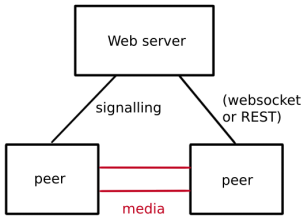
Protocoles des Services Internet VII
BROUILLON

Juliusz Chroboczek

20 novembre 2021

Dans les cours précédents, nous avons convergé sur une structure hybride, consistant d'un protocole client-serveur basé sur TCP (REST ou WebSocket) et d'un protocole pair-à-pair. Le protocole client-serveur sert à amorcer la communication, en servant notamment de point de rendez-vous pour les pairs (qui découvrent ainsi leurs adresses IP) et de point de synchronisation pour la traversée de NAT. Le canal de données, par contre, sert au transfert de données à basse latence ou volumineuse, car transiter par le serveur augmente la latence et met de la charge sur le hôte où tourne le serveur.



1 Sécurité du canal de contrôle

Le canal de contrôle est protégé par TLS. Comme nous l'avons vu précédemment, TLS fournit les fonctionnalités suivantes :

- l'authentification du serveur auprès du client à l'aide d'un *certificat* (une clé publique signée par une autorité de certification) ;
- la négociation de clés de session (dans les versions récentes de TLS, les anciennes versions transfèrent simplement des clés statiques) ;
- la communication des données chiffrée et authentifiée par des clés de session.

Il reste donc à authentifier le client auprès du serveur. On utilise pour cela soit des paires (nom d'utilisateur, mot de passe), ce qui est vulnérable aux attaques par dictionnaire, soit des *tokens* avec état ou des *tokens* cryptographiques (voir TP 5).

2 Attaques sur le canal de données

Si le canal de contrôle est sécurisé, il n'en est rien du canal de données. Les attaques suivantes sont possibles sur UDP :

1. écoute des données, par exemple si l'attaquant lance *tcpdump* sur un routeur qu'il contrôle ;

2. modification des données, par exemple si l'attaquant modifie les données sur un routeur qu'il contrôle ;
3. suppression des données ;
4. insertion des données ;
5. insertion « aveugle », ou un attaquant qui ne se trouve pas sur le chemin es données injecte un paquet sans voir les autres.

Si les points (1) et (2) sont applicables au traffic TCP non sécurisé, les points (3) à (5) sont plus faciles avec UDP. L'insertion et la suppression des données sont évitées par le mécanisme de séquençage de TCP. Quant à l'insertion aveugle, elle requiert de deviner un numéro de séquence, ce qui est possible (l'espace des numéros de séquence est limité) mais pas forcément facile.

Les point (3) et (4) peuvent être évités soit en construisant des protocoles qui résistent à la suppression des données soit en numérotant tous les paquets. Le point (5) peut être évité en utilisant un identificateur de session ou un identificateur de requête suffisamment grand pour qu'il ne puisse pas être deviné par l'attaquant. Quant aux points (1) et (2), ils recquièrent des techniques cryptographiques.

3 Négociation de clés authentifiée

Pour pouvoir authentifier et chiffrer les données, les deux pairs doivent se mettre d'accord sur un ou deux secrets, les « clés », que personne d'autre ne connaît. Ces clés dites « de session » ont une durée de vie limitée — elles seront supprimées de la mémoire des pairs dès que la communication sera terminée (à la fin de la « session »), ce qui garantit la propriété de *perfect forward secrecy*.

3.1 Clé transmise à travers le serveur

La solution la plus simple au problème de la négociation de clés consiste à les transmettre à travers le canal de contrôle, que nous supposons sécurisé (paragraphe 1). Le pair *A* choisit une clé à l'aide d'un générateur de nombres aléatoires cryptographique, et la transfère au serveur ; le serveur retransmet la clé au pair *B*.

Si cette solution a été utilisée dans beaucoup de protocoles (on peut citer SIP), elle n'est plus à la mode car elle expose toutes les clés au serveur ; or, si l'on peut généralement faire confiance à celui-ci pour ne pas monter d'attaques actives sur le traffic, l'expérience montre qu'on ne peut pas lui faire confiance pour ne pas fuiter des clés, par exemple en les stockant dans des *logs*. Les protocoles récents traitent le serveur comme un canal qui garantit l'authenticité et l'intégrité mais pas la confidentialité des messages échangés, et s'en servent donc pour un protocole de négociation de clés.

3.2 Rappel : négociation de clés

Rappel Diffie-Hellman.
Rappel attaque MITM.

3.3 Clé négociée à travers le serveur

4 *Encrypt-then-authenticate*

Il existe (au moins) deux manières de coder les messages authentifiés et chiffrés : on peut soit chiffrer le message puis authentifier le message chiffré *(encrypt-then-authenticate)* soit authentifier le message clair puis chiffrer la concaténation du message avec sa signature *(authenticate-then-encrypt)*.

D'un point de vue théorique, l'approche *authenticate-then-encrypt* semble préférable : elle authentifie le message en clair, qui est après tout ce que voit l'application. D'un point de vue pratique, cependant, seule l'approche *encrypt-then-authenticate* peut être implémentée de façon raisonnablement sûre.

Pour nous en convaincre, comparons la tâche du récepteur dans les deux approches. Dans l'approche *encrypt-then-authenticate*, le récepteur reçoit un paquet, extrait le MAC, et vérifie l'authenticité et l'intégrité ; si le MAC était incorrect, il jette le paquet, sans même le *parser*. Dans l'approche *authenticate-then-encrypt*, le récepteur doit d'abord déchiffrer le paquet, avant même de savoir s'il était authentique. Un attaquant peut donc forcer le récepteur à dépenser du temps CPU en déchiffrage, il peut mesurer le temps nécessaire pour déchiffrer les paquets pour fuiter des morceaux de clés (il est impossible d'implémenter AES en temps constant sans support matériel), et il peut aussi faire des attaques plus subtiles telles que les « *padding oracles* ».

Certains ont même défini le *principle of cryptographic doom* : toute approche à la confidentialité qui n'emploie pas le *encrypt-then-authenticate* mène à la ruine *(doom)*.