RFC 2818 : HTTP Over TLS

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 27 janvier 2018

Date de publication du RFC : Mai 2000

https://www.bortzmeyer.org/2818.html

C'est un RFC très ancien que ce RFC 2818 [1], et qui décrit une technologie qui ne s'est pourtant imposée presque partout que récemment, HTTPS (HTTP sur TLS). Le RFC a depuis été remplacé par le RFC 9110.

Remettons-nous dans le contexte de l'époque : en 2000, la NSA espionnait déjà massivement, mais on n'avait pas encore eu les révélations Snowden. Les pays comme la Russie et la Chine n'avaient sans doute pas encore d'activités d'espionnage sur l'Internet et la France…espionnait le trafic Minitel. L'inquiétude à l'époque n'était pas tellement tournée vers les services secrets officiels mais plutôt vers le méchant pirate qui allait copier les numéros de carte bleue quand on allait les utiliser pour le e-commerce (concept relativement récent à l'époque). HTTPS, le HTTP sécurisé par la cryptographie semblait à l'époque réservé aux formulaires où on tapait un numéro de carte de crédit. Ce n'est qu'à partir des révélations de Snowden que HTTPS a commencé à être déployé systématiquement, et qu'il forme sans doute aujourd'hui la majorité du trafic, malgré les indiscrets de tout bord qui auraient voulu continuer à surveiller le trafic, et qui répétaient « M. Michu n'a pas besoin de chiffrement ». (Au FIC <https://www.forum-fic.com/> en 2016, dans une réunion publique, un représentant d'un grand FAI se désolait que le déploiement massif de HTTPS empêchait d'étudier ce que faisaient ses clients…) Ce succès est en partie dû à des campagnes de sensibilisation (menées par exemple par l'EFF) et à des outils comme HTTPS Everywhere <https://www.eff.org/fr/https-everywhere>.

Revenons au RFC. Il est très court (sept pages) ce qui est logique puisque le gros du travail est fait dans le RFC 5246, qui normalise TLS (c'était le RFC 2246, à l'époque). Car, HTTPS, c'est juste HTTP sur TLS, il n'y a pas grand'chose à spécifier. Notons que le titre du RFC est « HTTP sur TLS », que le texte ne parle que de TLS comme protocole de cryptographie et pourtant, dix-huit ans plus tard, des gens ignorants parlent encore de SSL, le prédécesseur, abandonné depuis longtemps, de TLS.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc2818.txt

Ironie des processus internes de l'IETF, notre RFC n'est pas sur le chemin des normes Internet, il a seulement la qualité « pour information ».

La section 1 résume le RFC : HTTPS, comme TLS, protège le canal et non pas les données. Cela veut dire notamment que HTTPS ne protège pas si on a des sites miroir et que l'un d'eux se fait pirater. (En 2018, on n'a toujours pas de moyen standard de sécuriser les données envoyées sur le Web, malgré quelques briques possiblement utilisables comme XML Signature - RFC 3275 et JOSE <https://www.bortzmeyer.org/jose.html>.)

La section 2 décrit le protocole en une phrase : « Utilisez HTTP sur TLS, comme vous utiliseriez HTTP sur TCP. » Bon, j'exagère, il y a quelques détails. Le client HTTPS doit être un client TLS, donc un client TCP. Il se connecte au serveur en TCP puis lance TLS puis fait des requêtes HTTP par dessus la session TLS, elle-même tournant sur la connexion TCP. Cela se voit bien avec l'option `-v` de curl. D'abord du TCP :

```
% curl -v https://www.bortzmeyer.org/crypto-protection.html
*   Trying 2605:4500:2:245b::42...
* TCP_NODELAY set
* Connected to www.bortzmeyer.org (2605:4500:2:245b::42) port 443 (#0)
```

Puis TLS démarre :

```
* ALPN, offering h2
* ALPN, offering http/1.1
* Cipher selection: ALL:!EXPORT:!EXPORT40:!EXPORT56:!aNULL:!LOW:!RC4:@STRENGTH
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/certs/ca-certificates.crt
  CApath: /etc/ssl/certs
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS change cipher, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* ALPN, server accepted to use http/1.1
* Server certificate:
*  subject: CN=www.bortzmeyer.org
*  start date: Sep 21 06:34:08 2016 GMT
*  expire date: Sep 21 06:34:08 2018 GMT
*  subjectAltName: host "www.bortzmeyer.org" matched cert's "www.bortzmeyer.org"
*  issuer: O=CAcert Inc.; OU=http://www.CAcert.org; CN=CAcert Class 3 Root
*  SSL certificate verify ok.
```

Et enfin on peut faire de l'HTTP :

```
> GET /crypto-protection.html HTTP/1.1
> Host: www.bortzmeyer.org
> User-Agent: curl/7.52.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: Apache/2.4.25 (Debian)
< ETag: "8ba9-56281c23dd308"
< Content-Length: 35753
< Content-Type: text/html
<
<?xml version="1.0" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xml:lang="fr" lang="fr" xmlns="http://www.w3.org/1999/xhtml">
<head>
...
<title>Blog Stéphane Bortzmeyer: La cryptographie nous protège t-elle vraiment de l'espionnage par la NSA ou la
...
</body>
</html>
```

Et on termine :

```
* Curl_http_done: called premature == 0
* Connection #0 to host www.bortzmeyer.org left intact
```

Si on n'aime pas curl, on peut le faire avec le client en ligne de commande de GnuTLS, puis taper les requêtes HTTP à la main :

```
% echo -n "GET /who-buys-porn.html HTTP/1.1\r\nHost: www.bortzmeyer.org\r\nUser-Agent: à la main\r\n\r\n" |  \
  gnutls-cli www.bortzmeyer.org
Processed 167 CA certificate(s).
Resolving 'www.bortzmeyer.org:443'...
Connecting to '2605:4500:2:245b::42:443'...
- Certificate type: X.509
- Got a certificate list of 2 certificates.
- Certificate[0] info:
 - subject `CN=www.bortzmeyer.org', issuer `CN=CAcert Class 3 Root,OU=http://www.CAcert.org,O=CAcert Inc.', seri
Public Key ID:
sha1:82500582b33c12ada8891e9a36192131094bed22
sha256:141954e99b9c88a33af6ffc00db09f5d8b66185a7325f45cbd45ca3cb47d63ce
Public key's random art:
+--[ RSA 2048]----+
|o=o oo.          |
|O..+             |
|+Bo              |
|E= o .           |
|*.+ . . S        |
|+o     .         |
|o.+              |
|o=               |
|. .              |
+-----------------+

- Certificate[1] info:
 - subject `CN=CAcert Class 3 Root,OU=http://www.CAcert.org,O=CAcert Inc.', issuer `EMAIL=support@cacert.org,CN=
- Status: The certificate is trusted.
```

```
- Description: (TLS1.2)-(ECDHE-RSA-SECP256R1)-(AES-256-GCM)
- Session ID: A6:A6:CE:32:08:9D:B8:D2:EF:D6:C7:D4:85:B4:39:D7:0D:04:83:72:6F:87:02:50:44:B7:29:64:E5:69:0B:0
- Ephemeral EC Diffie-Hellman parameters
 - Using curve: SECP256R1
 - Curve size: 256 bits
- Version: TLS1.2
- Key Exchange: ECDHE-RSA
- Server Signature: RSA-SHA256
- Cipher: AES-256-GCM
- MAC: AEAD
- Compression: NULL
- Options: safe renegotiation,
- Handshake was completed

- Simple Client Mode:

HTTP/1.1 200 OK
Server: Apache/2.4.25 (Debian)
Content-Length: 8926
Content-Type: text/html
```

Et, vu par tshark, ça donne d'abord TCP :

```
1 0.000000000 2a01:e35:8bd9:8bb0:dd2c:41a5:337f:a21f → 2605:4500:2:245b::42 TCP 94 45502 → 443 [SYN] Seq=0
2 0.080848074 2605:4500:2:245b::42 → 2a01:e35:8bd9:8bb0:dd2c:41a5:337f:a21f TCP 94 443 → 45502 [SYN, ACK]
3 0.080902511 2a01:e35:8bd9:8bb0:dd2c:41a5:337f:a21f → 2605:4500:2:245b::42 TCP 86 45502 → 443 [ACK] Seq=1
```

Puis TLS :

```
4 0.084839464 2001:db8:57aa:8bb0:dd2c:9af:222f:a21f → 2605:4500:2:245b::42 SSL 351 Client Hello
5 0.170845913 2605:4500:2:245b::42 → 2001:db8:57aa:8bb0:dd2c:9af:222f:a21f TCP 86 443 → 45502 [ACK] Seq=1
6 0.187457346 2605:4500:2:245b::42 → 2001:db8:57aa:8bb0:dd2c:9af:222f:a21f TLSv1.2 1494 Server Hello
7 0.187471721 2001:db8:57aa:8bb0:dd2c:9af:222f:a21f → 2605:4500:2:245b::42 TCP 86 45502 → 443 [ACK] Seq=26
8 0.187583400 2605:4500:2:245b::42 → 2001:db8:57aa:8bb0:dd2c:9af:222f:a21f TLSv1.2 2320 Certificate, Server
9 0.187600261 2001:db8:57aa:8bb0:dd2c:9af:222f:a21f → 2605:4500:2:245b::42 TCP 86 45502 → 443 [ACK] Seq=26
10 0.199416730 2001:db8:57aa:8bb0:dd2c:9af:222f:a21f → 2605:4500:2:245b::42 TLSv1.2 212 Client Key Exchange
11 0.283191938 2605:4500:2:245b::42 → 2001:db8:57aa:8bb0:dd2c:9af:222f:a21f TLSv1.2 360 New Session Ticket,
```

C'est fait, TLS a démarré (notez le nombre de paquets qu'il a fallu échanger pour cela : TLS accroit la latence <https://www.bortzmeyer.org/latence.html>, et beaucoup de travaux actuellement à l'IETF visent à améliorer ce point). On peut maintenant faire du HTTP :

```
12 0.283752212 2001:db8:57aa:8bb0:dd2c:9af:222f:a21f → 2605:4500:2:245b::42 TLSv1.2 201 Application Data
13 0.365992864 2605:4500:2:245b::42 → 2001:db8:57aa:8bb0:dd2c:9af:222f:a21f TLSv1.2 1494 Application Data
```

Comment, on ne voit pas le HTTP, juste le "*Application Data*" générique de TLS ? Oui, c'est exprès, c'est bien à ça que sert TLS.

Bon, ce n'est pas tout d'ouvrir une session, il faut aussi la fermer. Le RFC précise qu'il faut utiliser la fermeture TLS avant la fermeture TCP, pour être sûr que toutes les données ont été transmises. Le cas des fermetures de session trop brutales forme d'ailleurs l'essentiel du RFC (que faire si la connexion TCP est rompue avant la session TLS ? Cela peut avoir des conséquences de sécurité, par exemple s'il n'y a pas d'en-tête HTTP `Conent-Length:` et que la connexion TCP est coupée, on ne peut pas savoir si on

a tout reçu ou bien si un attaquant a généré une fausse coupure de connexion TCP qui, contrairement à la fermeture TLS, n'est pas authentifiée, cf. RFC 5961.

Reste la question du port à utiliser. Comme vous le savez sans doute, HTTPS utilise le port 443. Il est enregistré à cet effet à l'IANA <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml>. Notez que `gnutls-cli`, utilisé plus haut, utilise ce port par défaut.

Et le plan d'URI à utiliser pour HTTPS ? Là aussi, vous le connaissez, c'est `https://` donc par exemple, , ou . Il est également enregistré à l'IANA <https://www.iana.org/assignments/uri-schemes/uri-schemes.xml>. Notez que le RFC 7230 a apporté, depuis, quelques changements (sans toutefois remplacer complètement le vieux RFC 2818). Le principal est que la définition de ce plan `https://` pour les URI est désormais dans le RFC 7230, et plus détaillée que la description informelle du vieux RFC.

Mais ça n'est pas tout d'avoir une connexion chiffrée...Comme le notent plusieurs auteurs « TLS permet d'avoir une session sécurisée avec un ennemi ». En effet, si vous n'authentifiez pas la machine à l'autre bout, TLS ne sera pas très utile. Vous risquerez d'être victime de l'attaque de l'homme du milieu où Mallory se place entre Alice et Bob, prétendant être Bob pour Alice et Alice pour Bob. Il est donc très important d'authentifier le partenaire, et c'est là la principale faiblesse de HTTPS. On utilise en général un certificat X.509 (ou, plus rigoureusement, un certificat PKIX, cf. RFC 5280). Comme HTTPS part en général d'un URI, le client a dans cet URI le nom de domaine du serveur. C'est ce nom qu'il doit comparer avec ce qu'il trouve dans le certificat (plus exactement, on regarde le `subjectAltName` si présent, puis le `Common Name`, CN, mais rappelez-vous que le RFC est vieux, les règles ont évolué depuis <https://cabforum.org/baseline-requirements-documents/>). D'autres techniques d'authentification sont possibles, comme l'épinglage de la clé (RFC 7469), qu'on peut combiner avec les clés nues du RFC 7250.

Si quelque chose ne va pas dans l'authentification, on se récupère un avertissement du navigateur, proposant parfois de passer outre le problème. Le RFC notait déjà, il y a dix-huit ans, que c'était dangereux car cela laisserait la session potentiellement sans protection. Globalement, depuis la sortie du RFC, les navigateurs sont devenus de plus en plus stricts, rendant ce débrayage de la sécurité plus difficile.

Voici un exemple de problème d'authentification. Le Centre Hospitalier de Bois-Petit <http://www.ch-boispetit.fr/> redirige automatiquement les visiteurs vers la version HTTPS de son site, qui n'a pas le bon nom dans le certificat. Ici, la protestation de Firefox (notez la possibilité d'ajouter une exception manuellement) :

curl râle également, bien sûr :

```
% curl https://www.ch-boispetit.fr/
curl: (51) SSL: no alternative certificate subject name matches target host name 'www.ch-boispetit.fr'
```

Une variante de cette erreur se produit lorsque le site est accessible par plusieurs noms en HTTP mais que tous ne sont pas authentifiables en HTTPS. Par exemple, pour le quotidien Le Monde, et marchent mais, en HTTPS, seul `www.lemonde.fr` est dans le certificat. (Même chose, aujourd'hui, pour .) D'où le refus de Chromium :

D'autres erreurs sont possibles avec la gestion des certificats, la plus fréquente étant sans doute le certificat expiré <https://www.bortzmeyer.org/tester-expiration-certifs.html>. Si vous voulez regarder avec votre navigateur Web des exemples d'erreurs, l'excellent site vous en propose plein, faites exprès.

Le RFC note à juste titre que l'URI de départ vient souvent d'une source qui n'est pas digne de confiance. Si vous suivez un lien qui est dans vos signets, ou que vous le tapez vous-même en faisant attention, l'URI est digne de confiance, et la vérification du certificat vous protégera. Mais si vous cliquez sur un lien dans un message envoyé par une nommée natacha@viagra-pharmacy.ru annonçant « Agrandissez votre pénis », ou même un lien dans un message **prétendant** venir de votre patron et annonçant « dossier très urgent à lire » (rappelez-vous que le courrier électronique n'est pas authentifié, sauf si on utilise PGP ou équivalent), alors, l'URI n'est pas sûr et HTTPS ne vous protégera pas (sauf si vous scrutez avec soin l'URI dans la barre d'adresses du navigateur, ce que personne ne fait et, de toute façon, c'est parfois trop tard). C'est ce qu'on appelle le hameçonnage et c'est une attaque fréquente, contre laquelle toute la cryptographie du monde ne vous protège pas. (Ceci dit, à l'heure actuelle, spammeurs et hameçonneurs ne se sont pas mis massivement à HTTPS, la plupart du temps, ils en restent au bon vieil HTTP.)

Notez qu'on n'a parlé que de la vérification par le client de l'identité du serveur (qui utilise le fait que le client connait le nom de domaine du serveur). La vérification par le serveur de l'identité du client est également possible (envoi d'un certificat par le client) mais elle nécessite que le serveur ait une base des clients connus (ou qu'il accepte tous les certificats d'une AC donnée).

Après cet article, et compte tenu du fait que HTTPS existe formellement depuis si longtemps, vous devez vous dire que ce blog que vous lisez est accessible en HTTPS, non ? Eh bien, il l'est <https://www.bortzmeyer.org/https-blog.html> **mais** je ne publie pas les URI en https:// et je ne fais pas de redirection automatique vers la version HTTPS. En effet, j'utilise une AC gratuite et facile d'usage mais que la plupart des vendeurs n'incluent pas <https://www.bortzmeyer.org/cacert.html> dans leur magasin des AC (ce qui est une décision arbitraire : regardez le magasin des AC via votre navigateur Web, et voyez si vous faites confiance aux gouvernements chinois et turcs, ainsi qu'aux entreprises privées qui ne pensent qu'au profit). Tant que ce problème durera, je ne pourrais pas faire de HTTPS par défaut. En attendant, si vous voulez voir ce blog sur HTTPS, sur une Debian, faites juste sudo aptitude install ca-cacert (puis redémarrez le navigateur), sur les autres systèmes, allez en et installez le certificat de CAcert.

Au fait, si quelqu'un a des références de bonnes études quantitatives sur le déploiement de HTTPS entre 2000 et aujourd'hui, je suis preneur. Je connais pour l'instant la télémétrie de Firefox. Elle est affichée ici <https://letsencrypt.org/stats/#percent-pageloads> (mais malheureusement uniquement depuis 2014) et montre aujourd'hui 70 % de pages Web en HTTPS.