RFC 6797 : HTTP Strict Transport Security (HSTS)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 20 novembre 2012

Date de publication du RFC : Novembre 2012

https://www.bortzmeyer.org/6797.html

La technique HSTS ("*HTTP Strict Transport Security*"), normalisée dans ce RFC (mais déjà largement déployée) vise à résoudre une attaque contre TLS. Si un site Web est accessible en HTTPS, et qu'un méchant arrive à convaincre un utilisateur de se connecter en HTTP ordinaire à la place, le méchant peut alors monter une attaque de l'homme du milieu. Le fait que le site soit protégé par TLS n'aidera pas dans ce cas. Pour éviter cette attaque, HSTS permet à un site de déclarer qu'il n'est accessible **qu'en** HTTPS. Si l'utilisateur a visité le site ne serait-ce qu'une fois, son navigateur se souviendra de cette déclaration et ne fera plus ensuite de connexions non sécurisées.

Bien sûr, il n'y aurait pas de problème si l'utilisateur utilisait systématiquement des URL `https://...`. Mais les erreurs ou les oublis peuvent arriver, l'utilisateur peut cliquer sur un lien `http://...` dans un spam, il peut être victime d'une page Web certes sécurisée mais qui contient des images chargées via un URL `http://...`. Bref, il y a encore trop de risques. D'où l'idée d'**épingler** la politique de sécurité du site Web dans le navigateur, en permettant à un site de déclarer au navigateur « je suis sécurisé, ne reviens **jamais** me voir en HTTP ordinaire » et de compter sur le fait que le navigateur respectera automatiquement cette politique, et s'en souviendra. Cette déclaration se fera par l'en-tête HTTP `Strict-Transport-Security:`, par exemple `Strict-Transport-Security: max-age=7905600`.

TLS est normalisé dans le RFC 5246[1]. (Son prédécesseur, SSL, est dans le RFC 6101.) L'utilisation de HTTP au dessus de TLS (HTTPS) est décrite dans le RFC 2818. Un navigateur Web va typiquement appliquer des politiques de sécurité différentes selon que la connexion se fait en HTTP ou en HTTPS. Un exemple typique est fourni par les "*cookies*" du RFC 6265 : s'ils ont été envoyés par le serveur avec l'attribut `Secure`, ils ne seront retransmis par le navigateur que si la connexion est en HTTPS.

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc5246.txt`

Pour vérifier l'identité du serveur en face, les navigateurs valident les certificats présentés par le serveur. Si cette vérification échoue (c'est en général suite à une erreur de l'administrateur du serveur, mais cela est parfois dû à une attaque d'un intermédiaire), le navigateur affiche typiquement un avertissement à l'utilisateur, en lui permettant de continuer de manière non sûre s'il le souhaite (cette possibilité est « officialisée » par la section 3.1 du RFC 2818). Ce mécanisme de "*click-through security*", que le navigateur rend plus ou moins facile, est très critiqué : comme l'utilisateur veut avant tout continuer sa tâche, il va toujours chercher à passer outre l'avertissement de sécurité. C'est même une attitude raisonnable, compte tenu du fait que la plupart des erreurs d'authentification avec TLS ne sont **pas** des attaques, et aussi compte tenu de la difficulté, même pour un expert, d'analyser ces problèmes et de comprendre ce qui s'est passé. Deux bons articles sur les questions d'interface utilisateur liées à ce problème de sécurité sont « "*Crying Wolf : An Empirical Study of SSL Warning Effectiveness*" <http://www.usenix.org/events/sec09/tech/full_papers/sunshine.pdf> » de Sunshine, J., Egelman, S., Almuhimedi, H., Atri, N., et L. Cranor, puis « "*Stopping Spyware at the Gate : A User Study of Privacy, Notice and Spyware*" <http://www.law.berkeley.edu/files/Spyware_at_the_Gate.pdf> » de Good, N., Dhamija, R., Grossklags, J., Thaw, D., Aronowitz, S., Mulligan, D., et J. Konstan. Mais d'autres articles, non cités par ce RFC, prennent un point de vue différent, expliquant qu'ignorer les avertissements de sécurité peut être une attitude rationnelle <https://www.bortzmeyer.org/rational-security.html>.

Pour éviter ces risques, l'article de C. Jackson et A. Barth (deux des auteurs du RFC), « "*ForceHTTPS : Protecting High-Security Web Sites from Network Attacks*" <https://crypto.stanford.edu/forcehttps/> » proposait une approche où le site Web disait à ses clients qu'ils devaient utiliser HTTPS pour se connecter. Ce RFC utilise une technique différente de celle de l'article mais l'idée est la même.

La section 2 décrit en détail le **problème** de sécurité. Si vous êtes plutôt intéressé par la **solution**, il faut sauter à la section 5. Sinon, si on continue dans l'ordre, une analyse des menaces. HSTS gère trois menaces :

— Les attaques passives. Ce sont celles où l'attaquant ne fait qu'écouter. Par exemple, sur un réseau WiFi typique, écouter est possible, que le réseau soit sécurisé ou pas (« "*Practical Attacks Against WEP and WPA*" <http://dl.acm.org/citation.cfm?id=1514286> » de Beck, M. et E. Tews). Il existe même des outils tous faits pour cela comme Aircrack L'attaquant peut alors intercepter des mots de passe, des "*cookies*", etc. Une extension Firefox spectaculaire, Firesheep, permet d'automatiser cette tâche et de se retrouver rapidement connecté au compte Facebook de ses voisins de Starbucks. HTTPS empêche complètement ces attaques passives **s'il est utilisé**. Par exemple, si un site est accessible normalement à la fois en HTTP et en HTTPS, il suffit d'une seule requête HTTP, faite par accident ou oubli, et les "*cookies*" peuvent être copiés.

— Plus sophistiqué est l'attaquant actif. Il peut agir sur le DNS, ou sur le routage ou tout simplement installer un point d'accès WiFi pirate, ayant le même SSID que le vrai. Il risque plus d'être détecté mais il a ainsi des possibilités bien plus grandes. Notons que l'attaquant actif peut être un État (l'Iran s'en est fait une spécialité) et que, dans ce cas, l'attaque est bien plus facile pour lui, car il contrôle toutes les liaisons du pays. Cette fois, contre ces attaquants actifs, juste faire tourner HTTP sur TLS ne suffit plus : il faut aussi authentifier le serveur distant. Les certificats auto-signés (ou signés par une AC qui n'est pas connue du navigateur) sont dangereux car ils ne protègent pas du tout contre les attaquants actifs. Si l'on accepte les sites présentant un certificat auto-signé (ce que permettent tous les navigateurs, après un sermon sur la sécurité et l'obligation de cliquer sur un lien « je sais ce que je fais »), c'est comme si on n'utilisait pas TLS du tout puisqu'on n'est pas sûr de parler au bon site. (Une protection minimale peut être fournie par des outils comme Certificate Patrol <https://addons.mozilla.org/en-us/firefox/addon/certificate-patrol/>.)

— Mais il n'y a pas que les attaquants actifs et passifs qui peuvent compromettre la sécurité de HTTP. Il y a aussi les bogues dans les sites Web. Par exemple, si une page a un contenu en Flash, et que ce contenu est chargé via une connexion non sécurisée, l'attaquant passif pourra lire les "*cookies*" et l'attaquant actif pourra remplacer le Flash par un autre code, qui fera des tas de choses vilaines chez le client. (Dans une certaine mesure, c'est également faisable avec JavaScript ou même une CSS.) Une protection contre cela est le refus, par le navigateur, de charger du « contenu

de contextes de sécurité différents » (page HTML chargée en HTTPS et Flash chargé en HTTP) mais tous ne le font pas (cf. section 12.4), et beaucoup se contentent d'un avertissement qui peut facilement passer inaperçu. (À noter que ce contenu de contextes de sécurité différents est souvent appelé « contenu mixte » mais notre RFC déconseille ce terme, qui a un sens bien précis - et différent - en XML.)

Une analyse de sécurité sérieuse doit indiquer quelles sont les menaces traitées mais aussi celles qu'on ne prétend pas traiter. La section 2.3.2 liste les « non-buts » de HSTS :

— Le hameçonnage, où un utilisateur va donner à un site Web, authentifié et utilisant HTTPS, mais pas digne de confiance, des informations secrètes. HSTS vous protège si vous essayez d'aller sur `https://www.mybank.example/` et qu'un attaquant tente de vous détourner. Mais il ne peut rien faire si, suite à un spam, vous visitez `https://s6712.example.net/php/mybank.example`...
— Les failles de sécurité dans le navigateur : HSTS ne protège que si le navigateur est fiable.
— Le client HTTP remplacera tous les URL `http://...` par des `https://...` avant de se connecter.
— Toute erreur de validation sera fatale, sans rémission possible.

Bref, avant de déclarer en HSTS qu'on ne doit être contacté que de manière sécurisée, il faut être sûr de soi...

La section 5 du RFC expose la solution. Comme indiqué plus haut, un serveur HTTPS qui veut être sûr de n'être accédé qu'en HTTPS le déclare en envoyant un en-tête `Strict-Transport-Security:` (en-tête désormais dans le registre des en-têtes <`https://www.iana.org/assignments/message-headers/perm-headers.html`>). Évidemment, cet en-tête ne sera utilisé que si la session était sécurisée avec HTTPS (envoyé sur de l'HTTP ordinaire, il sera ignoré). La **politique** HSTS du site comprend une durée de vie maximale de l'information, et le choix d'inclure ou pas les sous-domaines. Pour la durée de vie, cela se fait avec une valeur `max-age` en secondes. Le navigateur pourra se souvenir, pendant ce temps, que le site ne doit être accédé qu'en HTTPS. Après, il pourra redemander (cela permet éventuellement de ne plus avoir HSTS, si l'on veut se remettre à autoriser HTTP). Pour les sous-domaines, c'est fait avec une directive `includeSubDomains`.

La section 6 va davantage dans les détails en couvrant la syntaxe exacte de cet en-tête. Par exemple, un `max-age` de zéro spécifie que le client HTTP doit tout de suite oublier les choix HSTS : cela revient à dire « j'arrête d'exiger HTTPS ».

Plusieurs sites Web aujourd'hui utilisent déjà HSTS. Par exemple, Gmail annonce une politique d'une durée d'un mois :

```
Strict-Transport-Security: max-age=2592000; includeSubDomains
```

Et l'EFF (qui n'utilise pas l'option `includeSubDomains` mais a presque la même durée de vie) :

```
Strict-Transport-Security: max-age=2628000
```

On l'a vu, la syntaxe est simple. Les sections suivantes portent sur une question plus complexe, le comportement des logiciels lorsqu'ils rencontrent cette option. D'abord, le serveur (section 7). S'il veut utiliser HSTS, il met cette option dans la réponse, si elle est protégée par TLS. Sinon (HTTP ordinaire), le serveur ne doit pas mettre l'en-tête HSTS et le RFC recommande une redirection (code HTTP 301, par exemple) vers un URL `https://`. Notons que c'est risqué : un attaquant actif peut aussi faire des redirections, si la requête était en HTTP. Voir l'excellente « "*Transport Layer Protection Cheat Sheet*" <`http://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet`> » de Coates, M., Wichers, d., Boberski, M., et T. Reguly.

Et le client HTTP (typiquement un navigateur Web) ? La section 8 lui donne des instructions : si on reçoit l'en-tête HSTS sur une session HTTP ordinaire, l'ignorer (cela peut être une blague, puisque la session n'est pas authentifiée), si on le reçoit sur HTTPS, enregistrer cette information (avec la durée de vie associée). Puis, à la prochaine visite de ce serveur (les règles exactes sont un peu plus compliquées, avec les sous-domaines, et les IDN) :

— Ne se connecter qu'en HTTPS (si le navigateur reçoit, par frappe de l'utilisateur, ou dans une page HTML, un URL `http://...`, le réécrire automatiquement en `https://...`).
— N'accepter aucune erreur d'authentification (pas de dialogue « voulez-vous quand même continuer même si le certificat X.509 est pourri ? »). Si un serveur met cet en-tête, c'est que son administrateur sait ce qu'il fait et qu'il s'engage à gérer son serveur correctement (pas de certificat expiré, par exemple, cf. l'exemple à la fin).

À noter que l'en-tête HSTS **doit** être envoyé via le protocole réseau, pas via le contenu du document : le truc HTML `<meta http-equiv="Strict-Transport-Security" ...>` est explicitement interdit.

Les sections 11 et 12 donnent des avis aux programmeurs sur la bonne façon de mettre en œuvre ce système de sécurité. D'abord, les serveurs (section 11). Le RFC leur rappelle que le serveur ne peut pas savoir si son client respecte l'en-tête qu'on lui envoie. On ne peut donc pas compter uniquement sur HSTS pour traiter les menaces décrites en section 2.

Ensuite, le choix de la durée de vie (via l'attribut `max-age`). Il y a deux approches : une durée constante ou bien un `max-age` calculé à chaque fois pour correspondre à une date d'expiration choisie. La première est évidemment plus facile à mettre en œuvre (sur Apache, un `Header set Strict-Transport-Secu "max-age=604800"` dans la configuration ou bien dans un `.htaccess` suffit). La seconde nécessite un calcul dynamique mais a l'avantage de pouvoir faire correspondre la date d'expiration de HSTS avec une autre date importante (le RFC suggère celle d'expiration du certificat X.509 ; si on ne renouvelle pas le certificat, c'est une bonne chose si HSTS cesse également).

Le RFC met aussi en garde contre les certificats auto-signés ou bien signés par une AC non connue de certains navigateurs (comme CAcert `<https://www.bortzmeyer.org/cacert.html>`). HSTS interdisant formellement de passer outre un problème X.509 (pas de formulaire « voulez-vous continuer bien que je ne connaisse pas cette AC ? »), si on active HSTS en envoyant l'en-tête `Strict-Transport-Security:`, et qu'il est accepté, on risque, si on change ensuite de certificat, d'être injoignable par tous les clients (cas où on est passé à un certificat auto-signé) ou par une partie d'entre eux (cas où est est passé à une AC peu connue). Rappelez-vous que HSTS est une déclaration « je suis un parano expert en crypto et je ne ferai **jamais** d'erreur, n'utilisez que HTTPS ». Réflechissez donc bien avant de l'activer.

Autre piège pour l'administrateur d'un serveur HTTPS, le cas du `includeSubDomains`. Si on l'utilise, on peut bloquer tout sous-domaine qui ne serait pas accessible en HTTPS. Si vous gérez le domaine d'une grande entreprise, mettons `example.com`, et que vous mettez un `Strict-Transport-Security: max-age=15768000 ; includeSubDomains`, alors le petit site Web que vous aviez oublié, `http://service.de` qui n'était accessible qu'en HTTP, deviendra inutilisable pour tous ceux qui utilisent HSTS. Soyez donc sûr de vos sous-domaines avant d'activer cette option. (Le RFC cite l'exemple d'une AC `ca.example` qui veut permettre l'accès à OCSP en HTTP ordinaire : la directive `includeSubDomains` pourrait bloquer `ocsp.ca.example`.) D'un autre côté, sans cette option, vous n'êtes plus sûr de protéger les "*cookies*" (cf. section 14.4) puisque ceux-ci peuvent être envoyés à tout serveur d'un sous-domaine. (Avec `secure`, ce sera forcément en HTTPS mais pas forcément blindé par HSTS.)

Et pour les auteurs de navigateurs Web ? La section 12 leur dispense de sages conseils. D'abord, elle leur rappelle un principe central de HSTS : **pas de recours pour les utilisateurs**. Si un site est étiqueté en HSTS et qu'il y a un problème d'authentification, il est **interdit** de permettre aux utilisateurs de passer outre. HSTS est là pour la sécurité, pas pour faciliter la vie des utilisateurs. Le but est d'éviter qu'une attaque de l'Homme du Milieu soit possible parce que l'utilisateur est trop prêt à prendre des risques.

Une suggestion intéressante pour les auteurs de navigateurs est celle de livrer le logiciel avec une liste pré-définie de sites HSTS (grandes banques, etc), comme ils ont aujourd'hui une liste pré-définie d'AC. Cela permettrait de fermer une des faiblesses de HSTS, le fait que la première connexion, si elle

n'est pas faite en HTTPS (et sans erreur), n'est pas sécurisée. Après, on peut imaginer que cette liste soit éditable (possibilité d'ajouter ou de retirer des sites) mais permettre de retirer un site est dangereux : un utilisateur inconscient risquerait de virer un site important et de se retrouver ainsi vulnérable. En tout cas, le RFC dit que cette possibilité de retirer un site ne doit **pas** être accessible à JavaScript, pour éviter qu'un logiciel malveillant dans une page Web ne s'attaque à la sécurité de HSTS.

Tout ce RFC est évidemment consacré à la sécurité mais la traditionnelle section "*Security considerations*" (section 14) est néanmoins utile pour évaluer les risques restants. Ainsi, HSTS ne protège pas contre les attaques par déni de service. Il peut même les faciliter : par exemple, un attaquant actif peut envoyer le trafic HTTPS vers un serveur qu'il contrôle et, si l'utilisateur accepte le certificat invalide, le serveur pourra alors mettre un `Strict-Transport-Security:` non prévu et qui empêchera l'accès au serveur légitime dans certains cas. Autre possibilité, si le navigateur permet à l'utilisateur ou, pire, à du code JavaScript, d'ajouter un site à la liste locale des sites HSTS, les sites pas accessibles en HTTPS deviendront inutilisables.

Autre problème que HSTS ne résoud pas complètement : la première connexion. Si, sur une nouvelle machine (dont la mémoire HSTS est vide), vous vous connectez à `http://www.example.com/`, HSTS ne vous protège pas encore. Si aucun attaquant n'était là pendant la première connexion, vous serez redirigé vers `https://www.example.com/`, l'en-tête `Strict-Transport-Security:` définira ce site comme HSTS et tout sera protégé par la suite. Mais, si un attaquant était présent au moment de cette première connexion, HSTS ne pourra rien faire. C'est un des cas où la liste pré-définie de sites HSTS, livrée avec le navigateur, est utile.

Dernier piège, plus rigolo, un attaquant peut découvrir (par une combinaison de noms de domaines qu'il contrôle et d'un script tournant sur le navigateur) si le navigateur a déjà visité tel site HSTS (car, dans ce cas, les liens vers un URL `http://...` seront automatiquement réécrits en `https://...`). Cela permet une forme de flicage du navigateur (voir la synthèse « "*Web Tracking*" <`http://www.snet.tu-berlin.de/fileadmin/fg220/courses/SS11/snet-project/web-tracking_schmuecker.pdf`> » de N. Schmucker.)

Deux annexes du RFC intéresseront les concepteurs de protocoles. L'annexe A documente les raisons des choix effectués par HSTS. Par exemple, la politique HSTS est indiquée par un en-tête et pas par un "*cookie*" car les "*cookies*" peuvent être manipulés et modifiés quand ils sont dans la mémoire du navigateur. Autre cas qui a suscité des discussions, le fait que le client HTTPS fasse confiance à la dernière information reçue via `Strict-Transport-Security:`. Par exemple, si un serveur annonce un `max-age` d'un an le 18 novembre 2012, puis d'un mois le 19 novembre 2012, l'information expirera-t-elle le 18 novembre 2013 ou bien le 19 décembre 2012 ? Avec HSTS, c'est le dernier en-tête qui gagne, donc l'expiration sera le 19 décembre 2012. C'est peut-être une faiblesse, question sécurité, mais cela permet aux gérants des sites Web de corriger leurs erreurs (comme un `max-age` excessif).

À propos du `max-age`, la même annexe explique aussi pourquoi il indique une durée et pas une date d'expiration : c'est pour éviter d'être dépendant de la synchronisation des horloges. Le RFC donne une autre raison, le fait d'être ainsi dispensé de la définition et de l'implémentation d'une syntaxe pour représenter les dates. Cela ne me semble pas convaincant : il existe déjà une syntaxe normalisée et largement implémentée, celle du RFC 3339. Mais, bon , la sécurité préfère des normes simples, X.509 est un parfait exemple <`http://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt`> des résultats auxquels mènent des normes compliquées.

Quant à l'annexe B, elle explique les différences entre HSTS et la SOP ("*Same-Origin Policy*") du RFC 6454.

Ce RFC normalise une technique déjà largement déployée. Plusieurs sites Web envoient déjà cet en-tête et plusieurs navigateurs le reconnaissent (on trouve deux bonnes listes sur le Wikipédia anglophone).

À noter qu'il y eu des projets plus ou moins avancés de mettre cette déclaration « TLS seulement » dans le DNS (évidemment avec DNSSEC) plutôt que dans HTTP. Cela aurait permis de fermer la faille de la première connexion, et de gérer le cas des autres protocoles sécurisés avec TLS. Deux exemples de cette voie (tous les deux n'ayant pas été poursuivis) : l'"*Internet-Draft*" `draft-hoffman-server-has-tls` et un article de George Ou `<http://www.circleid.com/posts/20090105_problem_with_https_ssl_md5/>`.

J'ai testé HSTS avec un Chromium « Version 20.0.1132.47 Ubuntu 12.04 (144678) » et tcpdump qui écoute. Tout marche bien. L'en-tête HSTS est ignoré si le certificat n'a pas pu être validé (AC inconnue). Une fois le certificat validé (AC ajoutée au magasin du navigateur), dès que l'en-tête HSTS a été reçu, plus aucune tentative n'est faite en HTTP, le navigateur file sur le port 443 directement.

Quant à Firefox, voici ce qu'il affiche quand un site qui avait auparavant été accessible en HTTPS authentifié ne l'est plus (ici, parce que le certificat a expiré) :

HSTS est également mis en œuvre dans wget (voir la documentation `<https://www.gnu.org/software/wget/manual/html_node/HTTPS-_0028SSL_002fTLS_0029-Options.html#index-HSTS>`) et c'est activé par défaut.

Merci à Florian Maury pour sa relecture attentive. Un autre article sur HSTS est celui de Cloudflare `<https://blog.cloudflare.com/enforce-web-policy-with-hypertext-strict-transport-securi>`.