ISEN
ALL IS DIGITAL!
yncréa

Interception TLS

Projet de Cryptographie

2021-2022

Florian PAUL, Guillaume BARREAULT

Sommaire

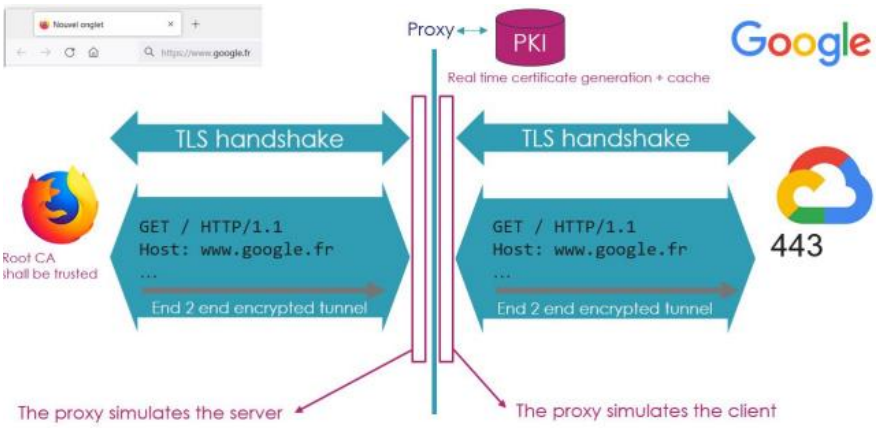
1. Présentation de l'interception TLS

a) Définition

De nos jours, le protocole HTTP (HyperText Transfer Protocol), est le protocole servant à la majorité des communications client-serveur sur Internet. Depuis maintenant quelques années, ce protocole a été complété avec un autre protocole, le TLS (Transport Layer Security). Le TLS est un protocole de chiffrement apportant des composantes de sécurité au HTTP telles que l'intégrité, la confidentialité et enfin l'authenticité aux échanges client-serveur. Le protocole TLS a donc permis de corriger le grand défaut du HTTP, c'est-à-dire la sécurité. Le protocole HTTP muni de TLS donne donc naissance au protocole HTTPS (HyperText Transfer Protocol Secure). Les communications sur le réseau ne sont donc plus affichées en clair et visibles par tout le monde. On utilisera une version ultérieure ou égale à la version 1.2 du protocole TLS, car les versions précédentes sont vulnérables à diverses attaques.

Le protocole HTTPS possède pour base le certificat. Ce certificat a pour but de prouver l'authenticité et l'intégrité de la clef publique du serveur, afin d'assurer au client qu'il s'adresse à la bonne personne ou au bon serveur. Cette clef sera notamment utile pour effectuer le handshake entre les deux parties, afin de créer une nouvelle clef symétrique secrète commune, qui permettra de chiffrer les échanges.

Maintenant, définissons ce qu'est l'interception TLS. Cette manipulation consiste à se positionner entre un client et un serveur tel un proxy, d'intercepter toutes les requêtes qu'ils pourraient s'échanger et de les déchiffrer. Ensuite, notre proxy se fait passer pour le serveur aux yeux du client. De même, le proxy va également se faire passer pour un client aux yeux du serveur. Puis le client va faire une demande de connexion, un handshake va être réalisé entre le client et le proxy, puis entre le proxy et le serveur que le client souhaite joindre.



Lorsque le client effectue une requête vers un nom de domaine, il attend en retour une réponse provenant du même nom de domaine. Si ce n'est pas le cas, la communication est impossible et ne se crée pas. Notre proxy doit donc utiliser un certificat ayant le même nom de domaine lors du handshake. Par conséquent, le proxy doit embarquer avec lui une autorité de certification racine (Root CA) qui a pour rôle de générer des certificats à la volée selon ce que demande le client comme nom de domaine. On devra placer le certificat de cette autorité racine dans le magasin de certificat du navigateur si on souhaite que le navigateur valide les certificats générés à la volée. L'utilisation de certificats auto-signés est impossible car le navigateur le détecte et estime qu'il ne peut pas lui faire confiance.

L'intérêt principal de l'interception TLS est le filtrage des données et la protection des systèmes d'informations d'entreprise, qui sont souvent vulnérables.

b) Avantages

Comme dit précédemment, l'intérêt de l'interception TLS se divise en plusieurs axes :

- Filtrage des requêtes pour gérer l'accès à certains sites
- Détection d'envoi de documents confidentiels par exemple, ou du moins à ne pas publier
- Détection d'une potentielle injection de malware grâce à un travail d'analyse de fichiers fait par le proxy

c) Inconvénients

L'interception TLS est un processus ingénieux qui possède de nombreux avantages. Cependant, il implique également des inconvénients par son principe. En effet, il est généralement mis en place par une entreprise sur son propre réseau mais rien n'empêche qu'un attaquant puisse le mettre en place.
Premièrement, l'attaquant pourrait lire tout le contenu des requêtes de la victime. Donc, l'aspect confidentialité serait effacé. Deuxièmement, les victimes en général ne consultent pas la configuration de leur navigateur sur leur ordinateur ou celui de l'entreprise, donc il est très peu probable qu'elles se rendent compte de l'attaque qu'elles subissent.
Enfin, le proxy mis en place nécessite une configuration précise, possiblement faite avec une blacklist ou whitelist de sites précis. S'il y a une faille dans la configuration du proxy, tout le processus ne serait pas très utile, et même donnerait un faux sentiment de sécurité.

2. Autorité de certification

Nous avons une autorité de certification racine qui possède une paire de clef privé/publique.

Nous avons créé notre rootCA grâce à OpenSSL. Celle-ci possède un certificat auto-signé, et nous devons donc l'insérer dans la liste des certificats de notre navigateur.

La clef privée de la rootCA est stockée dans un fichier où les accès sont limités au propriétaire du fichier et il ne peut que l'ouvrir en lecture. On utilise simplement un chmod 400.

C'est la rootCA qui va avoir la charge de générer des certificats pour les sites visités par l'employé qu'elle signera avec sa clef privée. Plus précisément, ce ne sont pas des certificats pour les sites mais pour les noms de domaines qui sont demandés. Le navigateur pourra, dans son magasin, trouver la clef publique de la root CA et vérifier la signature du certificat que le proxy a envoyé.

Voici le script que nous avons créé afin de générer notre rootCA.

```
# private key of CA ⇒ must be protected; can sign new cert.
# create Kpr_CA:
openssl genrsa -out CA-ROOT.key

# request signature from Kpr_CA:
openssl req -new -key CA-ROOT.key -out CA-ROOT.csr -sha256

# generate cert from Kpr_CA: (auto_sign)
openssl x509 -req -days 365 -in CA-ROOT.csr -out CA-ROOT.crt -signkey CA-ROOT.key

# we have 3 files:
# CA-ROOT.key ⇒ must be protected
chmod 400 CA-ROOT.key
# CA-ROOT.crt
# CA-ROOT.csr ⇒ we can remov
rm -fv CA-ROOT.csr

# create config file of CA:
vim ca-ssl.conf

# add 2 files:
mkdir indexs
cd indexs
touch index.txt     # incremente cert number
cd ..
mkdir serials
cd serials
echo '01' > serial  # keep trace of each cert edit
cd ..

~
~
~
"create_RootCA.sh" 31L, 686B                                    31,0-1
```

Quelques explications de notre script :

Nous utilisons l'algorithme RSA avec une paire de clef privé/publique de taille 2048 bits car c'est le plus commun sur OpenSSL et que cela assure une sécurité suffisante.

L'algorithme RSA est un algorithme de chiffrement basé sur la factorisation de grand nombres premiers en utilisant les congruences sur les entiers et le petit théorème de Fermat.

Cet algorithme utilise des fonctions fonctionnant avec une paire clef publique/clef privé, par conséquent, il est impossible de retrouver l'entrée grâce au résultat, puisqu'on chiffre avec la clef publique et on déchiffre avec la clef privée.

L'algorithme RSA est qualifié de sécurisé car il repose sur le fait que pour réussir à le casser et obtenir la clef privée, il faut réussir une factorisation en grands nombres premiers d'un entier n de grande taille. Ce type de factorisation est à ce jour un des plus grands problèmes mathématiques qui existent. En effet, comme on a choisi des nombres aléatoires premiers très grands, l'opération prendrait des ressources matérielles mais également un temps de calcul exorbitants. Comme en 25 ans de travail, personne n'a réussi à casser l'algorithme pour l'instant, il est considéré comme sécurisé. Il est recommandé d

Par ailleurs en 25 ans malgré les tentatives pour casser le RSA aucune n'y ait parvenu.
La taille de clé recommandée pour le RSA est 2048 bits.

La puissance de calcul des machines ne cesse continuellement d'augmenter, ce qui va permettre à l'algorithme de générer des clefs plus grosses apportant une sécurité plus élevée, rendant la tâche des attaquants plus compliqué pour casser les clefs.

Les certificats utilisés pour les connexions sont signés par notre autorité de certification racine.

Ils sont générés à la volée dans notre code source. Les paires de clefs sont généré avec le protocole de chiffrement RSA et ont une taille de 2048 bits. Les certificats quant à eux sont en version V3 avec une signature en SHA256.

3. Implémentation

a) Langages étudiés

Pour réaliser notre projet d'interception TLS, nous avons regardé plusieurs langages de programmation. Nous avons d'abord essayé d'implémenter ça en Python et Java, puisqu'ils sont fournis de librairies dédiées à la communication client/serveur. Cependant, nous sommes restés bloqués au niveau du proxy http sur ces deux langages et le passage en https n'a pas pu être fait. Nous avons fini par implémenter en Rust car c'est un langage sécurisé et que l'on connaît. Mais également, le Rust est un langage qui marche sur tout OS donc cela permettra à notre proxy de fonctionner sur n'importe quel système d'exploitation.
Enfin, le Rust nous est toujours recommandé par monsieur Paillard car il est tant performant que sécurisé puisqu'il permet une très bonne gestion de la mémoire et des erreurs.

b) Librairies utilisées

Voici la liste des librairies que nous avons utilisées et pourquoi :

- OpenSSL a été utilisé pour générer notre rootCA, les certificats et les paires de clé privé/publique
- Tokio pour faire fonctionner les fonctions asynchrones
- Hyper nous sert pour gérer toutes les requêtes header, body et http
- Hudsocker est utilisé pour ouvrir les sockets client/proxy et proxy/serveur

Cargo.toml

```
[package]
name = "proxy"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
openssl = { version = "0.10.38", features = ["v110"] }
hudsucker= {version = "0.16.1", features = ["openssl-certs"]}

tokio = { version = "1.6.1", features = ["macros","io-util", "rt","rt-multi-thread"] }
tokio-rustls = "0.23.0"
tokio-tungstenite = { version = "0.17.0", features = ["rustls-tls-webpki-roots"] }

actix-web = { version = "4", features = ["openssl"] }
actix= "*"
reqwest ="*"
env_logger= "0.9.0"
futures= "0.3.21"

tracing = { version = "0.1.21", features = ["log"] }
time = { version = "0.3.1", optional = true }

tracing-subscriber= "0.3.9"
```

c) Détail du code

Ce code est donc exécuté après le script présenté précédemment et donc la rootCA créée.
On récupèrera le certificat et la clé privée de la rootCA.
On configure un objet représentant notre rootCA avec ces trois lignes

```
let private_key: PKey<Private> = PKey::private_key_from_pem(private_key_byte).expect(msg: "Failed to parse private_key");
let rootca_cert: X509 = X509::from_pem(rootca_cert_byte).expect(msg: "Failed to parse Root-CA certificate");

let ca: OpensslAuthority = OpensslAuthority::new(pkey: private_key, ca_cert: rootca_cert, hash: MessageDigest::sha256(), cache_size: 100_000);
```

De plus, on configure où notre proxy tournera et également où récupérer les données de la rootCA.

```
// constant settings proxy
let host_name: [u8; 4] = [127, 0, 0, 1];
let bind_port: u16 = 8080;

// get path private_key & rootCA_cert + convert in Byte
let private_key_byte: &[u8] = include_bytes!("../../CA/CA-ROOT.key");
let rootca_cert_byte: &[u8] = include_bytes!("../../CA/CA-ROOT.crt");
```

Ensuite, cela va nous servir de paramètre dans la configuration de notre proxy.

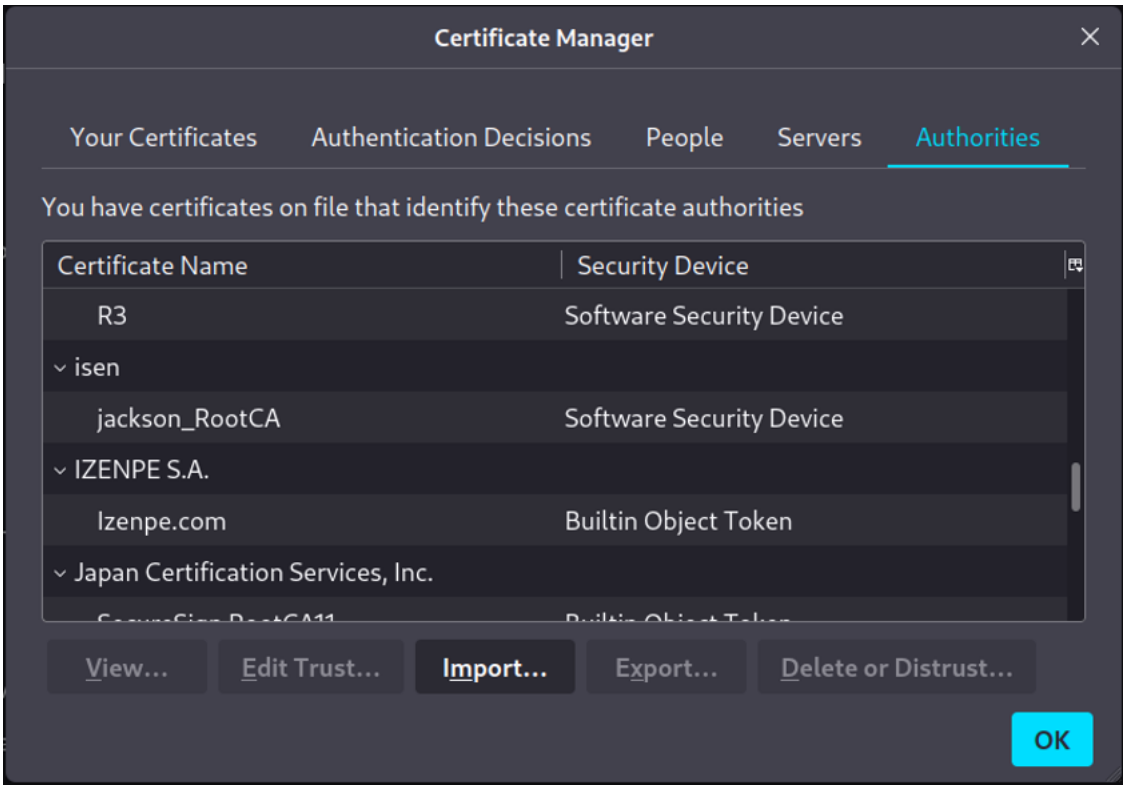
```
// configuration of proxy
let proxy: Proxy<HttpsConnector<HttpConnector>, …> = ProxyBuilder::new() ProxyBuilder<WantsAddr>
    .with_addr(SocketAddr::from((host_name, bind_port))) ProxyBuilder<WantsClient>
    .with_rustls_client() ProxyBuilder<WantsCa<HttpsConnector<…>>>
    .with_ca(ca) ProxyBuilder<WantsHandlers<…>>
    .with_http_handler(LogHandler {}) ProxyBuilder<WantsHandlers<…>>
    .build();
```

Maintenant que la base de notre proxy est construite, on va créer notre handler TLS qui fonctionne avec deux fonctions asynchrones : une qui récupère la requête et l'autre qui va envoyer la réponse à la partie client. Cela permettra de finaliser notre proxy.

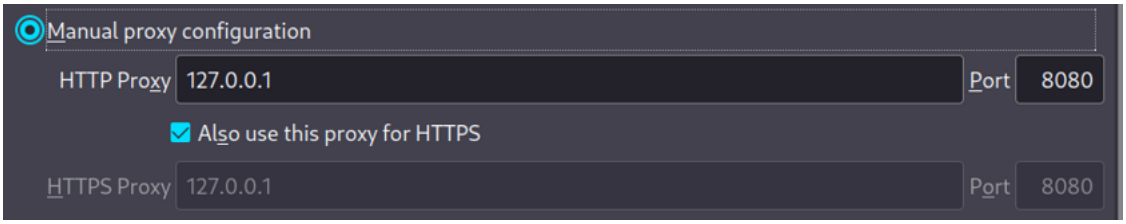
```
#[derive(Clone)]
2 implementations
struct LogHandler {}

#[async_trait]
impl HttpHandler for LogHandler {

    async fn handle_request(
        &mut self,
        _client_ctx: &HttpContext,
        request: Request<Body>,
    ) -> RequestOrResponse {
        println!("{:?}\n", request);
        println!("context: {:?}\n", _client_ctx);
        RequestOrResponse::Request(request)
    }
    async fn handle_response(&mut self, _client_ctx: &HttpContext, response: Response<Body>) -> Response<Body> {
        println!("handle response:\n{:?}\n", response);
        response
    }
}
```

d) Tests et résultats

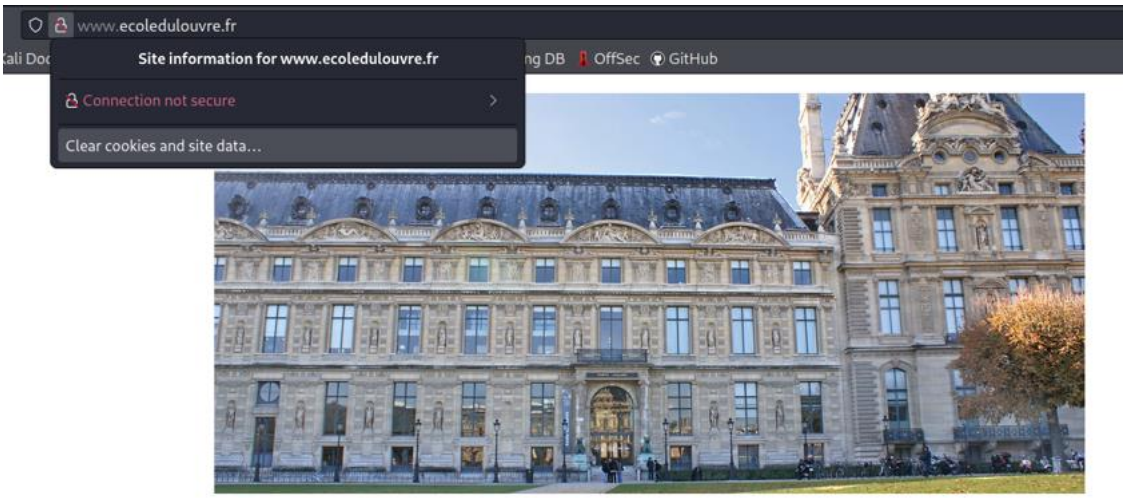
Tout d'abord, nous avons vérifié que le certificat de notre rootCA avait bien été ajouté au trust store de notre navigateur, c'est-à-dire la liste des certificats auxquels le navigateur fera confiance.



Nous vérifions également que le proxy est bien configuré sur notre navigateur (en l'occurrence Mozilla Firefox)

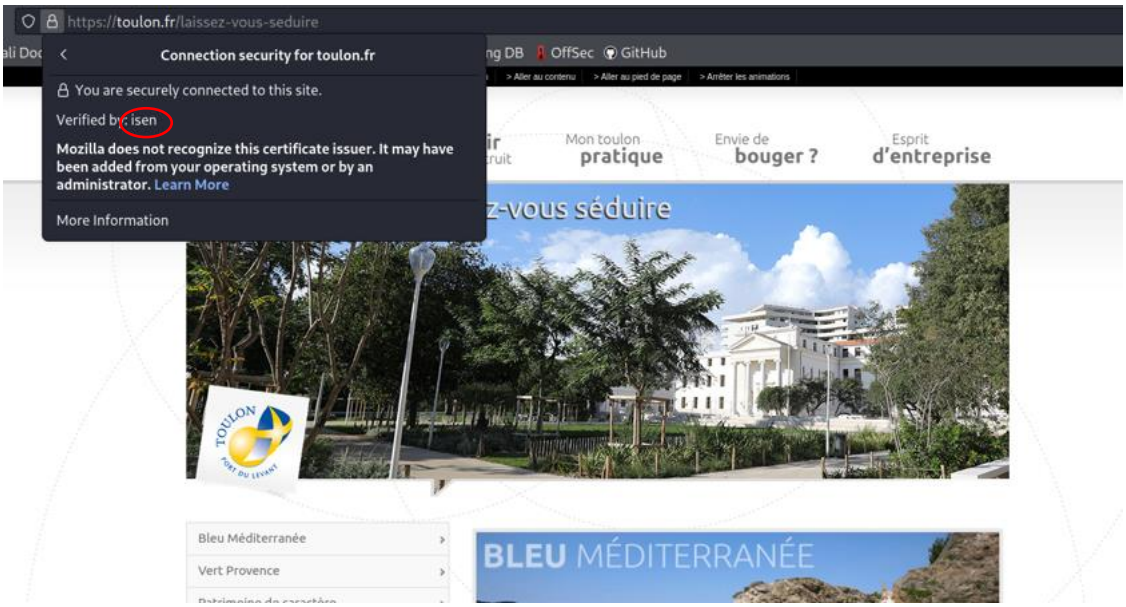


Maintenant que toutes les vérifications sont faites, nous pouvons tester notre intercepteur TLS.
En premier temps, nous testons sur le site http proposé en premier lors de notre recherche, c'est-à-dire http://ecoledulouvre.fr



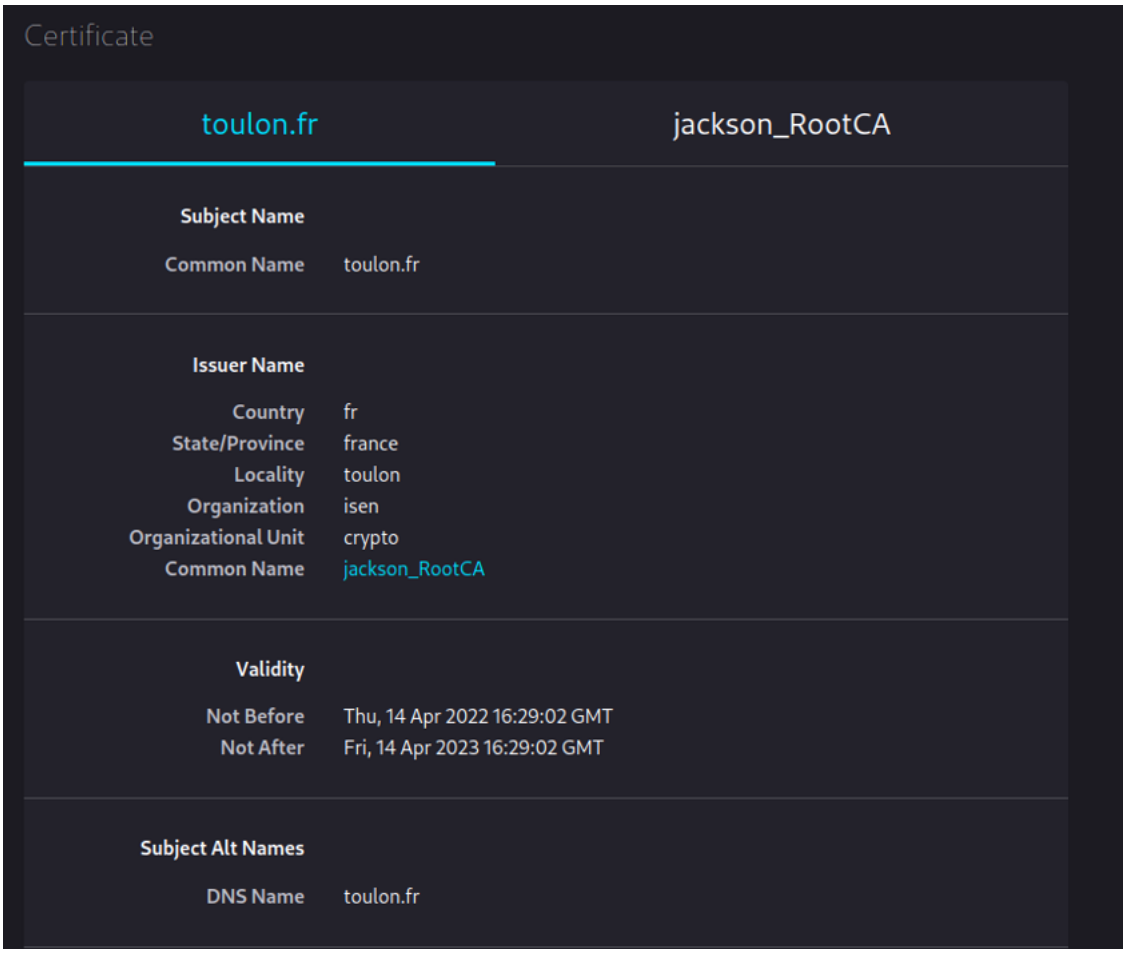
Le navigateur nous avertit de l'absence d'HTTP mais la page s'affiche sans problème malgré le proxy : c'est une réussite.

Enfin, le principal objectif du projet est de faire fonctionner notre proxy également en HTTPS.



Cela marche très bien sur le site https://toulon.fr, on peut bien voir que c'est notre certificat qui a été utilisé pour la connexion et que malgré ça, la page s'affiche parfaitement.

On le vérifie sur le certificat généré et fourni pour toulon.fr, et on voit bien que tout est comme on le désirait.



De manière plus technique, on observe que l'handler TLS intercepte la requête et la réponse.

```
┌──(kali㉿kali)-[~/Documents/projet_crypto/Interception_TLS/proxy]
└─$ cargo run
   Compiling proxy v0.1.0 (/home/kali/Documents/projet_crypto/Interception_TLS/proxy)
    Finished dev [unoptimized + debuginfo] target(s) in 30.14s
     Running `target/debug/proxy`

proxy is starting

Request { method: GET, uri: https://toulon.fr/, version: HTTP/1.1, headers: {"host": "toulon
.fr", "user-agent": "Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0",
"accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8", "acc
ept-language": "en-US,en;q=0.5", "accept-encoding": "gzip, deflate, br", "connection": "keep
-alive", "cookie": "_ga=GA1.2.493818040.1649944720; _gid=GA1.2.194190184.1649944720; tarteau
citron=!analytics=wait!twitterwidgetsapi=wait!facebook=wait!facebooklikebox=wait!instagram=w
ait!linkedin=wait!twitter=wait!dailymotion=wait!youtube=wait", "upgrade-insecure-requests":
"1", "sec-fetch-dest": "document", "sec-fetch-mode": "navigate", "sec-fetch-site": "none", "
sec-fetch-user": "?1"}, body: Body(Empty) }

context: HttpContext { client_addr: 127.0.0.1:48928 }

handle response:
Response { status: 200, version: HTTP/1.1, headers: {"date": "Thu, 14 Apr 2022 16:29:03 GMT"
, "server": "Apache/2.4.38 (Debian)", "x-content-type-options": "nosniff", "x-content-type-o
ptions": "nosniff", "x-drupal-cache": "MISS", "expires": "Sun, 19 Nov 1978 05:00:00 GMT", "c
ache-control": "public, max-age=1800", "set-cookie": "SSESSb6b80b245f8e86f4e23e35bf467cb3a9=
SRCqJaXqBxN9n84NG_tOUG1E0G5Qvck3ZYzRPBh3Tws; expires=Fri, 15-Apr-2022 00:29:03 GMT; Max-Age=
28800; path=/; domain=.toulon.fr; secure; HttpOnly", "content-language": "fr", "x-frame-opti
ons": "SAMEORIGIN", "permissions-policy": "interest-cohort=()", "x-generator": "Drupal 7 (ht
tp://drupal.org)", "link": "<https://toulon.fr/>; rel=\"canonical\",<https://toulon.fr/>; re
l=\"shortlink\"", "etag": "\"1649953743-1\"", "last-modified": "Thu, 14 Apr 2022 16:29:03 GM
T", "vary": "Accept-Encoding", "content-encoding": "gzip", "keep-alive": "timeout=5, max=100
```

4. Difficultés rencontrées

a) Génération de certificat à la volée

La génération de certificat a été un sacré défi mais après plusieurs heures de recherche, nous avons réussi à le faire. Nous mettons finalement les certificats créés à la volée dans le cache à l'aide de la librairie OpenSSL.

b) Amélioration du HTTP au HTTPS

Le problème de la mise en place du https est qu'il est plutôt simple de créer un proxy http puisqu'il existe de nombreuses explications sur Internet. Cependant, une fois cette étape terminée, la création d'une rootCA n'est pas très complexe mais la création du certificat X-509 pour un site ne fonctionnait pas très bien pour nous.

5. Potentielles améliorations

Comme il a été dit dans la première partie du rapport, l'utilisation de l'interception TLS se fait généralement au sein d'une entreprise afin de filtrer les entrées sur le réseau et l'accès à certains sites. On pourrait donc penser à mettre en place une blacklist de sites qui seraient interdits aux employés mais nous pensons qu'une whitelist est plus adaptée à la situation puisque cette liste changera sera plus courte et moins souvent modifiée que la blacklist.

Nous pourrions également envisager d'augmenter la sécurité ou du moins augmenter la confiance que l'on peut accorder à notre rootCA.
Par exemple, il serait possible de créer une sous-CA afin de consolider la chaine de certification et donc la confiance que l'on donne à notre rootCA.

Cela nous permettrait de déléguer la tâche de génération de certificats à la sous-CA afin de ne pas exposer la clé privée de la rootCA qui est essentielle au bon fonctionnement du TLS. Généralement, il est conseillé d'avoir une sous-CA vérifiée par une rootCA, et de signer les certificats.

6. Conclusion

Pour conclure, grâce à l'utilisation du Rust et de ses librairies complètes, nous avons pu mettre en place un intercepteur TLS fonctionnel, agissant comme un proxy sans interférer dans la connexion HTTP ou HTTPS tout en récupérant le contenu des requêtes envoyées dans les deux sens.
Cela permet donc de donner un droit de regard sur ce que fait un potentiel employé et aussi filtrer l'accès à certains sites.

Cependant, l'utilisation de ce dispositif nécessite un cadre légal bien défini afin de respecter les règles imposées par la CNIL par exemple. En réalité, notre dispositif n'est rien d'autre qu'une attaque Man In The Middle si la cible n'est pas mise au courant et cela ne peut pas être mis de côté.
Nous devons l'utiliser uniquement dans un intérêt de sécurité pour le système d'une entreprise et surtout en dehors de ce cadre sinon nous nous retrouverons dans une situation illégale.

De son côté, l'ANSSI émet également des conditions à respecter dans le cadre de l'interception TLS. Ils estiment nécessaire que l'infrastructure qui met en place le dispositif doit savoir assurer une confidentialité sur les flux interceptés. Elle doit utiliser par exemple, un HSM pour assurer le stockage des clés.

Finalement, ce dispositif est un excellent outil s'il se retrouve dans de bonnes mains, qualifiées et que l'opération est totalement encadrée par une entreprise.

7. Bibliographie

Informations :

https://docs.rs/

https://github.com/hyperium/hyper

https://tokio.rs/tokio/tutorial

Images :

https://www.thesslstore.com/blog/what-is-a-certificate-authority-ca-and-what-do-they-do/

https://www.cisco.com/c/fr_ca/support/docs/security-vpn/public-key-infrastructure-pki/211333-IOS-PKI-Deployment-Guide-Initial-Design.html