RFC 8164 : Opportunistic Security for HTTP/2

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 16 mai 2017

Date de publication du RFC : Mai 2017

https://www.bortzmeyer.org/8164.html

Pendant la mise au point de la version 2 du protocole HTTP (finalement normalisée dans le RFC 7540 [1]), un débat très vigoureux avait porté sur la possibilité de chiffrer les échanges avec TLS **même si** le plan de l'URL demandé était `http:` (et pas `https:`). Certains demandaient le chiffrement systématique <https://arstechnica.com/security/2013/11/encrypt-all-the-worlds-web-traffic-internet-arc> (que l'URL commence par `http:` ou `https:`), d'autres voulaient garder la même sémantique que HTTP version 1 (TLS pour `https:`, en clair pour `http:`). Cette dernière décision l'avait emporté à l'époque, en gardant la possibilité de permettre une extension à HTTP/2. Ce RFC décrivait justement une telle extension : en HTTP/2, on pouvait utiliser TLS (et donc HTTPS) même pour un URL de plan `http:`. Mais l'expérience a été abandonnée en décembre 2021 (personne n'utilisait cette extension) et le RFC reclassé comme d'intérêt historique seulement.

Le problème à résoudre est celui de la surveillance de masse, à laquelle procèdent un certain nombre d'acteurs (les États, bien sûr, mais pas uniquement, certains FAI, certains réseaux locaux, surveillent le trafic de leurs utilisateurs). Cette surveillance de masse est considérée, à juste titre, par l'IETF comme un problème de sécurité, et contre lequel il faut donc trouver des solutions ou au moins des mitigations (RFC 7258). Chiffrer massivement le trafic Web est évidemment indispensable pour diminuer l'efficacité de la surveillance.

Mais le modèle de HTTP version 1 rend cela difficile. En HTTP/1, on accède à un URL de plan `http:` avec du trafic en clair, passer à TLS nécessite de changer les URL, donc les pages Web qui les contiennent, les signets des utilisateurs, etc. Des logiciels comme HTTPS Everywhere aident à cela mais ne sont pas une solution parfaite (rappelez-vous par exemple qu'une bonne partie du trafic HTTP n'est pas due aux navigateurs Web).

Il serait tentant de résoudre le problème en disant « le client HTTP qui tente d'accéder à un URL de plan `http:` n'a qu'à essayer en même temps HTTPS. Si ça marche, tant mieux. Si ça rate, au moins on aura essayé. » C'est ce qu'on nomme parfois le « chiffrement opportuniste » (RFC 7435). Mais cela pose trois problèmes :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc7540.txt`

— Si on tente HTTPS d'abord, sur le port 443, et qu'un pare-feu sur le trajet absorbe ces paquets, on devra attendre l'expiration du délai de garde avant d'essayer avec succès sur le port 80. Ce problème est réel, mais soluble par l'algorithme des globes oculaires heureux, décrit dans le RFC 6555.
— Que faire si ça réussit en HTTPS mais que le certificat du serveur ne peut pas être validé ? La difficulté et/ou le coût d'un certificat sont après tout les principales raisons pour lesquelles HTTPS n'est pas davantage déployé. (Je ne publie pas des URL `https:` pour mon blog <`https://www.bortzmeyer.org/https-blog.html`> car beaucoup de gens n'ont pas mon AC dans leur magasin d'autorités.) On note qu'aujourd'hui les alertes de sécurité des navigateurs Web sont souvent absurdes : si on se connecte en HTTPS mais avec un certificat expiré (qui a donc été parfaitement valable), on a des alertes **plus effrayantes** que si on se connecte en clair !
— Enfin, et c'est le plus gros problème, rien ne garantit qu'on obtiendra le même contenu en HTTP et en HTTPS : la plupart des serveurs HTTP permettent de configurer deux "*virtual host*" différents sur les deux ports 80 et 443. Pas question donc de jouer à ça sans une autorisation explicite du serveur.

Bref, pour le HTTP traditionnel, il semble qu'il n'y ait pas de solution.

Celle proposée par notre RFC est d'utiliser le mécanisme des services alternatifs du RFC 7838. Le serveur va indiquer (typiquement par un en-tête `Alt-Svc:`) qu'il est accessible par un autre mécanisme (par exemple HTTPS). Cela a également l'avantage d'éviter les problèmes de contenu mixte <`https://www.w3.org/TR/2016/CR-mixed-content-20160802`> qui surviendraient si on avait mis la page en HTTPS mais pas tous ses contenus. Par contre, l'indication de service alternatif n'étant pas forcément bien protégée, ce mécanisme « opportuniste » reste vulnérable aux attaques actives. En revanche, ce mécanisme devrait être suffisamment simple pour être largement déployé assez vite.

Donc, maintenant, les détails concrets (section 2 du RFC). Le serveur qui accepte de servir des URL `http:` avec TLS annonce le service alternatif. Notez que les clients HTTP/1 n'y arriveront pas, car ils ne peuvent pas indiquer l'URL complet (avec son plan) dans la requête à un serveur d'origine (section 5.3.1 du RFC 7230). Cette spécification est donc limitée à HTTP/2 (RFC 7540). Si le client le veut bien, il va alors effectuer des requêtes chiffrées vers la nouvelle destination. S'il ne veut pas, ou si pour une raison ou pour une autre, la session TLS ne peut pas être établie, on se rabat sur du texte en clair (chose qu'on ne ferai jamais avec un URL `https:`).

Si le client est vraiment soucieux de son intimité et ne veut même pas que la première requête soit en clair, il peut utiliser une commande HTTP qui ne révèle pas grand'chose, comme OPTIONS (section 4.3.7 du RFC 7231).

Le certificat client ne servirait à rien dans ce chiffrement opportuniste et donc, même si on en a un, on ne doit pas l'envoyer. Par contre, le serveur doit avoir un certificat, et valide (RFC 2818) pour le service d'origine (si le service d'origine était en `foo.example` et que le service alternatif est en `bar.example`, le certificat doit indiquer au moins `foo.example`). Ce service ne permet donc pas de se chiffrer sans authentification, par exemple avec un certificat expiré, ou avec une AC inconnue du client, et ne résout donc pas un des plus gros problèmes de HTTPS. Mais c'est une exigence de la section 2.1 du RFC 7838, qui exige que le renvoi à un service alternatif soit « raisonnablement » sécurisé. (Notez que cette vérification est délicate, comme l'a montré CVE-2015-0799 <`https://bugzilla.mozilla.org/show_bug.cgi?id=1148328`>.)

En outre, le client **doit** avoir fait une requête sécurisée pour le nom bien connu (RFC 8615, pour la notion de nom bien connu) `/.well-known/http-opportunistic`. La réponse à cette requête doit être positive, et doit être en JSON, et contenir un tableau de chaînes de caractères dont l'une doit être le nom d'origine (pour être sûr que ce serveur autorise le service alternatif, car le certificat du serveur effectivement utilisé prouve une autorisation du serveur alternatif, et la signature d'une AC, ce qu'on

peut trouver insuffisant). Ce nouveau nom bien connu figure désormais dans le registre IANA <https://www.iana.org/assignments/well-known-uris/well-known-uris.xml>.

La section 4 de notre RFC rappelle quelques trucs de sécurité :

— L'en-tête `Alt-Svc:` étant envoyé sur une liaison non sécurisée, il ne faut pas s'y fier aveuglément (d'où les vérifications faites ci-dessus).
— Certaines applications tournant sur le serveur peuvent utiliser des drôles de moyens pour déterminer si une connexion était sécurisée ou pas (par exemple en regardant le port destination). Elles pourraient faire un faux diagnostic sur les connexions utilisant le service alternatif.
— Il est trivial pour un attaquant actif (un « Homme du Milieu ») de retirer cet en-tête, et donc de faire croire au client que le serveur n'a pas de services alternatifs. Bref, cette technique ne protège que contre les attaques passives. Ce point a été un des plus discutés à l'IETF (débat classique, vaut-il mieux uniquement la meilleure sécurité, ou bien accepter une sécurité « au mieux », surtout quand l'alternative est pas de sécurité du tout).
— Le client ne doit **pas** utiliser des indicateurs qui donneraient à l'utilisateur l'impression que c'est aussi sécurisé qu'avec du « vrai » HTTPS. Donc, pas de joli cadenas fermé et vert. (C'est une réponse au problème ci-dessus.)

Apparemment, Firefox est le seul client HTTP à mettre en œuvre ce nouveau service <https://arstechnica.com/security/2015/04/new-firefox-version-says-might-as-well-to-encrypting-all> (mais avec une syntaxe différente pour le JSON <https://github.com/mozilla/gecko-dev/blob/master/netwerk/protocol/http/WellKnownOpportunisticUtils.js>, pas encore celle du RFC). Notez que le serveur ne nécessite pas de code particulier, juste une configuration (envoyer l'en-tête `Alt-Svc:`, avoir le `/.well-known/http-opportunistic`...) Les serveurs de Cloudflare permettent ce choix <https://support.cloudflare.com/hc/en-us/articles/227253688-How-do-I-enable-H