

Securité des usages TIC

Ouarafana Badr / Joulain Vincent

Avril 2023

Contents

1	Mise en place du serveur frontal	1
2	Fonction Création	3
3	Fonction Vérification	4
4	Exemple d'exécution	5
5	Exemple d'exécution (certificat invalide)	6
6	Analyse de risques	6
6.1	Faible de l'application	7
6.2	Solutions	8

1 Mise en place du serveur frontal

La première chose à faire afin de mettre en place le serveur frontal a été de créer une AC ainsi que de lui faire signer un certificat authentifiant notre serveur. Nous avons utilisé l'algorithme **NIST P-256** aussi appelé **prime256v1** car c'est celui utilisé par défaut sur openssl et par conséquent sur TLS. Il est donc très utilisé et certainement optimisé pour ce genre d'utilisation ce qui est un des facteurs les plus importants lors du choix d'un algorithme de

pour-ce faire, il nous a fallu :

- générer les différentes clés privées

```
openssl ecparam -out ecc.ca.private.key.pem -name prime256v1 -genkey
openssl ecparam -out ecc.server.private.key.pem -name prime256v1 -genkey
```

- générer les différentes clés publiques

```
openssl ec -in ecc.ca.private.key.pem -pubout -out ecc.ca.public.key.pem
openssl ec -in ecc.server.private.key.pem -pubout -out ecc.server.public.key.pem
```

- créer à partir de celles-ci les certificats de l'AC et du serveur

– pour l'AC :

```
openssl req -config <(printf "[req]\ndistinguished_name=dn\n[dn]\n[ext]\nbasicConstraints=CA:TRUE")
-new -nodes -subj "/C=FR/L=LIMOGES/O=CRYPTIS/OU=SECUTIC/CN=ACCRYPTIS" -x509 -extensions ext -sha256
-key ecc.ca.private.key.pem -text -out ecc.ca.cert.pem
```

– pour le serveur :

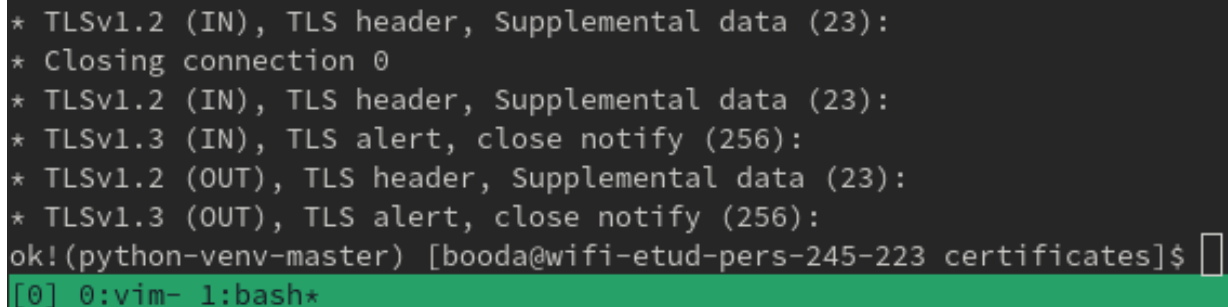
```
openssl req -config <(printf "[req]\ndistinguished_name=dn\n[dn]\n[ext]\nbasicConstraints=CA:FALSE")
-new -subj "/C=FR/L=LIMOGES/O=CRYPTIS/OU=SECUTIC/CN=localhost" -reqexts ext -sha256
-key ecc.server.private.key.pem -text -out ecc.server.csr.pem
```

```
openssl x509 -req -days 3650 -CA ecc.ca.cert.pem -CAkey ecc.ca.key.pem -CAcreateserial
-extfile <(printf "basicConstraints=critical,CA:FALSE") -in ecc.server.csr.pem -text
-out ecc.server.cert.pem
```

```
cat ecc.server.private.key.pem ecc.server.cert.pem > server_bundle.pem
```

le fichier **generatecert.sh** permet de générer toutes les clés et certificats utiles à notre application.

Il nous est maintenant possible de faire une requête HTTP sur TLS depuis le client :



```
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* Closing connection 0
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS alert, close notify (256):
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.3 (OUT), TLS alert, close notify (256):
ok!(python-venv-master) [booda@wifi-etud-pers-245-223 certificates]$ [ ]
[0] 0:vim- 1:bash*
```

Figure 1: requête du client : <https://localhost:9000/creation>

2 Fonction Création

Afin de créer un certificat de notre société certifplus nous commençons par vérifier la validité des arguments reçus :

```
# get certificat's data
identity = request.forms.get('identite')
title = request.forms.get('intitule_certif')

#check the validity of parameters
if identity == None or title == None or len(identity+title)>64:
    return "Not valid"
```

Nous utilisons ensuite openssl afin de signer les données avec la clé privée du serveur (fonction sha256) :

```
#hash the file with the private key
result = subprocess.run(['openssl', 'dgst', '-sha256', '-sign', '../certificates/ecc.server.private.key.pem', '-out',
'/tmp/output.sig', '/tmp/output.txt'], check=True)
if result.returncode != 0:
    print(f"failed with error:\n{result.stderr.decode('utf-8')}")
    return None
```

Il nous est maintenant possible de récupérer le Time Stamp, c'est à dire les fichier request et response qui nous permettrons par la suite de nous assurer de la date de la certification donnée au près de Free TSA :

```
#create the time stamp
result = subprocess.run(['openssl', 'ts', '-query', '-data', '/tmp/output.txt', '-no_nonce', '-sha512', '-cert', '-out',
'../certificates/file.tsq'], check=True)
if result.returncode != 0:
    print(f"failed with error:\n{result.stderr.decode('utf-8')}")
    return None

result = subprocess.run(['curl', '-H', 'Content-Type: application/timestamp-query', '--data-binary',
'@../certificates/file.tsq', 'https://freetsa.org/tsr', '--output', '../certificates/file.tsr'], check=True)
if result.returncode != 0:
    print(f"failed with error:\n{result.stderr.decode('utf-8')}")
    return None
```

La prochaine étape va être de fabriquer un QR code contenant la signature précédemment générée ainsi que définir les données que nous allons cacher dans la certification :

```
# create QR code
qr = qrcode.make(signature)
qr.save('/tmp/qrcode.png')

#create the steganographic data to be embedded in the image
stegano_data = concatenated_data + base64.b64encode(tsq_file+tsr_file).decode("utf_8")
```

Dernière étape, assembler les images composant notre attestation à l'aide d'ImageMagick ainsi que cacher les données souhaités à l'intérieur :

```
subprocess.run(["composite", "-gravity", "center", "/tmp/text.png", "../medias/fond_attestation.png",
"/tmp/combinaison.png"], check=True)

#steganography
image = Image.open("/tmp/combinaison.png")
cacher(image , stegano_data)
image.save("/tmp/combinaison.png")

subprocess.run(["composite", "-geometry", "+1418+934", "/tmp/qrcode.png", "/tmp/combinaison.png",
"/tmp/attestation.png"], check=True)
```

L'attestation est enfin prête à être envoyée à l'utilisateur :

```
response.set_header('Content-type', 'image/png')
descripteur_fichier = open('/tmp/attestation.png', 'rb')
contenu_fichier = descripteur_fichier.read()
descripteur_fichier.close()
return contenu_fichier
```

3 Fonction Vérification

Dans l'objectif de vérifier si l'attestation reçue est bien valide, nous commençons par extraire les données cachées (de taille fixe) dans l'image :

```
message_length = 7512
message = recuperer(image, message_length)
```

Nous avons maintenant en notre possession les données cachées, c'est-à-dire le le nom/prenom du titulaire de la certification, de son l'intitulé ainsi que du Time Stamp.

Il va nous falloir dans un premier temps vérifier le Time Stamp, nous commençons par récupérer les fichier fournis par freetsa permettant d'effectuer la vérification :

```
if not os.path.exists("../certificates/freetsa/tsa.crt") or not os.path.exists("../certificates/freetsa/cacert.pem"):
    try:
        # Download the files if they don't exist
        subprocess.run(["wget", "https://freetsa.org/files/tsa.crt", "-P", "../certificates/freetsa"], check=True)
        subprocess.run(["wget", "https://freetsa.org/files/cacert.pem", "-P", "../certificates/freetsa"], check=True)
        print("Free TSA Download complete!")
    except subprocess.CalledProcessError as e:
        print("Error: ", e)
```

Nous pouvons maintenant vérifier le Time Stamp :

```
#verify time stamp
result = subprocess.run(['openssl', 'ts', '-verify', '-in', '/tmp/file_verif.tsr', '-queryfile', '/tmp/file_verif.tsq',
'-CAfile', '../certificates/freetsa/cacert.pem', '-untrusted', '../certificates/freetsa/tsa.crt'], check= True)
```

La prochaine étape va être d'extraire le QR code afin de récupérer la signature :

```
# extract qrcode
qr_code = image.crop((1418,934,1418+210,934+210))
qr_code.save("/tmp/qr-code-verif.png", "PNG")
qr_code = Image.open("/tmp/qr-code-verif.png")
signature = zbarlight.scan_codes(['qrcode'], image)
```

Une fois la signature récupérée, nous pouvons comparer les données obtenues et la signature juste extraite à l'aide de la clé publique du serveur afin de nous assurer de l'authenticité de la certification fournie :

```
result = subprocess.run(['openssl', 'dgst', '-sha256', '-verify', '../certificates/ecc.public.key.pem', '-signature',
'/tmp/signature.sig', '/tmp/received_data'])
print(result.returncode)
if result.returncode != 0:
    return "Not valid"
```

Si aucune erreur n'est apparue jusqu'ici, nous pouvons donc conclure que notre certification est bien valide.

4 Exemple d'exécution

Un test a été effectué avec le netlab, en prenant h1 comme serveur, h3 un client qui demande un certificat et h2 qui vérifie la validité du certificat.

Nota :

- Il faut modifier le fichier **generatecert.sh** et mettre l'adresse ip de h1.

```
openssl req -config <(<printf "[req]\ndistinguished_name=dn\n[dn]\n[ext]\nbasicConstraints=CA:FALSE") \
-new -subj "/C=FR/L=LIMOGES/O=CRYPTIS/OU=SECUTIC/CN=192.168.10.1" -reqexts ext -sha256 \
-key ecc.server.private.key.pem -text -out ecc.server.csr.pem
```

- Donner accès internet et la résolution d'adresse à h1.

```
h3
projectx@Raider:~/Desktop$ [h3] curl -X POST -d 'identite=toto' -d 'intitule certif=SecuTIC' --cacert /home/projectx/Desktop/SEC_TIC/project/certifplus/certificates/ecc.ca.cert.pem https://192.168.10.1:9000/creation --output image.png
% Total % Received % Xferd Average Speed Time Time Time Current
100 3124k 100 3124k 100 37 1955k 23 0:00:01 0:00:01 --:--:-- 1954k
projectx@Raider:~/Desktop$ [h3]

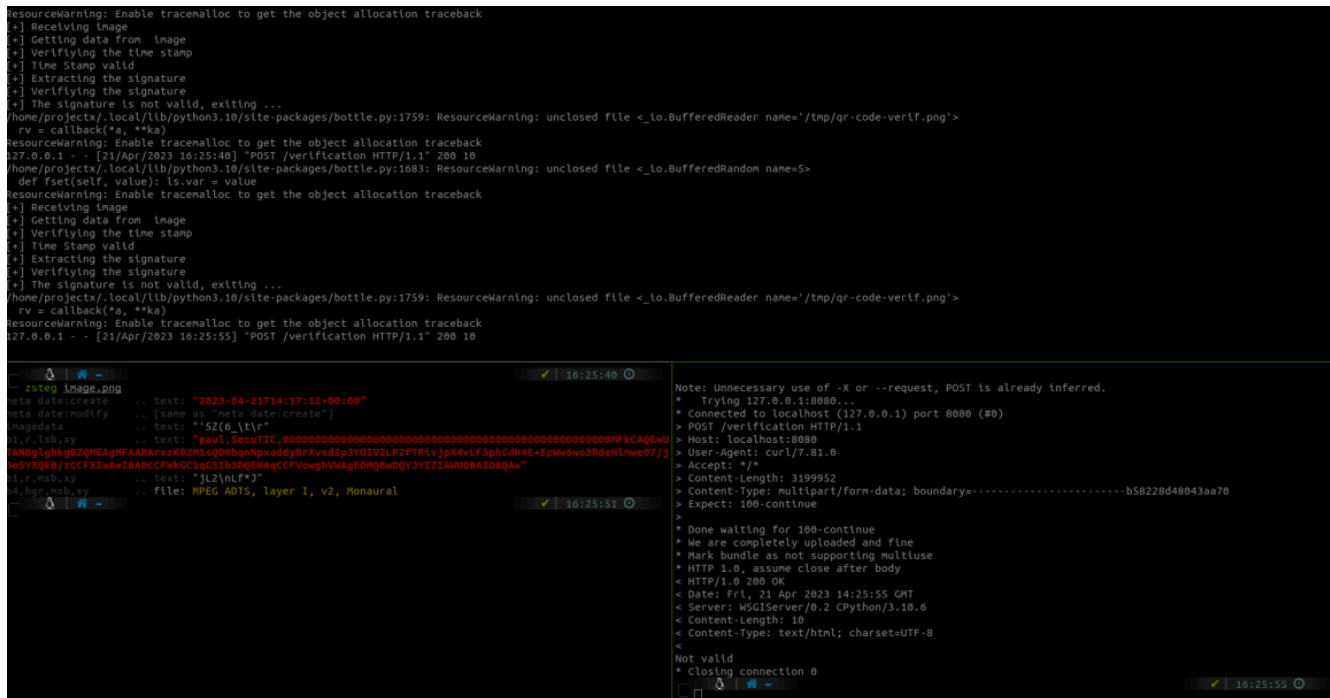
--1 "h2"
projectx@Raider:/etc/netns/h1$ [h2] cd
projectx@Raider:~$ [h2] cd Desktop/
projectx@Raider:~/Desktop$ [h2] curl -X POST -F image=@image.png --cacert /home/projectx/Desktop/SEC_TIC/project/certifplus/certificates/ecc.ca.cert.pem https://192.168.10.1:9000/verification
Valid
projectx@Raider:~/Desktop$ [h2]
projectx@Raider:~/Desktop$ [h2]

--2 "h1"
ResourceWarning: Enable tracemalloc to get the object allocation traceback
[+] Getting request
[+] Generating signature, identity : toto, title : SecuTIC
[+] Requesting time stamp
Using configuration from /usr/lib/ssl/openssl.cnf
% Total % Received % Xferd Average Speed Time Time Time Current
100 5585 0 5484 100 91 8375 138 --:--:-- --:--:-- --:--:-- 8513
[+] Encoding data into image
[+] Requesting API image from Google
[+] Generating QR code
[+] Processing image
[+] Sending image
127.0.0.1 - - [20/Apr/2023 22:41:30] "POST /creation HTTP/1.1" 200 3199638
[+] Receiving image
[+] Getting data from image
[+] Verifying the time stamp
[+] Time Stamp valid
[+] Extracting the signature
[+] Verifying the signature
[+] The signature is valid
/home/projectx/.local/lib/python3.10/site-packages/bottle.py:1759: ResourceWarning: unclosed file <_io.BufferedReader name='/tmp/qrcode-verif.png'>
rv = callback(*a, **ka)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
127.0.0.1 - - [20/Apr/2023 22:42:45] "POST /verification HTTP/1.1" 200 6

h1 @ibash* 1125b h3 23:16 20-Apr-23
```

5 Exemple d'exécution (certificat invalide)

Le certificat soumis à la validation est un certificat dont les données cachées ont été modifiées (paul à la place de toto), le serveur nous renvoie bien **"Non Valide"**



```
ResourceWarning: Enable tracemalloc to get the object allocation traceback
[+] Receiving image
[+] Getting data from image
[+] Verifying the time stamp
[+] Time Stamp valid
[+] Extracting the signature
[+] Verifying the signature
[+] The signature is not valid, exiting ...
/home/projectx/.local/lib/python3.10/site-packages/bottle.py:1759: ResourceWarning: unclosed file <_io.BufferedReader name='/tmp/qr-code-verif.png'>
  rv = callback(*a, **ka)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
127.0.0.1 - - [21/Apr/2023 16:25:40] "POST /verification HTTP/1.1" 200 10
/home/projectx/.local/lib/python3.10/site-packages/bottle.py:1683: ResourceWarning: unclosed file <_io.BufferedReader name='5'>
  def fset(self, value): ls.var = value
ResourceWarning: Enable tracemalloc to get the object allocation traceback
[+] Receiving image
[+] Getting data from image
[+] Verifying the time stamp
[+] Time Stamp valid
[+] Extracting the signature
[+] Verifying the signature
[+] The signature is not valid, exiting ...
/home/projectx/.local/lib/python3.10/site-packages/bottle.py:1759: ResourceWarning: unclosed file <_io.BufferedReader name='/tmp/qr-code-verif.png'>
  rv = callback(*a, **ka)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
127.0.0.1 - - [21/Apr/2023 16:25:55] "POST /verification HTTP/1.1" 200 10

Note: Unnecessary use of -X or --request, POST is already inferred.
* Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> POST /verification HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.81.0
> Accept: */*
> Content-Length: 3199952
> Content-Type: multipart/form-data; boundary=-----b58228d48043aa70
> Expect: 100-continue
>
* Done waiting for 100-continue
* We are completely uploaded and fine
* Mark bundle as not supporting multiuse
* HTTP 1.0, assume close after body
* HTTP/1.0 200 OK
< Date: Fri, 21 Apr 2023 14:25:55 GMT
< Server: WSGIServer/0.2 CPython/3.10.6
< Content-Length: 10
< Content-Type: text/html; charset=UTF-8
<
Not valid
* Closing connection 0
```

Figure 2: modification des métadonnées du certificat

6 Analyse de risques

Supposons que nous souhaitions réellement mettre en place cette architecture, on pourrait penser que le maillon faible se trouve être le serveur frontal étant donné que les certifications fournies sont protégées contre toute falsification et que notre serveur principal n'est pas exposé sur internet (on pourrait supposer qu'il ne possède que le port 8080 ouvert et accessible uniquement depuis le serveur frontal).

Il est vrai qu'il joue un rôle important avant la mise en place de la connexion, cependant, une fois celle-ci établie, il ne joue un rôle que secondaire (permettant le filtrage de certaines requêtes).

C'est pourquoi la sécurité de notre application ne doit pas reposer sur la sécurité fournie par le serveur frontal, dans le cas où celui-ci soit corrompu ou que des requêtes malveillantes aient pu passer au travers nous devons être capable de fournir une sécurité adéquate à notre serveur principal.

Actifs primaires à protéger

- les données présentes sur le serveur (clés privées et/ou données des utilisateurs si jamais nous décidons d'en collecter)
- le serveur principal

Actifs Secondaires à protéger

- le réseau reliant le serveur frontal au serveur principal

- le serveur frontal

Menaces sur ces biens

- tout type d'attaque informatique (DoS, Malware, Phishing)
- les erreurs humaines (erreurs de configuration, perte des données, inattention, arnaques ...)
- les dangers naturels et/ou matériels (inondation, appareils de stockage arrivés en fin de vie ...)

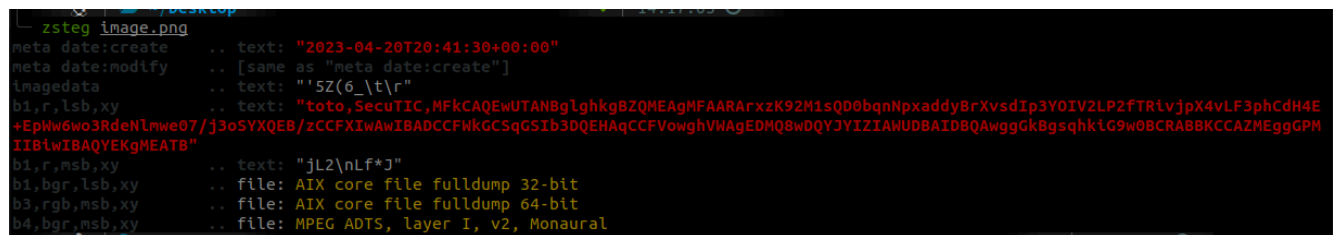
Toutes ces menaces représentent un réel danger sur les actifs que nous avons à protéger, les cyberattaques peuvent compromettre les données stockées ainsi que les applications s'exécutant sur le serveur, les erreurs humaines entraîner la perte de données et/ou la corruption de fichiers importants, les catastrophes naturelles endommager physiquement les serveurs, ordinateurs et périphériques utilisés par la société.

Il est donc important de mettre en place des mesures de sécurité robustes afin de protéger ces actifs, telles que la mise en place d'un pare-feu, l'utilisation de mots de passe forts, la mise en place de sauvegardes régulières et la formation des employés sur les bonnes pratiques en matière de sécurité informatique.

6.1 Faille de l'application

Lors de notre développement nous nous sommes rendus compte d'une faille dans l'application présentée :

Supposons que nous ayons un certificat valide en notre possession, il nous est alors possible de récupérer le message caché par stéganographie assez aisément sans avoir besoin de connaître l'algorithme utilisé à l'aide d'outils tels que zsteg :



```

zsteg image.png
meta date:create .. text: "2023-04-20T20:41:30+00:00"
meta date:modify .. [same as "meta date:create"]
imagedata .. text: "'5Z(6\t\r"
b1,r,lsb,xy .. text: "toto,SecuTIC,MfKCAQEwUTANBglghkgBZQMEAgMFARARxzK92M1sQD0bqnNpxaddyBrXvsdIp3Y0IV2LP2fTRlvjpX4vLF3phCdH4E
+EphW6wo3RdeNlme07/j3oSYXQEB/zCCFXIwAwIBADCCFNkGCSqGSIb3DQEHAQCCFVowghVWAgEDMQ8wDQYJYIZIAWUDBAIDBQAwggGkBgsgqhklG9w0BCRABBKCCAAMEggGPM
IIB1wIBAQYEkMEATB"
b1,r,msb,xy .. text: "jL2\nLf*J"
b1,bgr,lsb,xy .. file: AIX core file fulldump 32-bit
b3,rgb,msb,xy .. file: AIX core file fulldump 64-bit
b4,bgr,msb,xy .. file: MPEG ADTS, layer I, v2, Monaural
  
```

Figure 3: Obtention des données cachées

Rien ne nous empêche dorénavant de modifier le time stamp avec les données extraites cachées dans l'attestation ou avec des données choisies. Celle-ci resterait valide étant donné que notre serveur vérifie uniquement que les fichiers tsr et tsq soit bons.

Nous ne sommes pas capables de modifier le nom/prénom ou encore l'intitulé auquel le certificat a été délivré à cause de la signature présente dans le QR code mais nous sommes tout du moins capables de demander un nouveau time stamp et de changer ce dernier de manière totalement externe en gardant notre certificat valide.

La procédure est la suivante :

- Obtenir un certificat valide
- extraire le message caché par stéganographie
- générer un fichier tsq avec des données choisies
- quémander un fichier tsr auprès de Free TSA
- remplacer les anciens fichiers de time stamp par les nouveaux obtenus

- faire vérifier notre certificat

Le serveur va extraire les fichiers tsq et tsr présents dans le certificat reçu et vérifier leur validité auprès de Free TSA, ils le seront étant donné que c'est Free TSA qui les a délivrés.

6.2 Solutions

Nous avons trouvé plusieurs solutions afin de remédier à ce problème :

- Utiliser la date de délivrance et comparer cette dernière au time stamp fourni (nécessite que les deux dates soient équivalentes ce qui n'est pas forcément le cas (dépend de l'usage))
- ajouter le fichier tsr dans la signature du QR code afin de ne pas pouvoir le modifier
- garder dans une base de données les fichiers tsr associés au nom/prénom/titre des certifiés
- chiffrer les fichiers de time stamp afin d'empêcher leur modification (si modifiés ils ne seront plus valides après déchiffrement)

Chacune d'entre elles présente des avantages et des inconvénients, c'est au choix de la société de peser les pour et les contre afin de mettre en place la sécurité la plus optimale en fonction du budget alloué et du risque que représentent les failles.