RFC 5246 : The Transport Layer Security (TLS) Protocol Version 1.2

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 24 août 2008

Date de publication du RFC : Août 2008

https://www.bortzmeyer.org/5246.html

Le protocole de cryptographie TLS, autrefois nommé SSL, est un des protocoles de l'IETF les plus utilisés. À chaque seconde, des milliers de connexions TCP, en général HTTP, sont protégées contre l'écoute, et même parfois authentifiées par TLS. Ce RFC met à jour l'ancienne norme, le RFC 4346[1]. (Depuis, la version 1.3, très différente, a été normalisée dans le RFC 8446.)

Aujourd'hui, plus personne n'aurait l'idée de faire circuler un mot de passe en clair sur le réseau, car il serait très facile à capturer (par exemple avec un programme comme dSniff). TLS fournit la confidentialité à une connexion TCP. (TLS peut aussi fonctionner avec d'autres protocoles de transport comme SCTP - RFC 3436 ou bien UDP - RFC 6347.)

L'authentification par mot de passe a par ailleurs des défauts, notamment que le mot de passe soit réutilisable. Si l'utilisateur a été victime d'un faux serveur, par exemple par suite d'un hameçonnage, il donne son mot de passe et le méchant qui contrôle le faux serveur peut l'utiliser sur le vrai site. Pour traiter ce problème, TLS fournit une solution d'authentification reposant sur les certificats X.509 ou bien sur les clés OpenPGP (RFC 5081). TLS permet aussi bien au client d'authentifier le serveur qu'au serveur d'authentifier le client (cette deuxième possibilité est à l'heure actuelle peu utilisée).

Les certificats X.509 sont composés de la partie publique d'une clé asymétrique et de métadonnées, comme la date d'expiration. Ils sont signés par une CA, qui a elle-même un certificat signé par une CA et ainsi de suite jusqu'à arriver aux CA racines installés dans le logiciel. La plupart du temps, l'utilisateur n'a jamais examiné ces certificats des CA racine et n'a aucune idée de leur fiabilité. C'est l'un des gros points faibles de l'authentification par TLS / X.509.

TLS n'est pas une solution miracle, il n'en existe pas en sécurité informatique. Par exemple, TLS ne protège pas dans les cas suivants :

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc4346.txt

— Si un programme espion tourne sur la machine de l'utilisateur (ce qui est le cas de beaucoup, peut-être la plupart des machines MS-Windows),
— Si le serveur est méchant ou mal protégé. Par exemple, un numéro de Carte Bleue transmis par TLS n'est protégé que pendant le voyage ! Si le serveur à l'autre bout stocke ces numéros et se fait pirater, TLS n'y pourra rien.

Passons maintenant au RFC proprement dit. Il fait plus de cent pages et pourtant de nombreux points n'y sont pas, comme la définition des algorithmes cryptographiques utilisés. La sécurité est un domaine compliqué, d'autant plus qu'une petite erreur peut annuler toute la sécurité de même que l'oubli de fermer à clé peut rendre inutile une lourde porte blindée.

TLS est composé de deux parties (section 1 du RFC), le "*Record Protocol*", protocole de bas niveau qui transmet des messages en les chiffrant pour la confidentialité et l'intégrité et le "*Handshake protocol*", bâti sur le premier, et qui se charge entre autres de négocier les paramètres de la session.

En effet, TLS fonctionne en négociant au début un algorithme de chiffrement ainsi qu'une clé cryptographique de cet algorithme pour la session. Le "*Record protocol*" va ensuite chiffrer les messages avec cette clé.

La section 4 du RFC détaille le langage de présentation. L'IETF n'a pas de langage unique de spécification, chaque RFC utilise des outils différents pour décrire ses protocoles ou algorithmes. TLS crée donc le sien, proche du C. Par exemple la section 4.4 décrit les nombres, tous créés à partir d'un type `unit8`, qui stocke un octet. La section 4.5 décrit les types énumérés comme `enum { null, rc4, 3des, aes } BulkCipherAlgorithm;` (le type qui sert à définir les valeurs de l'algorithme de cryptographie symétrique).

La section 6 décrit en détail le "*Record protocol*". C'est un protocole qui va effectuer le chiffrement, mais aussi la compression, la fragmentation, etc.

La section 7 couvre le protocole de négociation ("*Handshake protocol*"). C'est le protocole qui permet l'authentification du serveur par le client (ou, plus rarement, le contraire) et la sélection des options, par exemple l'algorithme de chiffrement utilisé. L'authentification se fait en général en présentant un certificat X.509, certificat signé par un tiers de confiance (RFC 5280, en pratique, le tiers « de confiance » est une des CA installées par défaut avec le logiciel, dont personne ne sait pourquoi elles ont été choisies). Voici par exemple le résultat de ce protocole, tel qu'affiché par la commande `gnutls-cli` (qui est livrée avec GnuTLS) :

```
% gnutls-cli -d 9 www.example.org
Connecting to '192.0.2.20:443'...
...
|<3>| HSK[80708e8]: Selected cipher suite: DHE_RSA_AES_256_CBC_SHA
```

Le chiffrement AES en mode CBC a été choisi. `gnutls-cli` permet de voir le passage de tous les messages TLS de ce protocole, tels qu'ils sont décrits dans la section 7.4 :

```
enum {
    hello_request(0), client_hello(1), server_hello(2),
    certificate(11), server_key_exchange (12),
    certificate_request(13), server_hello_done(14),
    certificate_verify(15), client_key_exchange(16),
    finished(20), (255)
} HandshakeType;
```

Comme précisé dans la section 7.3, ce protocole de négociation a aussi ses propres vulnérabilités. Au démarrage, il n'y a évidemment pas d'algorithme de chiffrement ou de clé de session sélectionnés. Comme le précise la section 7.4.1, TLS démarre donc sans chiffrement. Un attaquant actif, capable de modifier les paquets IP, peut changer la liste des algorithmes acceptés, avant que le chiffrement ne soit actif, de façon à sélectionner un algorithme faible. (Le remède est de refaire la négociation une fois le chiffrement activé.)

Comme souvent avec TLS, le protocole est extensible. La section 7.4.1.4 décrit par exemple les extensions à l'annonce que font les machines TLS, possibilité qui est utilisée dans le RFC 5081 pour ajouter les clés OpenPGP aux certificats X.509. (Les extensions existantes étaient initialement décrites dans le RFC 4366 et sont désormais dans le RFC 6066. Le registre complet des extensions <https://www.iana.org/assignments/tls-extensiontype-values> est maintenu à l'IANA.)

La section 9 indique l'algorithme de chiffrement obligatoire, RSA avec AES. Il est nécessaire de définir un algorithme obligatoire, autrement, deux mises en œuvre légales de TLS pourraient être incompatibles si les deux ensembles d'algorithmes n'avaient aucun élément en commun.

Par rapport à son prédécesseur, le RFC 4346, qui normalisait TLS 1.1, les changements sont de peu d'importance (la liste figure en section 1.2). Un client TLS 1.2 peut interagir avec un serveur 1.1 et réciproquement (voir l'annexe E pour les détails.) Il s'agit en général de clarifications ou bien du changement de statut de certains algorithmes cryptographiques. En effet, la cryptographie bouge et certains algorithmes, qui paraissaient solides à une époque, sont vulnérables aux attaques modernes (c'est par exemple ce qui est arrivé à MD5 et SHA1). C'est pour cela que l'algorithme « RSA avec AES » fait désormais partie des algorithmes obligatoires et que par contre IDEA et DES sont partis.

Contrairement à IPsec (RFC 4301), TLS fonctionne entre la couche de transport et les applications. Il n'est donc pas invisible aux applications, qui doivent être explicitement programmées pour l'utiliser. Pour cela, il n'existe malheureusement pas d'API standard. OpenSSL, l'implémentation la plus populaire de TLS, a la sienne <http://www.openssl.org/docs/ssl/ssl.html>, son concurrent GnuTLS en a une autre <http://www.gnu.org/software/gnutls/manual/html_node/How-to-use-GnuTLS-in.html>, etc. (GnuTLS dispose d'une couche d'émulation de OpenSSL mais elle est incomplète.)

Un programme comme echoping <http://echoping.sourceforge.net/> qui peut utiliser aussi bien OpenSSL que GnuTLS, doit donc recourir aux services du préprocesseur C, par exemple :

```
#ifdef OPENSSL
        ...
        SSL_set_fd(sslh, sockfd);
        if (SSL_connect(sslh) == -1)
#endif
#ifdef GNUTLS
...
        gnutls_transport_set_ptr(session,
                                 (gnutls_transport_ptr) (long) sockfd);
        tls_result = gnutls_handshake(session);
...
#ifdef OPENSSL
        if ((rc = SSL_write(sslh, sendline, n)) != n) {
...
#ifdef GNUTLS
        if ((rc = gnutls_record_send(session,
                                     sendline, strlen (sendline))) != n) {
...
```

Une autre solution pour une application qui veut faire du TLS sans se fatiguer est de compter sur le fait que TLS est indépendant du protocole utilisé par l'application et qu'on peut donc simplement créer un tunnel TLS, par exemple avec stunnel. Ainsi, on n'a même pas besoin de modifier le source de l'application.

Le déploiement de protocoles est toujours lent dans l'Internet. TLS 1.2, décrit dans ce RFC, ne s'est retrouvé dans le navigateur Firefox que cinq ans après <https://bugzilla.mozilla.org/show_bug.cgi?id=861266>.