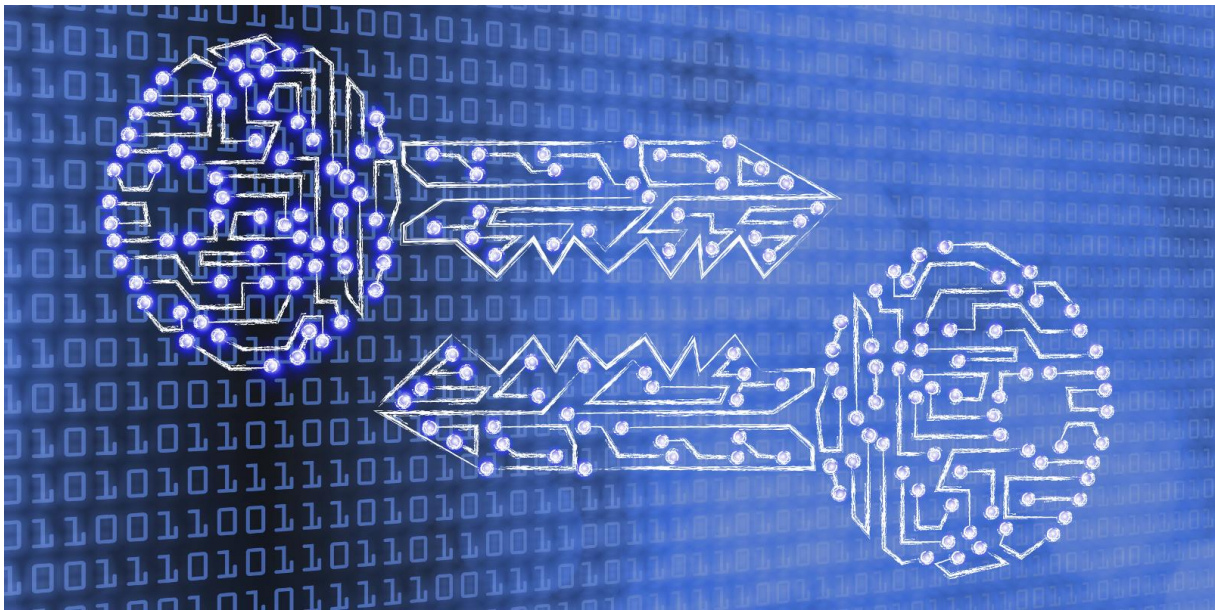


Mathieu GAUDE
Bastien BONGIORNO
Thibaut RIBE



Rapport du Projet Cryptographie
Du 05/03/22 au 15/04/22

Année 2021/2022
M1 Cybersécurité

SOMMAIRE

| | |
|------------------------------|----|
| Introduction..... | 2 |
| Intérêts..... | 3 |
| Inconvénients..... | 4 |
| Enjeux juridiques..... | 5 |
| Choix techniques..... | 6 |
| Difficultés rencontrées..... | 14 |
| Améliorations possibles..... | 16 |
| Ressources..... | 17 |
| Conclusion..... | 18 |

INTRODUCTION

L'interception TLS est un concept visant à intercepter la communication Internet chiffré TLS entre le client et le serveur. L'interception peut être effectuée dans les deux sens.

Dans ce cadre-là, nous avons été mandatés pour réaliser un proxy TLS permettant d'intercepter et de filtrer les recherches effectuées sur Internet.

INTERETS

L'intérêt principal de l'interception TLS est de séparer la communication en deux : le privé avant le proxy et le public après, tout en gardant l'aspect de sécurité proposé par le TLS.

De nos jours, les contenus malveillants sont de plus en plus cachés dans le trafic chiffré. Et parce qu'il est chiffré, il passe donc inaperçu aux yeux des mécanismes de sécurité basique. L'interception TLS peut être destinée à inspecter et à filtrer le contenu qui circule sur le proxy. On a, par exemple, l'analyse antivirus, du filtrage Web, du filtrage d'E-mail mais aussi de la restriction sur certains sites Web.

Cela peut permettre de détecter l'exfiltration de données. Cela correspond à de l'interception sortante.

Ce type d'architecture est très souvent utilisé dans les entreprises afin que chaque ordinateur ne soit pas connecté directement à Internet. Cela permet à l'entreprise de conserver une certaine confidentialité vis-à-vis de l'extérieur, de se protéger et de limiter les échanges directs avec de potentiels attaquants. Il permet également de garder une trace des requêtes effectuées par les employés. Cela peut s'avérer utile en cas d'attaque pour remonter à la source du problème. De plus, cela permet de surveiller les fuites de données concernant l'entreprise.

INCONVENIENTS

L'interception TLS peut injecter des vulnérabilités dans une connexion présumée sécurisée. En effet, cela est en fait une usurpation d'identité avec de bonnes intentions. Mais si les intentions sont mauvaises, il se pourrait très bien que certains certificats installés ne soient pas les bons et donc tout le processus de sécurité serait alors inutile.

De plus, la fiabilité du proxy n'est pas parfaite. Le navigateur et l'utilisateur deviennent "aveugles". Le navigateur ne sera plus en mesure d'avertir l'utilisateur des connexions HTTPS et l'utilisateur n'aura aucun moyen de voir si le certificat est valide.

La perturbation de l'utilisation personnelle est un inconvénient majeur. Les réseaux sociaux, les messageries Web, les banques en ligne, demandent à leurs utilisateurs de vérifier la connexion HTTPS. Donc dans ces cas-là, l'utilisateur est restreint.

Si l'entreprise garde une trace de toutes les requêtes effectuées, cela peut entraîner un problème par rapport à la confidentialité et la vie privée des employés et peut poser des débats éthiques quant à l'utilisation de ces données par l'entreprise.

ENJEUX JURIDIQUES

La forte croissance de l'utilisation du TLS dans les usages du quotidien empêche bon nombre d'organismes de procéder à des contrôles de sécurité satisfaisants. Or, le manque de contrôle sur les données en transit sur un réseau peut entraîner des répercussions juridiques importantes. L'interception TLS est donc un réel enjeu juridique sur le fonctionnement d'une structure.

Les enjeux juridiques principaux sont les suivants :

- **L'information**

Il est important que si la technique d'interception TLS est mise en pratique, les utilisateurs impactés soient avertis.

- **L'accès aux courriers électroniques**

Il est nécessaire de définir précisément des processus d'accès.

- **Minimisation des traces conservées**

Le contenu des messages échangés ne doit pas faire l'objet d'un stockage ciblé.

- **Une protection des données d'alertes extraites de l'analyse**

C'est-à-dire que tout résultat à la suite d'un traitement ne doit être accessible que par des personnes habilitées à voir ces résultats.

- **La protection des données personnelles**

L'interception TLS est un traitement et une manipulation de données. Par conséquent, son utilisation non déclarée ni encadrée peut entraîner une peine de cinq ans de prison et de 300 000 euros d'amende.

- **Obligation légale de déchiffrement lors d'enquête judiciaire**

De plus, faire de l'interception TLS sans le consentement des utilisateurs revient à réaliser une attaque Man In the Middle.

D'après la CNIL : "l'utilisation de l'interception TLS est considérée comme légitime si elle intervient dans un intérêt de sécurité pour l'organisme tout en étant transparent avec ses salariés."

CHOIX TECHNIQUES

Au début de notre projet, nous avons hésité entre NodeJS et Python car NodeJS paraissait plus abordable pour la création d'un proxy. En effet, les bibliothèques que l'on a utilisées étaient de plus haut niveau, ce qui nous a permis de créer un serveur proxy sans écrire un code trop long. Nous avons pu tester notre proxy sur des sites HTTP.

Pour le réaliser, nous avons utilisé les librairies suivantes :

'http' qui est un module de NodeJS qui nous permet de créer un serveur HTTP et d'effectuer des requêtes.

'url' qui nous permet de changer le type de l'URL récupéré. De base, nous récupérons un string et on le transforme en objet de type url afin de mieux réaliser les requêtes par la suite.

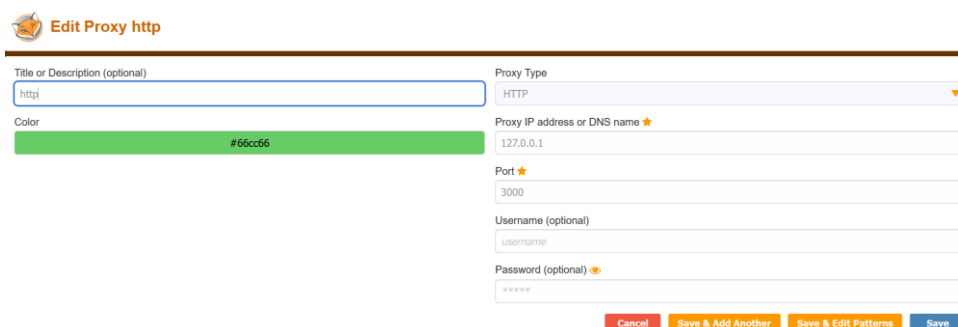
'colors' pour pouvoir faire des console log plus lisible en y rajoutant des couleurs.

Pour réaliser le proxy HTTP nous avons créé un serveur HTTP qui écoute les requêtes sur le port 3000.

```
// On crée un serveur http qui prend en option la méthode pour effectuer l'interception
const server = http.createServer(parseIncomingRequest);

// On écoute sur le port 3000
server.listen(PORT, () => {
  console.log(`***** PROXY STARTED ON http://localhost:${PORT} *****\n`);
});
```

Quand un utilisateur fait une requête, nous le redirigeons automatiquement sur le port 3000 en modifiant les paramètres proxy du navigateur. Pour simplifier nous avons utilisé l'extension foxy-proxy.



Cette requête est récupérée par notre serveur qui va en extraire l'URL.

```
//méthode pour récupérer la requête émise par l'utilisateur
const parseIncomingRequest = (clientRequest, clientResponse) => {
  //On récupère l'url de la requête émise par l'utilisateur
  const requestToFulfil = url.parse(clientRequest.url);

  // Options et infos utiles pour envoyer la requêtes au site
  const options = {
    method: clientRequest.method,
    headers: clientRequest.headers,
    host: requestToFulfil.hostname,
    port: requestToFulfil.port || 80,
    path: requestToFulfil.path
  }
}
```

A ce moment-là, il est possible de bloquer la requête si l'url est blacklist.

```
//On peut blacklist certaines adresses ou bloquer certaines ressources
if (clientRequest.url == "http://www.ens-lyon.fr/") {
  //Si on remarque des ressources que l'on veut bloquer
  //On n'autorise pas la requête
  options.allowed = false;
  logger(options);
  //Et on la termine
  clientResponse.end();
} else {
  //Sinon on accepte
  options.allowed = true;
  logger(options);

  //on exécute la requête
  executeRequest(options, clientRequest, clientResponse);
}
```

Sinon le serveur va effectuer la requête et renvoyer les données reçues à l'utilisateur.

```

//méthode qui exécute la requête et la renvoie à l'utilisateur
const executeRequest = (options, clientRequest, clientResponse) => {
  //On utilise la librairie http et sa méthode request pour faire une requête vers un site http
  const externalRequest = http.request(options, (externalResponse) => {

    // On écrit le header
    clientResponse.writeHead(externalResponse.statusCode, externalResponse.headers);

    //On renvoie les données reçu à l'utilisateur
    externalResponse.on("data", (chunk) => {
      clientResponse.write(chunk);
    });

    // Quand la le site a fini de renvoyer les données on fini l'envoi des données au client.
    externalResponse.on("end", () => {
      clientResponse.end();
    });
  });

  // Map data coming from client request to the external request being made
  clientRequest.on("data", (chunk) => {
    externalRequest.write(chunk);
  });

  // On termine la requête externe quand la requête du client est aussi fini pour être sûr d'effectuer toutes les requêtes
  clientRequest.on("end", () => {
    externalRequest.end();
  });
}

```

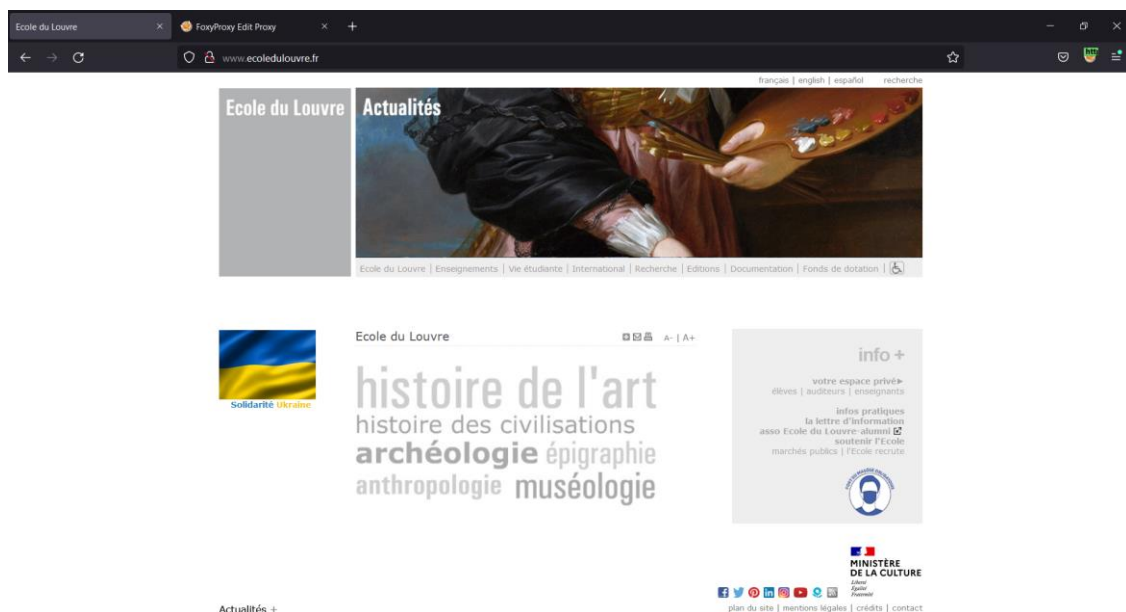
Il est également possible de bloquer certaines ressources comme des images qui ne seront pas renvoyées. Toutes les requêtes et les ressources renvoyés sont log dans la console.

```

Secure server on port 3443
GET : http://www.ecoledulouvre.fr/ - ALLOWED
GET : http://www.ecoledulouvre.fr/misc/jquery.js?D - ALLOWED
GET : http://www.ecoledulouvre.fr/misc/jquery-extend-3.4.0.js?D - ALLOWED
GET : http://www.ecoledulouvre.fr/misc/jquery-html-prefilter-3.5.0-backport.js?D - ALLOWED
GET : http://www.ecoledulouvre.fr/misc/drupal.js?D - ALLOWED
GET : http://www.ecoledulouvre.fr/modules/node/node.css?D - ALLOWED
GET : http://www.ecoledulouvre.fr/sites/default/files/languages/fr_69f38c39e623dd1434766f1beac5c44d.js?D - ALLOWED
GET : http://www.ecoledulouvre.fr/sites/all/modules/dhtml_menu/dhtml_menu.js?D - ALLOWED
GET : http://www.ecoledulouvre.fr/modules/system/defaults.css?D - ALLOWED
GET : http://www.ecoledulouvre.fr/sites/default/themes/ecoledulouvre/js/edl.js?D - ALLOWED

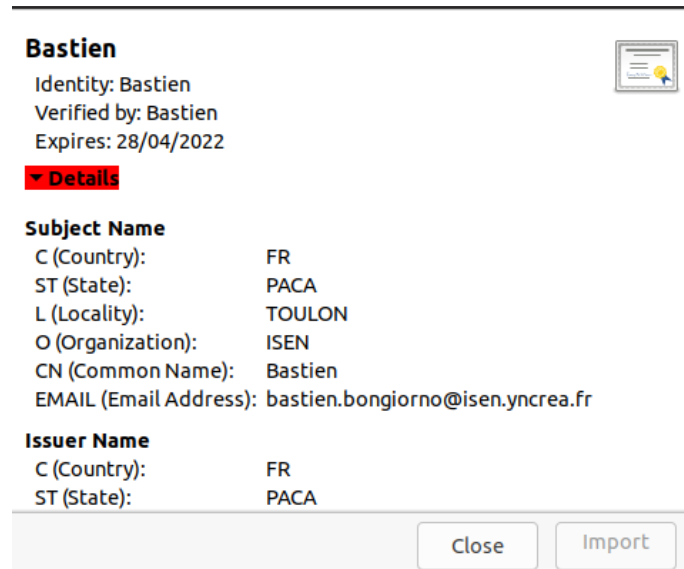
```

Voilà le résultat final du proxy HTTP :



Ensuite nous avons créé une Root CA et une Sub CA.

Nous avons choisi d'utiliser des clés RSA 2048 bits car nous avons estimé que la protection requise ne nécessite pas de clé 4096 bits en RSA.



Nous avons, en revanche, lors de la création de notre petite PKI, signé les certificats et aussi mis un mot de passe pour ouvrir les clés privées.

Unlock: cakey.pem

The contents of "cakey.pem" are locked. In order to view the contents, enter the correct password.



Nous avons fait tout d'abord notre Root CA puis nous avons créé un certificate request et la génération d'une nouvelle clé privée pour le Sub CA. Nous avons choisi d'utiliser du PKCS#10.

Grâce à cette génération de ce certificate request, nous avons pu générer le certificat pour le Sub CA.

Bastien

Certificate request

Identity: Bastien



▼ Details

Subject Name

C (Country): FR
ST (State): PACA
L (Locality): TOULON
O (Organization): ISEN
CN (Common Name): Bastien
EMAIL (Email Address): bastien.bongiorno@isen.yncrea.fr

Certificate request

Type: PKCS#10
Version: 1

Au niveau de la politique, nous avons demandé que le certificat de Sub CA soit bien signé uniquement s'il a les mêmes données suivantes :

- Nom du pays
- Etat
- Localisation
- Organisation
- Nom
- adresse mail

Nous considérons que la création d'un intercepteur TLS est très sensible donc nous avons choisi d'implémenter une politique stricte.

Nous avons ensuite créé un serveur https avec nos certificats.

```
//On crée un serveur https avec une clé et un certificat qu'on va lire directement grâce à la librairie fs
const sslServer = https.createServer(
  {
    key: fs.readFileSync(path.join(__dirname, 'cert', 'key.pem')),
    cert: fs.readFileSync(path.join(__dirname, 'cert', 'cert.pem')),
  },
  //On utilise la même méthode que précédemment pour gérer les requêtes
  parseIncomingRequest
)
//on écoute sur un port différent que celui du http
sslServer.listen(3443, () => console.log('Secure server 🚀🔒 on port 3443'))
```

Nous avons décidé de changer de langage et d'utiliser du python car nous n'arrivions pas à faire des certificats dynamiquement avec NodeJS.

A l'aide de certains tutoriels sous python, nous avons compris comment récupérer un certificat et le forward. Nous avons donc décidé de continuer l'HTTPS sous Python.

En ce qui concerne les librairies nous n'en avons pas beaucoup utilisées.

En effet, nous avons fait le choix d'utiliser les librairies basiques qui permettent de mettre en place un serveur proxy et d'utiliser les tâches basiques en réseau. Nous avons, par exemple, socket pour la création de sockets, sys pour interagir avec l'interpréteur python, ainsi que la bibliothèque thread pour exécuter des tâches en parallèles.

```
import socket, sys, datetime, time
from _thread import start_new_thread
```

Nous avons, au début, utilisé un serveur proxy HTTPS basique avec du certificat forwarding.

Ce code comprend l'initialisation du serveur, un service de listening et de read, un service de gestion du serveur.

Grâce à la bibliothèque socket, on peut facilement communiquer sur une interface réseau.

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((webserver, port))
s.send(request)
```

Nous avons, par la suite, implémenté les logs pour que chaque action apparaisse en direct dans le cmd. Puis, nous avons stocké ces informations dans un fichier log.txt.

```
# Function to write log
def write_log(self, msg):
    with open("log/log.txt", "a+") as file:
        file.write(msg)
        file.write("\n")
```

En pratique, ce fichier doit être protégé et n'être accessible que par les admins. C'est un cas de politique de confidentialité très complexe. Dans

notre cas, et pour ce projet nous avons décidé de ne pas protéger les logs. Enfin, nous avons implémenté une fonction permettant de bloquer certaines IP et certains domaines. Pour faire ceci, on peut mettre en paramètre les IP et domaines à bloquer et tout simplement stopper la connexion lorsque le service de listening reçoit une requête provenant de cet IP ou ce domaine.

Ci-dessous l'exemple pour le blocage d'IP :

```
# Checking for blacklisted ips
if addr[0] in self.blacklisted_ip_lookup:
    print(self.getTimeStamp() + "    IP Blacklisted")
    self.write_log(self.getTimeStamp() + "    IP Blacklisted")
    conn.close()
```

Nous avons aussi mis en place un système de cache qui permet de sauvegarder une page qui a déjà été cherchée afin de réduire les coûts lors des prochaines requêtes.

```
# Trying to find in cache
try:
    print(self.getTimeStamp() + "  Searching for: ", requested_file)
    print(self.getTimeStamp() + "  Cache Hit")
    file_handler = open(b"cache/" + requested_file, 'rb')
    self.write_log(self.getTimeStamp() + "  Cache Hit")
    response_content = file_handler.read()
    file_handler.close()
```

Le `requested_file` est le fichier stocké dynamiquement dans le sous dossier `cache` afin de sauvegarder le cache.

Cela est pratique mais cette implémentation rajoute une surface d'attaque. En effet, les données stockées dans notre cache peuvent être des données sensibles. Dans notre projet, nous n'avons pas protégé le cache mais en réalité, toutes les données mises en cache contenant des identifiants de session par exemple doivent être supprimées.

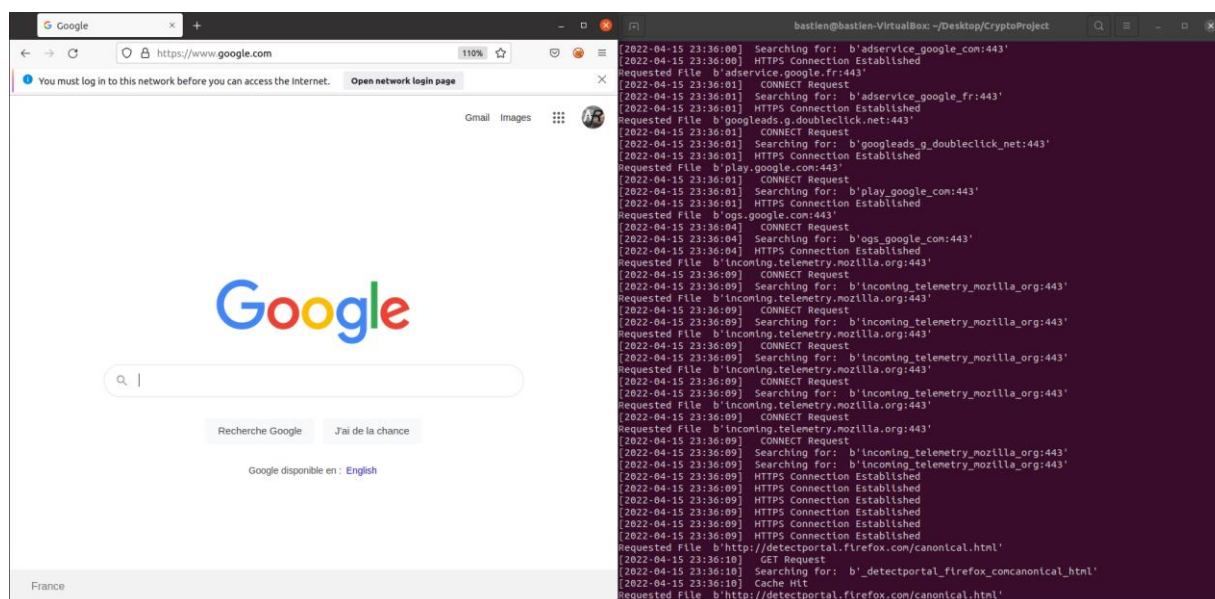
Au niveau de la connexion HTTPS, on a donc récupéré le certificat dynamiquement du site web recherché puis on a forward l'entièreté des informations à l'aide d'un buffer.

```
while True:
    try:
        request = conn.recv(buffer_size)
        s.sendall(request)
    except socket.error as err:
        pass

    try:
        reply = s.recv(buffer_size)
        conn.sendall(reply)
    except socket.error as e:
        pass
```

Voici le résultat final quand on lance notre proxy Https

```
bastien@bastien-VirtualBox: ~/Desktop/CryptoProject$ python3 https-proxy.py  
[2022-04-15 23:34:37] Listening...
```

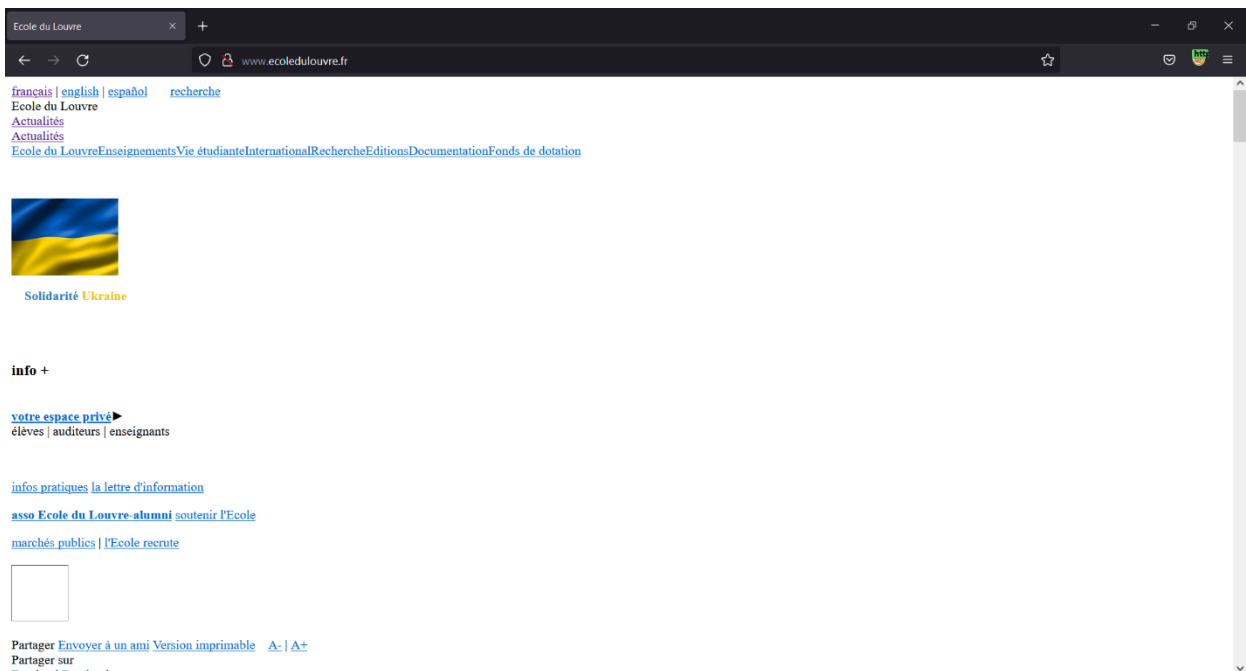


On peut voir que, dès qu'une requête est réalisée, le proxy va chercher dans le cache. Si le cache est présent alors il va l'utiliser, sinon il va établir la connexion normalement. On a aussi ajouté l'heure afin que les logs possèdent une empreinte qui permet d'effectuer des recherches.

DIFFICULTES RENCONTREES

Nous avons tout d'abord rencontré des difficultés pour choisir le langage de programmation. Nous avons, dans un premier temps, essayé en java mais nous nous sommes vite retrouvés limité par notre manque de connaissance dans ce langage. Thibaut étant plus familier avec le Javascript, nous avons continué la recherche dans ce langage en utilisant NodeJS.

La première difficulté rencontrée lors de la création du proxy HTTP a été lors du renvoi de la requête au client, nous ne renvoyions que l'HTML de la page sans le CSS et les images.



Nous avons donc changé de technologie pour créer le serveur. Au lieu d'utiliser express et de récupérer l'URL lors de la requête de cette manière :

```
var express = require('express');
var app = express();
var httpserv = require('http').createServer(app);
var request = require('request')
var fs = require("fs");
const url = require('url');
// input your url here

// use a timeout value of 10 seconds
var timeoutInMilliseconds = 10 * 1000

const http = require('http');

app.get('/', (req, res) => {
  console.log(req.url);

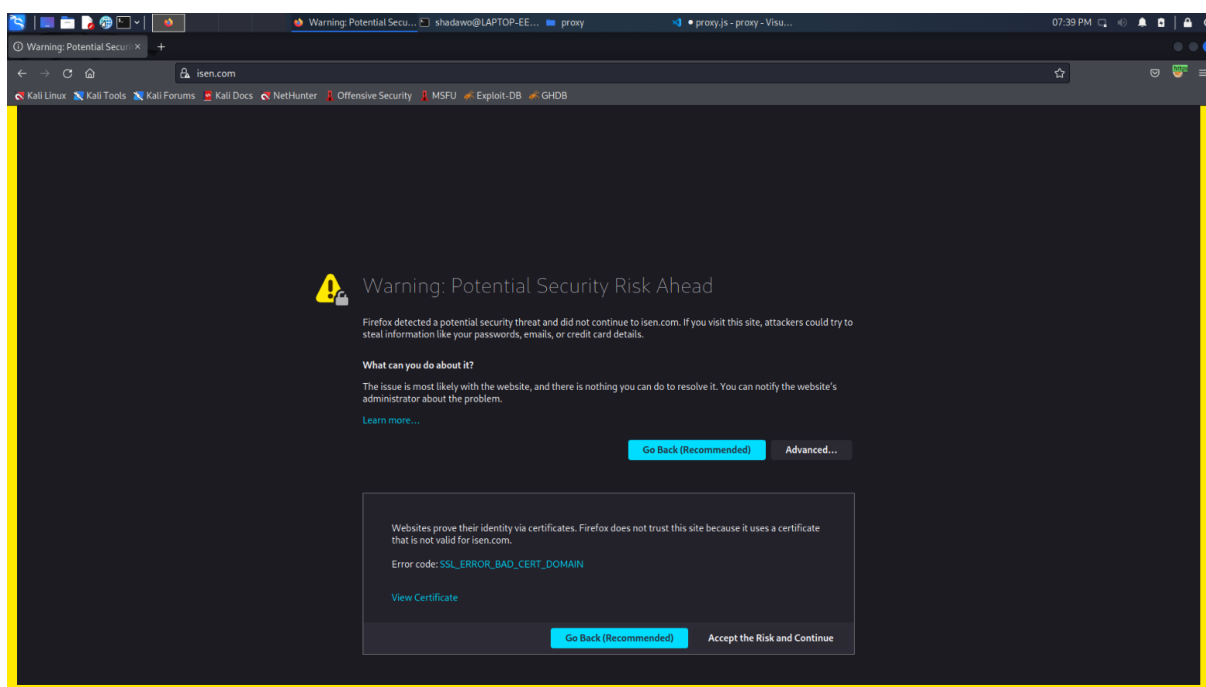
  const requestToFulfil = url.parse(req.url);
```

Nous créons directement un serveur HTTP avec, en paramètre, une fonction qui permet d'effectuer la requête initiale et d'envoyer la réponse à l'utilisateur.

```
// Create a HTTP server
const server = http.createServer(parseIncomingRequest);
```

Par suite de cela, nous avons réussi à avoir un proxy HTTP fonctionnel avec possibilité d'effectuer un middleware en bloquant, par exemple, certaines requêtes ou en effectuant du log de données.

Nous avons donc décidé de commencer la partie TLS avec NodeJS. Cependant, nous n'arrivions pas à inclure nos scripts SSL avec notre code existant. Nous n'avons ainsi que des certificats signés à l'avance et avec des informations ne correspondant pas au site sur lequel la requête était effectuée. Nous avons alors une erreur évidente de la part du navigateur.



Nous n'arrivions pas à signer des certificats directement dans le code et à les renvoyer.

À la suite de cela, nous avons décidé d'utiliser le python qui était mieux maîtrisé par les trois membres du groupe et mieux documenté sur ce point.

AMELIORATIONS POSSIBLES

Au niveau des améliorations possibles sur notre projet, la priorité pour nous est la protection des données et des logs des utilisateurs du proxy. Dans l'état actuel du proxy, les connexions et les logs des utilisateurs peuvent être récupérés par des attaquants grâce à une interception SSL. Pour éviter cela, nous devrions installer un paramètre d'anonymat en couplant le proxy à un VPN par exemple.

En attendant cette implémentation, une solution alternative pourrait être la mise en place d'une session administrateur pouvant avoir accès aux logs pour surveiller et prévenir en cas de fuite de données personnelles.

À la suite de ça, nous pensons à augmenter la taille de la clé RSA, actuellement du RSA 2048, pour augmenter la sécurité. De plus et comme dit précédemment, l'ensemble des logs n'est pas protégé donc nous envisageons de protéger le fichier en chiffrant les logs.

Ensuite, le proxy marche parfaitement sur le navigateur sur lequel nous l'avons implémenté. Nous devons donc le tester sur un autre navigateur voire sur une autre machine pour vérifier son intégrité.

Et pour terminer, nous souhaitons développer la création des certificats à la volée. Actuellement les certificats sont réalisés côté serveur puis transférés côté client, l'implémentation dynamique pour chaque site web est la prochaine étape pour cette fonctionnalité.

RESSOURCES

- <https://www.silicon.fr/la-cnile-encadre-le-dechiffrement-des-flux-https-113184.html>
- https://www.memoireonline.com/12/21/12533/m_L-interception-SSLTLS--le-fonctionnement-entre-enjeux-et-risques-les-bonnes-pratiques9.html
- <https://itsecworks.com/2010/11/22/create-your-own-ca-or-root-ca-subordinate-ca/>
- <https://www.helpnetsecurity.com/2017/03/08/https-interception-dilemma/>
- https://www.youtube.com/watch?v=USrMdBf0zcg&ab_channel=youRsTRULY
- <https://nodejs.org/api/http.html>
- <https://www.cnil.fr/fr/analyse-de-flux-https-bonnes-pratiques-et-questions>

CONCLUSION

Pour conclure, l'interception TLS possède de nombreux avantages mais aussi beaucoup d'inconvénient. Au niveau sécurité, nous avons des URL qui sont bloqués pour la protection des données d'une entreprise par exemple. Un utilisateur mal intentionné pourrait tromper l'URL Filtering.

Le premier scénario d'attaque serait de retirer l'extension SNI (Server Name Indication) du Client-Hello message. Le TLS Handshake est initialisé avec ce Client Hello sans SNI et donc envoyé un header HTTP avec la valeur host du site bloqué. De ce fait, la connexion va devenir chiffré et il sera impossible pour le firewall d'identifier le serveur demandé après le TLS Handshake.

Le deuxième scénario consiste à modifier l'extension SNI avec une fausse valeur mais qui ressemble à un vrai nom de domaine. Par exemple, g00gle.com

Les étapes de l'attaque sont les suivantes :

- Créer un socket TLS avec le nom de domaine et le port
- Configurer le socket TLS avec un SNI personnalisé où la valeur du nom du serveur est falsifiée, puis utiliser le socket pour se connecter à un site Web bloqué
- Envoyer l'en-tête d'hôte HTTP (chiffré) avec le champ d'hôte détenant la bonne adresse du site Web bloqué.

Ce projet nous a apporté à tous de nombreuses connaissances sur le sujet de la cryptographie et sur ses enjeux. Nous avons aimé travailler sur ce projet car il a été très enrichissant.

Nous vous remercions pour ce cours et ce projet