Mon blog plus à poil sur l'Internet, grâce à TLS

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 10 septembre 2014. Dernière mise à jour le 25 novembre 2018

https://www.bortzmeyer.org/https-blog.html

Voilà, plus d'un an après les révélations de Snowden, ce blog est enfin sécurisé par HTTPS et vous pouvez donc vous connecter à <https://www.bortzmeyer.org/>.

L'intérêt de TLS (et de son utilisation pour le Web, HTTPS) pour sécuriser un site ne se discute plus. C'est conseillé à la fois par l'ANSSI (dans ses « Recommandations pour la sécurisation des sites web <http://www.ssi.gouv.fr/fr/guides-et-bonnes-pratiques/recommandations-et-guides/securite-des-applications-web/recommandations-pour-la-securisation-des-sites-web.html> » : « Le recourt [sic] à TLS, c[Caractère Unicode non montré [1]]est à dire [re-sic] l[Caractère Unicode non montré]emploi du protocole HTTPS, est recommandé dès lors que les communications entre le client et le serveur doivent être protégées en confidentialité ou en intégrité »), mais également par tous les hackers, darknetteurs, cryptoterroristes, cipherpunks, etc. Depuis qu'Edward Snowden a courageusement publié plein de détails sur l'intensité de l'espionnage de masse sur le Web, c'est encore plus crucial. Sans TLS, n'importe qui peut savoir ce que vous regardez sur ce blog, quelles recherches </search> vous faites, et un attaquant actif peut même modifier le contenu des pages avant qu'elles ne vous arrivent, me faisant dire encore plus n'importe quoi que d'habitude (voir un joli exemple avec Haka <https://www.bortzmeyer.org/haka.html>).

Lorsque HTTPS a été activé sur ce blog, j'avais fait le choix d'une autorité de certification que votre navigateur Web ne connait peut-être pas. Pour comprendre, il faut revenir au modèle de TLS (et donc de HTTPS). Client et serveur se mettent d'accord sur un mécanisme de chiffrement et sur ses paramètres, comme la clé. Une fois le chiffrement en route, un attaquant passif ne peut plus comprendre le contenu des communications (mais il peut toujours regarder qui communique, donc savoir que vous parlez à mon serveur) et un attaquant actif ne peut plus le modifier sans que ce soit détecté. Mais il reste un problème : comment le client peut-il savoir qu'il parle au vrai serveur et pas à un homme du milieu, un serveur qui se serait glissé au milieu de la conversation ? (Il existe plusieurs techniques pour être homme du milieu. Le domaine bortzmeyer.org étant signé avec DNSSEC, cela en élimine une, l'empoisonnement DNS, mais il en reste d'autres.) Ce genre d'attaques est fréquent : beaucoup d'entreprises font cela

1. Car trop difficile à faire afficher par LaTeX

pour surveiller leurs employés, par exemple, et il existe une offre commerciale pour cela (par exemple, Blue Coat se vante <https://www.bluecoat.com/products/proxysg> d'être capable de « *"visibility of SSL [sic] -encrypted traffic (including the ability to stream decrypted content to an external server with an Encrypted Tap license)"* »). C'est réalisé en détournant le trafic dans le routeur de sortie et en l'envoyant à un équipement qui déchiffre et rechiffre après, pour l'envoyer au vrai serveur après examen. Cela se fait aussi au niveau des États (en Iran, par exemple).

Comment est-ce que TLS se protège contre ce genre d'attaques ? En authentifiant le serveur : il existe plusieurs techniques pour cela mais la plus courante, de loin, est d'utiliser un certificat à la norme X.509 (souvent appelé, par très gros abus de langage, « certificat SSL »). Ce certificat est composé d'une clé publique et de diverses métadonnées, notamment un nom (en X.509, on dit un sujet) comme `www.bortzmeyer.org`, et une signature de la clé publique par une autorité de certification (AC). Le client TLS va vérifier la signature des messages (prouvant que le serveur connait bien la clé privée) et la signature du certificat (prouvant que l'AC a validé que le certificat appartenait bien au titulaire légitime du nom). Quelles sont les autorités de certification acceptées ? Eh bien, c'est justement le piège : il n'existe pas de liste officielle, chacun a la sienne, stockée dans son magasin de certificats, et un même site Web peut donc parfaitement être accepté sans histoires par un navigateur et rejeté par un autre. En pratique, la plupart des utilisateurs font une confiance aveugle aux auteurs de leur navigateur Web (Mozilla, Google, Microsoft, etc) et utilisent sans se poser de questions les AC incluses par défauts dans les navigateurs. Cela marche à peu près, tant qu'une AC ne trahit pas (par exemple, au Ministère des finances français <http://www.nextinpact.com/news/84805-google-bloque-certificats-corrompus-emis-par-autorite-liee-a-l-anssi.htm>).

Mais le problème est qu'on ne peut jamais être sûr que son certificat sera accepté partout. Pour limiter les risques, la plupart des webmestres ne prennent pas de risques et font appel à une des grosses AC connues partout. Ce n'est pas satisfaisant, ni du point de vue de la sécurité, ni du point de vue du "*business*" (il devient très difficile d'introduire une nouvelle AC, la concurrence reste donc assez théorique). J'ai choisi une AC qui est très simple à utiliser, rend un bon service, et est contrôlée par ses utilisateurs, CAcert <https://www.bortzmeyer.org/cacert.html> (je suis depuis passé à Let's Encrypt <https://www.bortzmeyer.org/passage-blog-lets-encrypt.html>). Mais CAcert est dans peu de magasins et, si vous visitez mon blog en HTTPS <https://www.bortzmeyer.org/>, il y a des chances que vous ayiez un message d'erreur du genre « "*Unknown issuer*" ». Ce n'est pas satisfaisant, je le sais mais, comme je ne fais pas de "*e-commerce*" sur ce blog et que je ne manipule pas de données personnelles, je trouve que c'est acceptable et que c'est l'occasion de faire de la publicité pour les AC « alternatives ». Je vous encourage donc à ajouter le certificat de CAcert <http://www.cacert.org/index.php?id=3> à votre magasin (j'ai mis un lien HTTP ordinaire car, si vous n'avez pas déjà le certificat CAcert, le HTTPS vous donnera une erreur : vous devez vérifier l'empreinte du certificat par un autre moyen). Au fait, pour les utilisateurs d'Android, on me souffle que sur un téléphone non rooté <https://www.bortzmeyer.org/j-ai-roote-mon-phone.html>, il faut passer par les paramètres Sécurité ¿ Installer depuis stockage. En tout cas, ça a bien marché sur mon Fairphone <https://www.bortzmeyer.org/fairphone.html> rooté.

Un problème supplémentaire est la pauvreté des messages d'erreur des navigateurs, qui ne permettent pas au visiteur de comprendre clairement ce qui s'est passé. Félicitons Safari pour son bon message d'erreur :

Bien sûr, il y a d'autres AC, plus traditionnelles et qui auraient créé moins de surprises à mes visiteurs. Certaines sont très chères et, surtout, l'obtention d'un certificat nécessite un processus compliqué (et pas forcément plus sûr). En revanche, on m'a souvent cité StartSSL comme bonne AC assez reconnue. J'ai réussi à m'y créer un compte (non sans difficultés car ils testent l'adresse de courrier, non pas par la bonne méthode - envoyer un courrier - mais en faisant une connexion SMTP directe, ce qui pose un problème avec le "*greylisting*" et génère un stupide message d'errreur « "*We were not able to verify your email address ! Please provide us with a real email address !*" »). Je testerai plus loin un autre jour. À noter

que ce débat « AC commerciales privées ou bien CAcert » agite tous les acteurs du libre, par exemple LinuxFr <https://linuxfr.org/suivi/changer-d-autorite-de-certification>, qui utilise également CAcert (pour ces raisons <https://linuxfr.org/aide#aide-autrecertificatssl>), ce qui suscite de vigoureuses discussions <https://linuxfr.org/news/firefox-32> (merci à Patrick Guignot pour les références).

En pratique, l'utilisation de CAcert a suscité trop de problèmes. Chaque fois que j'envoyais un URL avec https://, il y avait quelqu'un pour me dire « ah, il y a une erreur de configuration », souvent avec un diagnostic faux (« ton certificat est auto-signé »). Le matraquage marketing des seules AC officielles fait qu'il est très difficile de proposer autre chose. Je suis donc finalement passé à let's Encrypt <https://www.bortzmeyer.org/passage-blog-lets-encrypt.html>, AC qui, elle, est acceptée par les auteurs de logiciels.

En raison de cettte méconnaissance de CAcert par beaucoup de magasins, je n'avais donc pas imposé HTTPS (par exemple, <http://www.bortzmeyer.org/> ne redirigeait pas d'autorité vers <https://www.bortzmeyer.org/>). Pour la même raison, je n'avais pas mis d'en-tête HSTS (RFC 6797 [2]) car HSTS ne permet pas d'erreur : tout problème est fatal. Depuis le passage à let's Encrypt, ces pratiques de sécurité sont en place.

Pour les liens dans le flux de syndication, pour essayer de faire en sorte qu'ils soient cohérents avec le protocole utilisé pour récupérer ce flux, j'avais essayé de les faire commencer par // ce qui, normalement (ce n'est pas très clair dans la norme des URI, le RFC 3986), fait du HTTP si le flux a été récupéré en HTTP et du HTTPS s'il l'a été en HTTPS. En pratique, cela ne marche pas avec beaucoup de logiciels de syndication et j'ai donc désormais plusieurs flux de syndication <https://www.bortzmeyer.org/deux-flux-syndication.html>.

Bien sûr, une solution à ce problème des AC non connues est d'utiliser DANE (RFC 6698 et mon article à JRES <https://www.bortzmeyer.org/jres-dane-2011.html>). C'est sur ma liste des choses à faire (ceci dit, tant qu'aucun navigateur ne fait du DANE, cela limite l'intérêt).

Bon, assez parlé de ce problème des autorités de certification. Un autre sujet de dispute quand on configure TLS est le choix des algorithmes cryptographiques utilisés. TLS (RFC 5246) permet en effet un large choix, qui va être présenté par le client, et parmi lequel le serveur choisira. Il est donc important de ne **pas** proposer d'algorithmes qui soient trop faibles. Les sélections par défaut des logiciels utilisés pour faire du TLS sont en général bien trop larges. Cela convient au webmestre (car cela augmente les chances d'accepter tous les clients, même les plus anciens ou bogués) mais moins à l'auditeur de sécurité. Celui-ci teste le serveur avec SSLlabs <https://www.ssllabs.com/> ou bien un outil comme SSLyze <https://scottlinux.com/2014/01/28/analyze-ssl-configurations-with-sslyze/> ou encore <https://testssl.sh/> et il vous engueule en disant « quoi, mais vous proposez encore SSLv3, c'est bien trop dangereux ». Pour éviter cette engueulade, on peut mettre dans sa configuration TLS une liste complète des bons algorithmes (on en trouve par exemple dans le document de l'ANSSI cité plus haut et Mozilla en publie une pour GnuTLS <https://wiki.mozilla.org/Security/Server_Side_TLS#GnuTLS_ciphersuite)>) mais cette méthode est longue et pénible et difficile à maintenir car la liste change évidemment dans le temps. Cela ne devrait pas être au webmestre de devenir expert en crypto ! Pour arriver à un réglage raisonnable, j'ai donc choisi une autre solution, en partant des jeux d'algorithmes disponibles dans le logiciel utilisé. Dans mon cas, c'est GnuTLS <https://www.bortzmeyer.org/apache-et-gnutls.html>. Il fournit <http://www.outoforder.cc/projects/apache/mod_gnutls/docs/#GnuTLSPriorities> des identificateurs désignant des jeux d'algorithmes déterminés comme ayant certaines propriétés. Ainsi, NORMAL est le jeu par défaut. Il est très large. Pour le paranoïaque, il existe un jeu SECURE qui est plus raisonnable. On peut donc configurer son Apache ainsi :

2. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6797.txt

```
GnuTLSPriorities SECURE
```

et c'est déjà mieux. Mais, tout l'intérêt de la méthode, est qu'on peut partir d'un jeu prédéfini puis ajouter ou enlever protocoles ou algorithmes. Par exemple, le jeu `SECURE` inclut actuellement le protocole SSLv3, vieux et bogué, et qui n'est apparement nécessaire que pour Internet Explorer version 6. Comme il est peu probable que beaucoup de visiteurs de mon blog utilisent cette horreur d'IE 6, je supprime SSL :

```
GnuTLSPriorities SECURE:-VERS-SSL3.0
```

(Cette directive veut dire : partir de la liste `SECURE` et retirer SSLv3.) Maintenant, si je teste avec testssl <`https://testssl.sh/`>, j'ai :

```
% ./testssl.sh https://www.bortzmeyer.org/
...
 SSLv3      NOT offered (ok)
 TLSv1      offered (ok)
 TLSv1.1    offered (ok)
 TLSv1.2    offered (ok)
 SPDY/NPN   not offered

--> Testing standard cipher lists

 Null Cipher              NOT offered (ok)
 Anonymous NULL Cipher    NOT offered (ok)
 Anonymous DH Cipher      NOT offered (ok)
 40 Bit encryption        NOT offered (ok)
 56 Bit encryption        Local problem: No  56 Bit encryption        configured in /usr/bin/openssl
 Export Cipher (general)  NOT offered (ok)
...
--> Checking RC4 Ciphers

RC4 seems generally available. Now testing specific ciphers...

 Hexcode    Cipher Suite Name (OpenSSL)   KeyExch. Encryption Bits      Cipher Suite Name (RFC)
-------------------------------------------------------------------------------------------------
 [0x05]     RC4-SHA                       RSA        RC4      128       TLS_RSA_WITH_RC4_128_SHA

RC4 is kind of broken, for e.g. IE6 consider 0x13 or 0x0a
...
```

Pas mal. SSLv3 a bien été retiré, le seul gros problème restant est que RC4 est encore proposé alors que cet algorithme a de sérieuses failles. Un dernier effort, pour le supprimer :

```
GnuTLSPriorities SECURE:-VERS-SSL3.0:-ARCFOUR-128:-ARCFOUR-40
```

Et, cette fois, RC4 n'est plus proposé :

```
...
--> Checking RC4 Ciphers

No RC4 ciphers detected (OK)
```

Voilà, j'ai maintenant une configuration TLS raisonnable. Principal manque : augmenter la priorité des algorithmes offrant la "*forward secrecy*", ce sera pour une prochaine fois.

Ah, question outils de test, j'ai bien sûr utilisé SSLlabs <https://www.ssllabs.com/> mais il a deux inconvénients : stupidement, il teste également le nom sans www (et échoue donc souvent), et il ne connait pas CAcert donc la note globale reste à « T » (non documenté). Avec un certificat qu'il connaitrait, <https://www.ssllabs.com/ssltest/analyze.html?d=www.bortzmeyer.org> me donnerait une meilleure note (« "*If trust issues are ignored : A-*" »). J'ai aussi testé le logiciel sslscan (qui est en paquetage sur ma Debian) mais son ergonomie est vraiment horrible, il sort juste une longue liste de trucs testés, sans autre précision, hiérarchisation ou explication.

Quelques autres détails pratiques maintenant. Comme j'aime bien regarder les journaux, j'ai voulu indiquer si la connexion s'était faite en TLS ou pas (je rappelle que, pour l'instant, TLS n'est pas obligatoire sur ce blog). Il existe une variable HTTPS mais qui est spécifique à OpenSSL <http://httpd.apache.org/docs/current/mod/mod_ssl.html> (voir un exemple de son utilisation <http://serverfault.com/questions/359476/how-to-log-the-url-scheme-http-https-in-apache>). Pour GnuTLS, j'ai donc créé deux directives LogFormat :

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %v" complete
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %v TLS" completetls
```

et j'utilise l'une ou l'autre selon le VirtualHost :

```
<VirtualHost *:80>
    CustomLog ${APACHE_LOG_DIR}/access.log complete
...
<VirtualHost *:443>
    CustomLog ${APACHE_LOG_DIR}/access.log completetls
...
```

La visite de SSLlabs est ainsi correctement enregistrée :

```
64.41.200.103 - - [10/Sep/2014:08:43:13 +0000] "GET / HTTP/1.0" 200 285954 \
      "-" "SSL Labs (https://www.ssllabs.com/about/assessment.html)" \
      www.bortzmeyer.org TLS
```

Je n'ai pas encore changé la description OpenSearch <https://www.bortzmeyer.org/opensearch.html> de ce blog. Les recherches <https://www.bortzmeyer.org/recherche-blog.html> sont évidemment sensibles (elles indiquent à un attaquant vos sujets d'intérêt) mais je ne sais pas trop comment dire en OpenSearch « utilise HTTPS si tu peux et HTTP sinon ». Si on met deux éléments url, je ne sais pas lequel est choisi. Donc, il faut que je relise la spécification <http://www.opensearch.org/Specifications/OpenSearch/1.1#The_.22Url.22_element>.

Un piège classique avec X.509 est l'oubli du fait que les certificats ont, parmi leurs métadonnées, une date d'expiration. Comme le mécanisme théorique de révocation des certificats ne marche pas en pratique <https://www.imperialviolet.org/2014/04/29/revocationagain.html>, cette expiration est la seule protection qui empêche un voleur de clé privée de s'en servir éternellement. Il faut donc superviser l'expiration des certificats <https://www.bortzmeyer.org/tester-expiration-certifs.html>, ce que je fais depuis Icinga :

```
# HTTPS
define service{
use                             generic-service
hostgroup_name                  Moi
service_description             HTTPS
check_command                   check_http!-S -C 30,7
}
```

Cette directive indique de se connecter en HTTPS (option `-S`), d'avertir s'il ne reste que 30 jours de validité au certificat et de passer en mode panique s'il ne reste plus que 7 jours. Bien sûr, CAcert prévient automatiquement par courrier lorsqu'un certificat s'approche de l'expiration mais, renouveler le certificat chez l'AC ne suffit pas, encore faut-il l'installer sur le serveur, et c'est cela que teste Icinga.

Merci à Manuel Pégourié-Gonnard pour la correction d'une grosse erreur sur TLS.