecuTIC' --cacert \
  ecc.ca.cert.pem https://localhost:9000/certificat
  ```

  Attention : l'identité donnée dans le certificat doit être `localhost` (`CN = localhost`).
- □ la commande `socat` doit s'exécuter en même temps que le script Python du serveur Web.

Ajout du SSO de l'Université pour autoriser la délivrance du diplôme

Soit le code suivant permettant de récupérer le « cookie » d'autorisation une fois l'authentification réussie auprès du server `cas.unilim.fr`.

```
#!/usr/bin/python3

import urllib.request
import urllib.parse
import re

re_token = re.compile(rb'name="token" value="([^"]+)"')

request = urllib.request.Request('https://cas.unilim.fr')
rep = urllib.request.urlopen(request)
contenu = rep.read()

resultat = re_token.search(contenu)

if resultat:
    token = resultat.group(1)
    print(token)
else:
    sys.exit(1)
cookieProcessor = urllib.request.HTTPCookieProcessor()
opener = urllib.request.build_opener(cookieProcessor)
data = urllib.parse.urlencode({'user':'toto','password':'secret','token':token})

request = urllib.request.Request('https://cas.unilim.fr',bytes(data,encoding='as
cii'))
reponse = opener.open(request)
cookies = [c for c in cookieProcessor.cookiejar if c.name=='lemonldap']
print(cookies)
```

Intégrez ce code pour ne délivrer le certificat à l'étudiant que s'il arrive à se connecter au serveur « cas » de l'Université.

Vous rajouterez dans votre rapport final une explication du fonctionnement du SSO et des différents éléments échangés.

Travail à soumettre pour l'évaluation en binôme (trinôme possible en accord avec l'encadrant)

a. Ajouter le code nécessaire au code initial du WebService .

b. Rédiger un **court** rapport, au format PDF, pour détailler le fonctionnement de votre programme, avec un exemple d'exécution (notice d'utilisation avec copie d'écran).

c. Rédiger une courte **analyse de risques** en identifiant :
   - des biens à protéger en tant qu'actifs primaires et secondaires ;
   - des menaces sur ces biens ;
   - des relations enre les deux.

Pour rendre votre projet

- Vous créerez une archive contenant :
  - le certificat racine de l'AC ;
  - les fichiers de configuration de l'AC ;
  - le certificat de l'application généré par l'AC ;
  - le mot de passe d'accès à la clé privée de l'AC ;
  - les sources de vos différents programmes Python ;
  - les scripts de requêtes curl ;
  - un exemple d'attestation réalisée avec votre application qui puisse être vérifié par votre programme ;
  - le rapport ;
  - l'analyse de risques.
- Vous déposerez l'archive sur Community.