Chiffrer la connexion entre le navigateur et le serveur web

Note : Pour pouvoir faire ce TP, il faut avoir fait les TP "héberger une page web", "enregistrer un nom de domaine pour votre conteneur" et "servir plusieurs sites web avec le même serveur". La configuration de virtualhosts dédiés pour chacun de vos domaines est nécéssaire.

L'objectif de ce TP est de permettre le chiffrement de la connexion entre le serveur web installé sur votre conteneur et les clients qui vont s'y connecter.

Besoin de chiffrer

Le but de cette introduction est de comprendre la nécéssité de chiffrer.

1. Installez le paquet `ngrep`, regardez rapidement la page du manuel de la commande `ngrep`.
2. Dans un terminal, exécutez la commande suivante qui va afficher le contenu de tous les segments qui ont `80` comme port source ou destination.

   ```
   # ngrep -W byline port 80
   ```
3. Depuis un navigateur web, consultez la page web http://formtest.netlib.re/ et renseignez un login et un mot de passe de votre choix.
4. Que constatez vous ?
5. Refaites la même opération, mais en observant le port `443` et en remplaçant `http` par `https` dans l'URL de la page web.
6. Que constatez vous ?

Notions de chiffrement, signature, certification

Comment se passe le chiffrement entre un client web (comme `firefox` ou `wget`) et un serveur web (comme `nginx` ou `apache2`) ?

Plus généralement, comment un client (C) et un serveur (S) communiquent de façon chiffrée ?

Supposons d'abord que S possède une paire de clefs asymétrique publique/privée et que C possède la clef publique de S.

Chiffrement de la communication

Le *chiffrement asymétrique* (avec une paire de clefs publique/privée) est très coûteux en ressources.

Ainsi, le chiffrement asymétrique est réservé pour le chiffrement de messages courts (emails), pour la signature de fichiers (en fait on ne signe que le hash du fichier pour cette même raison), pour l'authentification.

Le chiffrement de la communication entre C et le S se fait à l'aide de *chiffrement symmétrique* : une *clef de session* temporaire est établie entre C et S, c'est cette clef commune qui est utilisée pour le chiffrement et le déchiffrement des messages, dans les deux sens.

Une façon naîve de partager une clef de session commune (valable jusqu'à TLS 1.2) est la suivante : C génère la clef de session aléatoirement, chiffre cette clef de session avec la clef publique de S et envoie la clef de session chiffrée à S. S la déchiffre et la communication chiffrée peut commencer (personne d'autre n'a eu accès à la clef de session).

Cette méthode fonctionne mais ne fournit pas de confidentialité persistante (en: *forward secrecy*) : si un attaquant enregistre la communication chiffrée et récupère la clef privée de S plus tard (par exemple en force brute parce que la puissance de calcul a augmenté), il peut retrouver le contenu de la communication (il lui suffit de commencer par déchiffrer la clef de session).

À partir de TLS 1.3 (et pour SSH depuis longtemps), le pargage d'une clef de session utilise la méthode de Diffie-Hellman qui assure la confidentialité persistante (en particulier, la clef de session est construite par S et C mais elle ne circule pas, même chiffrée, entre S et C).

Comment transmettre une clef publique ?

Reste le problème suivant : comment le serveur S peut-il transmettre sa clef publique au client C ?

Le serveur ne peut pas simplement envoyer sa clef publique au serveur car un attaquant qui se trouverait sur la route pourrait la changer et se faire passer pour le serveur auprès du client.

Un mécanisme permettant cette transmission s'appelle une infrastructure à clef publique (en: public-key infrastructure, PKI).

Lorsque vous avez invité vos collègues sur votre conteneur, vous avez vérifié ensemble que le fingerprint qu'affichait le client SSH de vos invité·es correspondait bien au fingerprint d'une des clefs publiques de votre serveur `SSH` et vous avez fait cette verification en présence. Ainsi, vos collègues avaient une entière confiance au fait que la clef publique fournie par votre serveur `SSH` était la bonne (et pas une fausse clef crée par un·e attanquant·e se trouvant sur la route entre votre conteneur et le portable de vos collègues).

Au début des TP, lorsque l'enseignant·e vous a envoyé par mail le fingerprint de la clef publique du serveur SSH, vous avez aussi pu faire la vérification à distance. Pour limiter la possibilité d'un attaquant qui pourrait changer ce fingerprint puis se faire passer pour le serveur `SSH` de votre conteneur, on a fait les échanges en utilisant le mail de l'université, qu'on a considéré comme un canal sécurisé.

Le problème pour un serveur web est qu'on ne connait pas à l'avance les personnes qui vont s'y connecter. Ainsi, il n'est pas pensable de vérifier en présence la clef publique d'un serveur web avant de pouvoir s'y connecter.

Le protocole `HTTPS` n'est autre que `HTTP` over `TLS` (anciennement `SSL`). Autrement dit, les serveurs et clients web, délèguent la gestion du chiffrement à `TLS`.

La solution proposée par `TLS` est l'utilisation d'une autorité de certification : le client web possède une liste de clefs publiques d'organismes habilités à signer les clefs publiques des serveurs web. Le serveur web du nom de domaine `example.netlib.re` fait signer sa clef publique à une ces autorités en prouvant qu'il détient bien le nom de domaine `example.netlib.re` (soit en répondant à des challenges, soit en envoyant une preuve de paiement du nom de domaine, etc). Lorsque le client web reçoit la clef publique du serveur web, il reçoit aussi cette signature (appelée certificat) et peut vérifier que la clef publique du serveur web a bien été signée par la clef privée de l'autorité de certification.

Remarque : souvent les clefs publiques des organismes de certifications sont possédées par le système d'exploitation et partagées aux clients web (`wget`, `firefox`, `chromium`, etc). Sous Debian, le répertoire `/etc/ssl/certs/` contient des liens symboliques vers ces clefs.

Let's Encrypt

Jusqu'à récemment, l'obtention d'un certificat auprès d'un organisme de certification était compliquée et chère.

Nous allons utiliser l'autorité de certification Let's Encrypt, qui est reconnue par la plupart des systèmes d'exploitation et/ou navigateurs.

Cette autorité de certification a été fondée par l' Electronic Frontier Foundation, une ONG de défense des libertés sur internet, la fondation Mozilla (qui développe entre autres le navigateur firefox) et l'université du Michigan afin de généraliser l'utilisation du chiffrement sur internet (en particulier l'utilisation de `HTTPS`), en simplifiant radicalement l'étape de certification (via des challenges automatisés entre `Let's Encrypt` et le serveur web) et en la rendant gratuite.

Dans la pratique

Le logiciel `certbot` permet de créer une paire de clefs et de la faire certifier par `Let's Encrypt` (en répondant aux challenges) et même de modifier la configuration de vos virtualhosts.

7. Cherchez le paquet `certbot` qui corresponde à votre serveur web (pensez à combiner les commandes `apt search` et `grep`).
8. Consultez la description du paquet à l'aide d'une sous-commande d'`apt` (`man apt`).
9. Installez le paquet.
10. Sauvegardez le fichier de configuration du virtualhost correspondant au nom de domaine que vous voulez certifier (par exemple copiez `/etc/nginx/sites-available/<vhost>` vers `/etc/nginx/sites-available/<vhost>.bak`).
11. Créez une paire de clefs pour un site installé sur votre serveur (et répondant à un nom de domaine particulier) et faites la certifier par `Let's Encrypt`, le tout en un seul appel à la commande `certbot` (consultez les options `-d` (ou `--domain`) et `--nginx`).
12. Observez comment le fichier de configuration du virtualhost correspondant à votre nom de domaine a changé (vous pouvez par exemple utiliser la commande `diff` ou `vimdiff` pour voir les différences côte à côte).
13. Observez le répertoire `/etc/cron.d/`, quelle est sa fonction ? Observez le fichier `certbot` s'y trouve.
14. Allez sur https://dev.ssllabs.com/ssltest et testez la qualité des configurations mises en place par `certbot`.

    Remarque : comme expliqué sur cette page, le site https://www.ssllabs.com/ssltest a actuellement un problème avec les machines IPV6-only.
15. Faites une copie d'écran avec votre domaine et votre note (A), et ajoutez la sur votre page personnelle du wiki.
16. Lorsque vous avez obtenu un certificat TLS et que les clients de votre serveur web peuvent s'y connecter en HTTPS, vous pouvez ajouter `tls` à vos tags.
17. Identifiez l'item qui ne va pas (en orange), et consultez la page de blog liée.

Les failles du protocole `X.509`

Le protocole X.509 de certification par autorité de certification a le problème majeur suivant : toutes les autorités de certification reconnues par le client web peuvent signer des clefs publiques concernant votre nom de domaine. Il est arrivé, à plusieurs reprises que des autorités de certifications (commerciales ou étatiques) signent de mauvaises clefs (pour des raisons financières ou de surveillance par exemple).

En général, ces fraudes finissent par se voir et l'autorité frauduleuse finit par être exclue des listes installées dans les navigateurs.

Ainsi, ça n'est pas parce que vous avez utilisé une autorité de certification fiable pour certifier votre domaine que les communications entre votre serveur web et ses clients le sont.

Il suffit d'une mauvaise autorité de certification dans la liste du client pour qu'une attaque MITM soit possible.

Il suffit aussi qu'un attaquant arrive à se faire passer par vous auprès d'une autorité de certification de bonne foi pour qu'une attaque MITM soit possible.

Le DNS CAA sert à empêcher cette dernière attaque (mais pas la précédente).

La RFC 6844 introduit l'enregistrement `CAA` dans le DNS qui permet à l'entité qui possède un nom de domaine de spécifier quels autorités de certifications peuvent certifier ce domaine. Ainsi, si un attaquant réussit à se faire passer pour le propriétaire d'un nom de domaine, il ne pourra pas se faire certifier par n'importe quelle autorité, car celle-ci refusera d'émettre un certificat si elle voit que ce champ existe et ne lui correspond pas.

17. Faites en sorte que seule l'autorité `Let's Encrypt` accepte de certifier votre nom de domaine.
18. Vérifiez auprès de ssllabs que votre modification fonctionne bien.
19. Faites une copie d'écran avec votre domaine et le `CAA` en vert, et ajoutez la sur votre page personnelle du wiki.
20. Lorsque vous avez mis en place un enregistrement `CAA` et que seule l'autorité `Let's Encrypt` accepte de certifier votre nom de domaine, vous pouvez ajouter `caa` à vos tags.