777-Team
Yohann LE CLECH
Jean-Christophe TURCAT
Paul-Emile NEU

Projet de cryptographie
http-https proxy

0) Préambule

Dans le cadre d'un projet de cryptographie nous avons dû réaliser un intercepteur TLS afin de mettre en pratique les notions vues en cours. Ce projet est réalisé dans un but éducatif et le code ne doit pas être détourné de son but premier ou utilisé à mauvais escient.

Dans la suite du rapport, nous détaillerons les aspects législatifs et cyber sécurité pouvant conduire à la mise en place d'un tel système notamment en entreprise.

Lorsque l'on modifie le fonctionnement d'un système de traitement automatisé de données il convient toujours d'être au courant de la législation en vigueur et des potentielles peines encourues s'il s'avérait que cette modification était malveillante.

1) Rappel de loi

> Article 323-1 — Modifié par LOI n°2015-912 du 24 juillet 2015 - art. 4

Le fait d'accéder ou de se maintenir, frauduleusement, dans tout ou partie d'un système de traitement automatisé de données est puni de deux ans d'emprisonnement et de 60 000 € d'amende.

Lorsqu'il en est résulté soit la suppression ou la modification de données contenues dans le système, soit une altération du fonctionnement de ce système, la peine est de trois ans d'emprisonnement et de 100 000 € d'amende.

Lorsque les infractions prévues aux deux premiers alinéas ont été commises à l'encontre d'un système de traitement automatisé de données à caractère personnel mis en œuvre par l'Etat, la peine est portée à cinq ans d'emprisonnement et à 150 000 € d'amende.

> Article 323-2 — Modifié par LOI n°2015-912 du 24 juillet 2015 - art. 4

Le fait d'entraver ou de fausser le fonctionnement d'un système de traitement automatisé de données est puni de cinq ans d'emprisonnement et de 150 000 € d'amende.

Lorsque cette infraction a été commise à l'encontre d'un système de traitement automatisé de données à caractère personnel mis en œuvre par l'Etat, la peine est portée à sept ans d'emprisonnement et à 300 000 € d'amende.

2) Contextualisation et législation sur l'interception TLS

D'un point de vue juridique, l'interception du trafic TLS est une mesure légitime notamment en entreprises, mais qui doit être encadrée.

Du point de vue " informatique et libertés ", ce déchiffrement est légitime du fait que l'employeur doit assurer la sécurité de son système d'information. Pour ce faire, il peut fixer les conditions et limites de l'utilisation des outils informatiques. Toutefois, le recours au déchiffrement doit être encadré et peut faire l'objet des mesures suivantes afin d'éviter toute déconvenue :

- Une information précise des salariés (sur les catégories de personnes impactées par la solution, la nature de l'analyse réalisée, les données conservées, les modalités d'investigation, les sites faisant l'objet d'une liste blanche, l'existence de dispositifs permettant une utilisation personnelle qui ne serait pas soumis à l'analyse des flux), par exemple dans la charte d'utilisation des moyens informatiques. **L'information doit aussi préciser les raisons de cette mesure** (identification de logiciels malveillants, protection du patrimoine informationnel, détection de flux sortants anormaux) ;

- **Une gestion stricte des droits d'accès des administrateurs aux courriers électroniques**, lesquels sont présumés avoir un caractère professionnel sauf s'ils sont identifiés comme étant personnels ;

- Une minimisation des traces conservées (ex : fichier malveillant, source, destination, et non identifiants et mots de passe) ;

- Une protection des données d'alertes extraites de l'analyse (ex : chiffrement, stockage en dehors de l'environnement de production et durée de conservation de 6 mois maximum).

3) Pourquoi est-ce important d'intercepter le trafic

Pour assurer la sécurité d'un réseau les entreprises doivent mettre en place des intercepteur. Avant l'application de la couche TLS, filtrer les requêtes HTTP était facile mais depuis il a fallu ruser et duper les clients.

L'interception du trafic TLS rajoute une couche de sécurité pour protéger un réseau. Lorsqu'un intercepteur TLS est installé sur une infrastructure, toutes les requêtes entrantes et sortantes sont interceptées. Cela permet - par exemple - de détecter des manœuvres suspectes, comme la connexion à des serveurs malveillants, l'entrée dans le réseau de logiciels corrompus ou de code malveillants, mais aussi d'empêcher la sortie de documents ou d'informations censées rester confidentielles. Il faudrait le coupler à un firewall pour rendre le proxy obligatoire pour tous les utilisateurs du réseau. Dans ce cas précis, le proxy devient un single point of Failure.
Il faut donc redoubler de vigilance quant à la fiabilité du proxy, car de mauvaises manipulations peuvent mener à des conséquences inquiétantes, comme l'exposition du trafic TLS en clair si le proxy est attaqué et cassé.

3.1) étude pratique

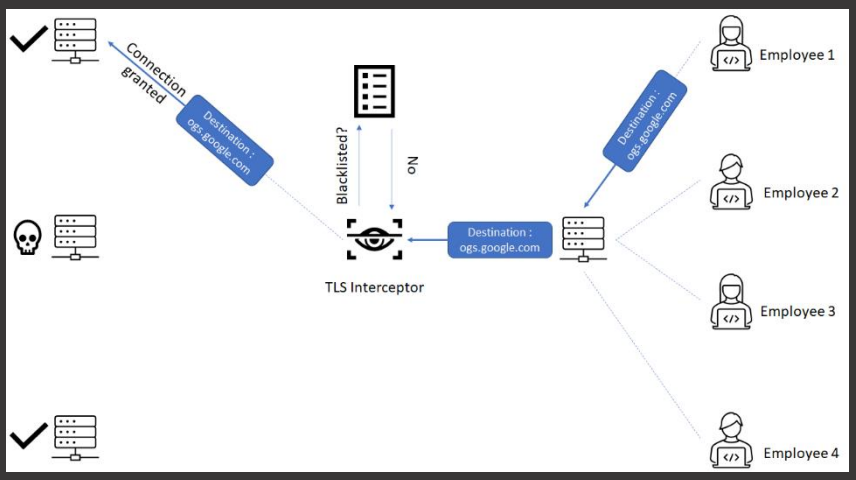

Figure 1 : Interception du TLS dans le cas d'un requête légitime

Lors de l'envoi d'une requête légitime, le trafic après analyse est simplement transféré du proxy vers le serveur cible sans aucun blocage.

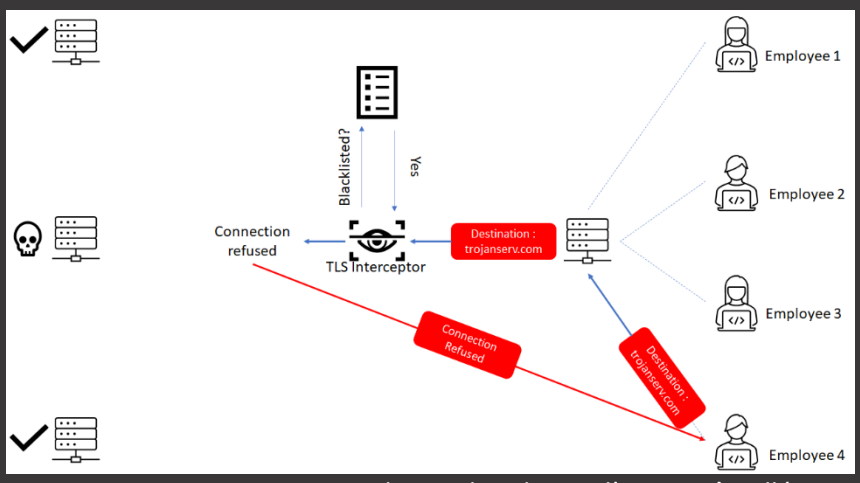

Figure 2 : Interception du TLS dans le cas d'un requête illégitime

```
Server runnig at http://localhost:8199
CONNECT push.services.mozilla.com:443 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Fire
Proxy-Connection: keep-alive
Connection: keep-alive
Host: push.services.mozilla.com:443


wrong domain
```

Figure 3 : Illustration avec notre proxy

Ce dessus on peut observer que lorsque l'adresse du serveur cible est blacklistée le proxy bloque la connexion et affiche « wrong domain » (solution implémenté dans notre projet)

Pour rendre le proxy incontournable on peut ajouter un firewall qui permettra de rediriger les flux vers celui-ci et nous assurer une visibilité sur tous le trafic.

4) Comment on intercepte le TLS

Le but de l'interception TLS est de duper le client lorsqu'il initie une connexion au serveur cible en le faisant passer par un proxy qui va simuler ce serveur cible. Pour cela, le proxy doit intégrer un serveur TLS qui fera office de "Endpoint" pour les sessions HTTPS. Ce proxy sera donc ensuite le client qui tentera de se connecter au serveur cible d'origine. Lors de cette nouvelle connexion on établira un nouveau tunnel TLS pour sécuriser les échanges. Notre intercepteur TLS est alors en position "Man In The Middle". L'intérêt de ce type de manipulation est qu'entre les deux tunnels TLS établis par le proxy on dispose du contenu HTTP en clair que l'on peut ensuite modifier ou analyser.

Pour illustrer ces explications, voici un schéma d'architecture explicatif, mis en place par l'ANSSI

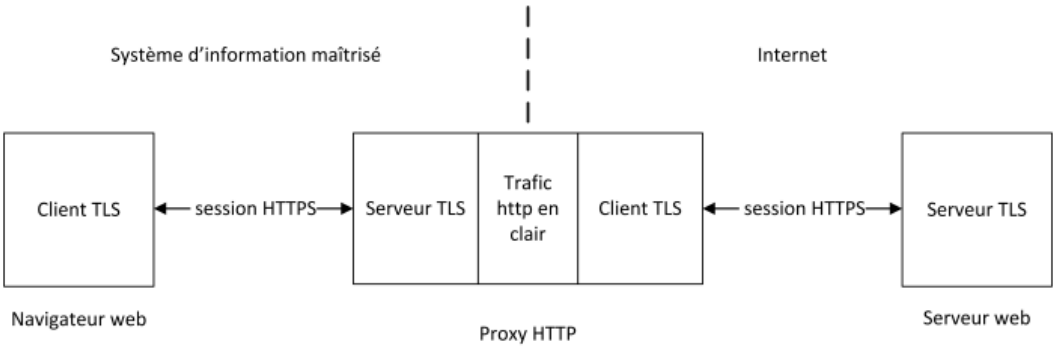

FIGURE 3 – Traitement des flux HTTPS sortants par déchiffrement

Lors de la phase d'authentification, le proxy doit pouvoir présenter un certificat valide généré pour le serveur cible.

5) Etude technique de la solution

5.1) Choix du langage

Pour développer notre solution nous avons commencés par travailler avec du python en utilisant quelques bibliothèques classique de crypto et d'HTTPS. Malheureusement la documentation était particulièrement peu fournie ou adaptée seulement en python 2. À la suite de ces problèmes nous avons donc choisis de redévelopper le proxy en nodeJS qui est un langage tout à fait adapté pour ce genre de programme. nodeJS est un langage très performant en backend qui embarque de nombreuses bibliothèques très bien optimisées pour travailler avec le système ou les modules cryptographiques. Nous avons par ailleurs trouvé beaucoup plus de documentation et d'explications techniques qu'en python ce qui nous a tout de même permis de partiellement rattraper notre retard.

5.2) Description de notre solution

Notre proxy embarque presque toutes les fonctionnalités d'un proxy http – https. Il nous permet de suivre en temps réel le flux de données et les requête TLS envoyées par chaque site consulté. De ce fait nous pouvons aussi agir dessus et faire du filtrage, du forwarding ou du blocage. Le programme que nous avons développé est à la fois un proxy http et https. A l'aide d'une bibliothèque de réseau nous pouvons détecter le type de connexion et modifier le port d'écoute ainsi que le traitement des requêtes.

```
CONNECT www.youtube.com:443 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
Proxy-Connection: keep-alive
Connection: keep-alive
Host: www.youtube.com:443


CONNECT i.ytimg.com:443 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
Proxy-Connection: keep-alive
Connection: keep-alive
Host: i.ytimg.com:443


CONNECT googleads.g.doubleclick.net:443 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
Proxy-Connection: keep-alive
Connection: keep-alive
Host: googleads.g.doubleclick.net:443
```

Figure 4 : Illustration des données récupérées par le proxy

Afin de connecter notre client TLS au serveur cible après l'interception nous utilisons la méthode CONNECT qui nous permet de rediriger l'adresse du serveur vers la bonne sorte. Nous avons aussi mis en place un système d'authentification par certificat avec un root-ca et un serveur-ca que nous avons aussi installés dans le trust store de notre browser. Un point d'amélioration pour le programme serait d'améliorer cette partie pour être ensuite capable de déchiffrer le TLS et de pouvoir accéder aux données chiffrées. Dans notre solutions nous nous sommes plutôt concentrés sur le filtrage et la modification des requêtes.

```
Proxying HTTPS request for : www.wikipedia.org
on port 443
readable content ? false
Proxy HTTP request for: Url {
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'r3.o.lencr.org',
  port: null,
  hostname: 'r3.o.lencr.org',
  hash: null,
  search: null,
  query: null,
  pathname: '/',
  path: '/',
  href: 'http://r3.o.lencr.org/'
}
```

Voici un exemple des données que l'on récupère avec notre proxy. Nous pouvons voir que même si nous parvenons à détecter le site et tout le trafic, le contenu reste chiffré et illisible.

Nous avons aussi mis en place un système de log par fichiers afin de garder une trace des connexions. Mais cela était plus à but éducatif.

6) Les difficultés du projet

La première difficulté rencontrée a été le choix du langage et l'étude de la documentation. Nous avons beaucoup travaillés à comprendre comment fonctionnait un proxy. Après avoir étudiés les textes de l'ANSSI nous avons ensuite été confrontés à la pratique qui, en python d'abord et en nodeJS ensuite, nous a mis face à de nombreux bug d'authentification et de confiance. Un point reste toujours à améliorer, authentifier notre serveur avec le browser. Malgré avoir installés les certificats les connexions ne sont pas sécurisées et nous n'avons donc pas accès au TLS déchiffré.

7) Les apports du projet

Ce projet nous a permis de mettre en pratique de nombreuses compétences de cyber sécurité et de cryptographie dans un exercice à la fois pratique et utile. Nous avons pu comprendre l'architecture d'un proxy, nous avons pu développer plusieurs types de proxy pour comprendre comment cela fonctionnait (forward proxy, middleware, intercepteur) tout en gardant en tête l'aspect législatif de notre travail qui entre de mauvaises mains peut devenir un outil malveillant. Les expérimentations que nous avons faites nous ont aussi permis de comprendre l'organisation des autorités de certifications et la place que prenaient les certificats dans la sécurisation des échanges TLS.

Nous avons aussi pu consolider nos compétences en nodeJS et développer ensemble des outils fonctionnels et intéressants.

8) Déploiement de notre solution avec nodeJS

Installation de nodeJS :

Pour Windows/MacOs

- Télécharger NodeJS en allant sur la page officielle de NodeJS : https://nodejs.org/fr/download/

- Télécharger et lancer l'exécutable.

Sur Linux (Ubuntu) :

- Installer le dépôt Node Source correspondant à la version de NodeJs que vous souhaitez utiliser. Pour la version 12.x de NodeJS, taper les commandes suivantes :
    - `curl -sL https://deb.nodesource.com/setup_12.x | sudo -E Bash –`
    - `sudo apt-get install nodejs`

- Vous pouvez vérifier la bonne installation de NodeJs en tapant cette commande :
    - `node –-version`

9) Exécuter le programme

Notre intercepteur TLS écoute sur le port 8199 en localhost. Nous vous demandons donc de changer vos paramètres de proxy navigateur, afin que l'intercepteur TLS puisse fonctionner.
(écoute à l'adresse 127.0.0.1, port 8199)

Afin de faire fonctionner notre intercepteur TLS, **le certificat nommé "ca-cert.pem" devra être placé dans le Trust Store de votre navigateur**. Sans ce dernier, tout certificat généré dynamiquement sera reconnu par votre navigateur comme étant un certificat autosigné et la connexion au serveur cible n'aura pas lieu.

Ouvrir un terminal dans le dossier où se trouve notre intercepteur TLS, et simplement taper la commande `npm start`

Une fois cette commande lancée, votre terminal vous affichera ceci :

```
[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Server runnig at http://localhost:8199
```

Si vous avez déjà des pages ouvertes sur le navigateur ou vous avez configuré votre proxy, elles seront affichées (voir exemple ci-dessous)

```
Pragma: no-cache
Cache-Control: no-cache

0R0P0N0L0J0     ♦♦+♫♥☻→♦♦╛╒╒▲╒╒◘On◘[╒╒╒╒]♦╛Jo↔5I►l2◘╒╒╒╒╒◘◘z°☻◄╒╒╒╒╒╒╒]◘◘◘^╛
POST http://geant.ocsp.sectigo.com/ HTTP/1.1
Host: geant.ocsp.sectigo.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
Accept: */*
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/ocsp-request
Content-Length: 84
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

0R0P0N0L0J0     ♦♦+♫♥☻→♦♦╛╒╒▲╒╒◘On◘[╒╒╒╒]♦╛Jo↔5I►l2◘╒╒╒╒╒◘◘z°☻◄╒╒╒╒╒╒╒]◘◘◘^╛
POST http://geant.ocsp.sectigo.com/ HTTP/1.1
Host: geant.ocsp.sectigo.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
Accept: */*
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/ocsp-request
Content-Length: 84
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
```

10) Mise en place des certificats

Création d'une autorité de certification :

- *openssl req -new -x509 -days 9999 -config ca.cnf -keyout ca-key.pem -out ca-crt.pem*

avec ca.cnf un fichier de configuration qui nous permet de faciliter la création du certificat.

```
[ ca ]
default_ca      = CA_default

[ CA_default ]
serial = ca-serial
crl = ca-crl.pem
database = ca-database.txt
name_opt = CA_default
cert_opt = CA_default
default_crl_days = 9999
default_md = md5

[ req ]
default_bits           = 4096
days                   = 9999
distinguished_name     = req_distinguished_name
attributes             = req_attributes
prompt                 = no
output_password        = password

[ req_distinguished_name ]
C                      = US
ST                     = MA
L                      = Boston
O                      = Example Co
OU                     = techops
CN                     = ca
emailAddress           = certs@example.com

[ req_attributes ]
challengePassword      = test
```

Notre fichier de configuration pour le root CA

On génère ensuite une clé privée :

- *openssl genrsa -out server-key.pem 4096*

On génère ensuite une requête pour signer notre certificat :

- *openssl req -new -config server.cnf -key server-key.pem -out server-csr.pem*

Puis on signe notre certificat :

- openssl x509 -req -extfile server.cnf -days 999 -passin "pass:password" -in server-csr.pem -CA ca-crt.pem -CAkey ca-key.pem -CAcreateserial -out server-crt.pem

Nous venons de créer un certificat autosigné pour authentifier notre serveur.

11) Annexe

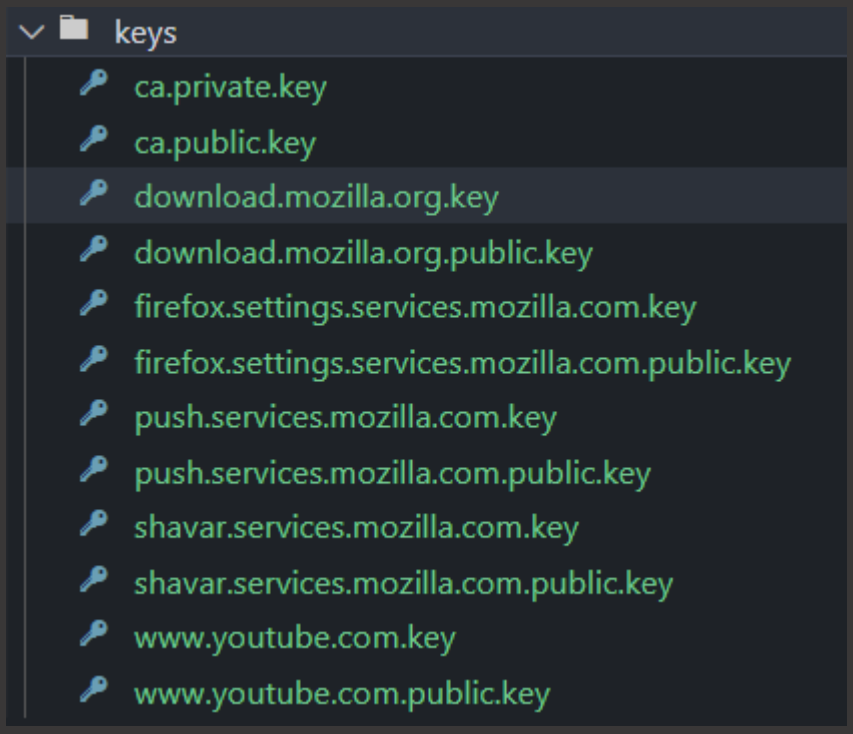

Figure 5 : Cache des clés des sites déjà visités

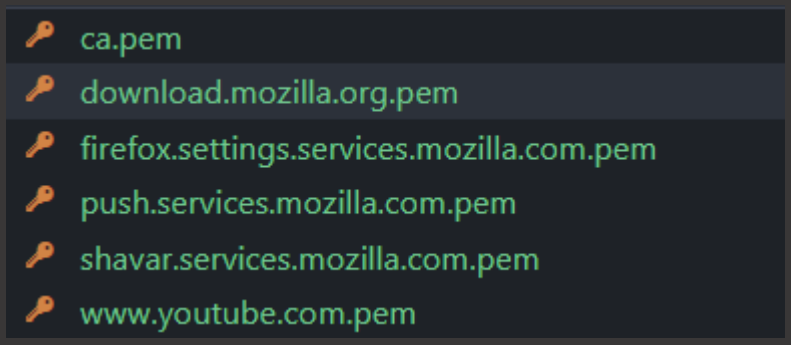

Figure 6 : Cache des certificats des sites déjà visités