RFC 7457 : Summarizing Known Attacks on TLS and DTLS

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 10 février 2015

Date de publication du RFC : Février 2015

https://www.bortzmeyer.org/7457.html

Depuis quelques années, on entend souvent parler de failles dans TLS ou dans les mises en œuvre de TLS, comme OpenSSL. Un groupe de travail a été créé à l'IETF pour essayer de durcir la sécurité de TLS, le groupe UTA (Using TLS in Applications) <https://tools.ietf.org/wg/uta>. Son premier RFC est un rappel de ces récentes failles. C'est l'occasion de réviser BEAST, CRIME ou Heartbleed.

Le protocole de cryptographie TLS (autrefois nommé SSL) est normalisé dans le RFC 5246[1]. Les attaques récentes contre TLS sont très variées : mauvaise utilisation de TLS par les applications (la principale motivation pour la création du groupe UTA <https://tools.ietf.org/wg/uta>, et son centre d'intérêt principal), bogues dans un programme mettant en œuvre TLS, erreurs dans le protocole, erreurs dans un des algorithmes cryptographiques qu'il utilise... Ce RFC dresse une liste des problèmes de sécurité de ces dernières années, mais on peut être sûr que la liste va s'allonger. Comme le dit la NSA, « "*Attacks always get better ; they never get worse*" » (cf. RFC 4270).

Le gros morceau du RFC est sa section 2, qui liste les problèmes. Par exemple, la plus évidente est d'empêcher complètement l'usage de TLS (ce qu'on nomme parfois SSL stripping). Par exemple, si un client HTTP se connecte en HTTP et compte sur le serveur pour le rediriger en HTTPS, un attaquant actif peut modifier la redirection, empêchant ainsi TLS de démarrer. Le "*SSL stripping*" est donc un cas particulier des attaques par repli ("*downgrade attacks*"). Normalement, HSTS (RFC 6797) est la bonne solution contre cette attaque.

Pour HTTP, il est habituel de démarrer directement en TLS. Pour les protocoles où on démarre sans TLS et où on utilise ensuite une commande `STARTTLS` pour commencer à chiffrer (comme SMTP), une attaque par repli analogue est possible en supprimant la commande `STARTTLS`. La seule protection possible serait un équivalent de HSTS (le RFC oublie de mentionner DANE, qui pourrait être utilisé à des

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

fins de signalisation). Autre problème avec STARTTLS, les attaques où le tampon d'entrée n'est pas vidé après le passage à TLS <https://www.bortzmeyer.org/tls-vidage-tampon.html>, ce qui fait que les commandes injectées avant que TLS ne sécurise la connexion sont exécutées après, en les croyant sûres (CVE-2011-0411 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0411>). Ce n'est pas une faille du protocole mais de l'application.

Autre attaque, avec un nom rigolo, cette fois, BEAST <https://www.bortzmeyer.org/beast-tls.html> (CVE-2011-3389 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3389>). Elle ne touchait que TLS 1.0 (version dépassée depuis longtemps) et seulement le mode CBC.

Plusieurs attaques sont du type "*padding oracle*". C'est le cas de Lucky Thirteen (CVE-2013-0169 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-0169>). Elle peut être combattue en utilisant un mode de chiffrement intègre, ou bien en faisant le chiffrement avant de calculer le MAC (mécanisme dit « encrypt-then-MAC » et recommandé pour TLS par le RFC 7366). Une autre attaque "*padding oracle*", Poodle (CVE-2014-3566 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3566> ou bien l'article d'Andréa Fradin <http://www.slate.fr/story/93437/poodle-bug-caniche-peut-vous-devoiler>), qui ne touche que des versions encore plus anciennes (avant que cela ne s'appelle TLS) et n'a pas de solution connue (à part arrêter d'utiliser des versions archaïques de SSL).

Le protocole TLS est complexe et plein de failles se cachent dans des détails subtils. Par exemple, l'attaque "*triple handshake*" <https://secure-resumption.com/> (CVE-2014-1295 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1295>) permet de faire en sorte que deux sessions TLS utilisent les mêmes clés, autorisant ainsi plusieurs autres attaques.

Passons à des attaques plus cryptographiques, celles portant sur l'algorithme RC4. RC4 est utilisé depuis longtemps, a des faiblesses connues depuis longtemps (voir le RFC pour une bibliographie comportant plusieurs articles), et ces faiblesses commencent à devenir exploitables en pratique (aujourd'hui, elles requièrent encore trop de calculs). Il ne faut donc plus utiliser RC4 (RFC 7465).

La compression des données dans TLS apporte ses propres attaques comme CRIME (CVE-2012-4929 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4929>). Pour les empêcher, on peut couper la compression ou, pour certaines attaques, choisir des solutions au niveau de l'application.

Autre propriété de TLS qui semblait pratique quand elle a été définie, mais qui s'est avérée dangereuse, la renégociation <https://www.bortzmeyer.org/tls-renego.html> (CVE-2009-3555 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555>). La solution a été fournie par le RFC 5746.

TLS utilisant presque toujours des X.509 pour authentifier le serveur (et parfois le client), il n'est pas étonnant que les failles de X.509, ou d'un des algorithmes utilisés dans les certificats, se retrouvent dans la liste des problèmes de sécurité avec TLS. C'est le cas par exemple de l'attaque RSA de Klima, V., Pokorny, O., and T. Rosa, « "*Attacking RSA-based sessions in SSL/TLS*" <https://eprint.iacr.org/2003/052.pdf> ». ou de celle de Brubaker, C., Jana, S., Ray, B., Khurshid, S., et V. Shmatikov, « "*Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations*" <https://www.cs.utexas.edu/~shmat/shmat_oak14.pdf> ».

Une autre attaque possible est lorsque l'attaquant met la main sur la clé privée a posteriori, et qu'il a enregistré le trafic chiffré. Avec la plupart des algorithmes utilisés par TLS, la connaissance de la clé privée permet de déchiffrer le trafic passé, qui peut être encore intéressant. Cela permet, par exemple,

l'utilisation de Wireshark <https://www.bortzmeyer.org/crypto-debug.html> pour analyser un trafic HTTPS. Mais c'est dangereux pour la vie privée : un attaquant patient et ayant beaucoup de place sur ses disques durs pourrait enregistrer tout le trafic chiffré, attendant le moment où il entre en possession des clés privées (ce que fait la NSA, cf. RFC 7258). On peut se protéger contre ces attaques en sécurisant mieux ses clés privées (un HSM ?) mais le mieux est sans doute de passer aux algorithmes qui fournissent la "*perfect forward secrecy*".

Même lorsque le protocole est parfait, des failles peuvent apparaitre en raison de bogues dans les programmes qui mettent en œuvre TLS. On peut même dire qu'il y a bien plus de bogues d'implémentation que de protocole, et elles sont en général plus facilement exploitables, et avec des conséquences plus graves. Ces failles peuvent être dans les bibliothèques TLS (comme OpenSSL, qui en a connu beaucoup, dont la fameuse Heartbleed, CVE-2014-0160 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160>, cf. RFC 6250 ou bien l'article de Fradin <http://www.slate.fr/monde/85743/bug-heartbleed-internet-chiffrement>) ou dans les programmes qui utilisent ces bibliothèques (une excellente étude à ce sujet est celle de Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., et V. Shmatikov, « "*The most dangerous code in the world : validating SSL certificates in non-browser software*" <http://doi.acm.org/10.1145/2382196.2382204> »). Une liste non-exhaustive de ces bogues dans les applications :

— Absence de vérification du certificat, comme l'ont fait pendant longtemps les bibliothèques Python (voir le CVE-2013-2191 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-2191>). Ce n'est pas une faille du protocole mais de l'application ou des bibliothèques.
— Si on vérifie le certificat, absence du test du nom du serveur dans le certificat. La faille « "*goto fail*" <http://seenthis.net/messages/230904> » était de cette catégorie. Voir mon exposé à Devoxx <http://blog.soat.fr/2014/04/devoxx-2014-utiliser-tls-sans-se-tromper-une-conf> (et ses transparents (en ligne sur https://www.bortzmeyer.org/files/bortzmeyer-tls-devoxx.odp)).

Il n'y a pas que des attaques dues aux bogues du protocole ou des implémentations. Un méchant peut aussi utiliser les simples attaques par déni de service. TLS impose davantage de calculs aux machines qui communiquent et cela peut être utilisé pour le déni de service. Si les processeurs ont progressé (rendant réaliste l'idée d'activer systématiquement TLS pour toutes les connexions), un attaquant disposant d'un grand botnet peut toujours épuiser les capacités de sa victime, en lui imposant des calculs cryptographiques complexes.

De même que les programmeurs d'application peuvent se tromper, mal interpréter les complexes API des bibliothèques TLS, et introduire ainsi des failles de sécurité, les utilisateurs finaux peuvent également prendre des décisions qui annulent la sécurité de TLS, quelle que soit la qualité des protocoles et des logiciels. L'**utilisabilité** est un élément de la sécurité, si on ne veut pas que M. Michu fasse des erreurs. Par exemple, les logiciels permettent d'accepter un mauvais certificat et les utilisateurs disent presque toujours Oui (et souvent pour des raisons rationnelles <https://www.bortzmeyer.org/rational-security.html>). Ici, un exemple avec le logiciel de messagerie instantanée Pidgin :

Lutter contre ce problème nécessitera à la fois HSTS (pour priver de choix l'utilisateur), de l'épinglage des clés, et de meilleurs interfaces utilisateur, une tâche très complexe.

Si vous voulez vous documenter sur ces problèmes TLS, le RFC recommande l'article de Sarkar, P. et S. Fitzgerald, « "*Attacks on SSL, a comprehensive study of BEAST, CRIME, TIME, BREACH, Lucky13 and RC4 biases*" <https://www.isecpartners.com/media/106031/ssl_attacks_survey.pdf> ».

Un autre mécanisme TLS contre les attaques par repli est celui décrit dans le RFC 7507.