La crypto n'a pas que des avantages

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 4 octobre 2013. Dernière mise à jour le 14 février 2015

https://www.bortzmeyer.org/crypto-debug.html

Suite aux révélations du héros Edward Snowden, bien des gens ont pris conscience de ce que tous les experts en sécurité annonçaient depuis longtemps : les services d'espionnage espionnent et ne respectent aucune limite. Notamment, tout le trafic envoyé sur l'Internet peut être écouté, si on ne prend pas de précautions particulières. La solution technique la plus souvent citée est l'usage systématique de la cryptographie. Ce choix est tout à fait justifié. Mais il ne faut pas s'imaginer qu'il va être gratuit : tout chiffrer va faire perdre certaines possibilités, notamment en matière de débogage.

Cet article a été motivé par une formation où on programmait des accès à un service réseau, via une API qui reposait sur HTTPS. Un moment, on avait un doute sur ce qu'on envoyait, quelqu'un a dit « on va utiliser Wireshark pour examiner ce qu'on envoie vraiment » et paf : à cause du S de HTTPS, la session était entièrement chiffrée par TLS et Wireshark ne pouvait pas aider. Une décision de sécurité parfaitement justifiée (ne permettre l'accès qu'en HTTPS) a fait perdre un remarquable outil de débogage des applications HTTP.

Bien sûr, compte tenu des révélations de Snowden citées plus haut, il n'y a guère le choix. Même si on n'est pas sûr que la cryptographie protège bien contre un adversaire de la puissance de la NSA <https://www.bortzmeyer.org/crypto-protection.html>, ne pas se protéger serait une folie, puisque la NSA et tous les espions plus petits pourraient alors regarder le contenu du trafic sans problème. Donc, il faut chiffrer. Mais, personnellement, je regrette que les géniaux outils de débogage réseau comme tcpdump et Wireshark soient de moins en moins utiles à cause du « tout chiffrement ».

Alors, certains et certaines vont me dire « mais il existe des outils qui savent déchiffrer le trafic chiffré, si on leur fournit la(les) clés privée(s), par exemple Wireshark ». Mais ce n'est plus vrai non plus. Voyons d'abord les outils disponibles :

- tcpdump ne sait déchiffrer que l'IPsec, protocole peu utilisé.
- ssldump <http://www.rtfm.com/ssldump/Ssldump.html> sait déchiffrer le TLS (autrefois nommé SSL).
- Wireshark sait déchiffrer le TLS.
- Aucun ne sait déchiffrer le SSH.

Naturellement, ssldump et Wireshark vont avoir besoin de la clé privée du serveur pour cela (autrement, TLS ne servirait à rien). Si on utilise ssldump sans cette clé privée, on voit la négociation TLS :

```
% ssldump -d -r /tmp/tls.pcap
...
1 1  0.2319 (0.2319)  C>S  Handshake
      ClientHello
        Version 3.1
        cipher suites
        TLS_DHE_RSA_WITH_AES_128_CBC_SHA
...
1 2  0.4557 (0.2238)  S>C  Handshake
      ServerHello
        Version 3.1
```

Mais plus rien ensuite :

```
...
1 11 0.8215 (0.0110)  C>S  application_data
1 12 1.6280 (0.8065)  S>C  application_data
1 13 1.6845 (0.0564)  S>C  application_data
1 14 1.6993 (0.0148)  S>C  application_data
...
```

(Notez quand même que la négociation TLS se passe en clair, ce qui peut donner des informations à un espion.)

Si on copie la clé privée `server.key` sur le serveur TLS et qu'on permet à ssldump de s'en servir :

```
% ssldump -d -k server.key -r /tmp/tls.pcap
```

On ne récupère rien de plus ! C'est encore par la faute de la NSA. Celle-ci stocke apparemment les communications chiffrées sur ses disques durs, dans l'espoir de pouvoir les déchiffrer plus tard, soit par les progrès de la cryptanalyse, soit simplement en obtenant la clé privée (par espionnage, injonction d'un tribunal, etc). Les sessions TLS sont donc vulnérables à ces attaques du futur, ce qui a mené au concept de PFS ("*Perfect Forward Secrecy*"). La PFS est la propriété comme quoi un attaquant qui a copié la session **et** qui a obtenu la clé privée ne pourra quand même pas déchiffrer le message. Elle est mise en œuvre dans TLS via des algorithmes comme ceux dont le nom contient DHE (Diffie-Hellman éphémère), comme le `TLS_DHE_RSA_WITH_AES_128_CBC_SHA` de l'exemple plus haut. Avec cette procédure DHE, les deux parties (le client et le serveur TLS) se mettent d'accord sur des clés qui ne sont pas transmises et donc pas conservées dans les enregistrements de trafic. (Les autres algorithmes sont souvent nommés « RSA statique ». Avec eux, une clé de session est générée et envoyée, après chiffrement RSA, au pair. Elle sera donc accessible dans le trafic capturé.) Ainsi, ssldump et Wireshark ne peuvent rien faire (Wireshark affiche piteusement « "*Entire conversation (0 bytes)*" »). Les mises en œuvre modernes de TLS choisissent souvent ces algorithmes et, si vous avez du TLS récent et que vous n'avez pas changé la configuration, vous avez souvent du PFS par défaut... et donc pas de débogage possible. Vous voyez en général qu'on a le PFS lorsque la négociation TLS comprend un `ServerKeyExchange` (section 7.4.3 du RFC 5246 [1]). Avec ssldump :

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

```
1 4  0.4582 (0.0025)  S>C  Handshake
      ServerKeyExchange
```

Au fait, pour comparaison, une session TLS où on n'a **pas** employé la PFS, le serveur ne la gérant pas :

```
...
1 8  0.7662 (0.1631)  S>C  ChangeCipherSpec
1 9  0.7664 (0.0002)  S>C  Handshake
      Finished
1 10 0.7762 (0.0097)  C>S  application_data
    ---------------------------------------------------------------
    GET / HTTP/1.0
    Host: www.example.net
    Accept: text/html, text/plain, text/css, text/sgml, */*;q=0.01
    Accept-Encoding: gzip, compress, bzip2
    Accept-Language: en
    User-Agent: Lynx/2.8.8dev.15 libwww-FM/2.14 SSL-MM/1.4.1 GNUTLS/2.12.20
...
```

Cette fois, ssldump peut déchiffrer la communication en HTTP. Ne tentez pas cela avec le paquetage ssldump de Debian, il a une bogue énorme <http://bugs.debian.org/661276> et jamais réparée, qui fait qu'on ne peut pas l'utiliser pour déchiffrer.

Donc, une nouvelle fois, la sécurité n'a pas été gratuite. La PFS est indispensable contre les attaquants modernes, mais elle fait perdre la possibilité d'utiliser les outils de débogage avec déchiffrement. Le caractère très ouvert et très visible de l'Internet, qui m'avait tant facilité la vie lorsque j'avais appris TCP/IP avec un "*sniffer*", en a pris un coup.

Des bonnes lectures :

— La documentation de ssldump explique bien le problème du « "*Ephemeral keying*" » dans « "*PROBLEM 2 : decryption doesn't work*" <http://ssldump.sourceforge.net/TROUBLESHOOTING> ».
— Si vous utilisez du RSA statique (et que vous êtes donc vulnérable aux attaquants qui mettraient la main après coup sur la clé privée), il y a une très bonne explication sur comment déchiffrer avec Wireshark <http://segfault.in/2010/11/decrypt-https-traffic-using-wireshark-and-key-fi
>.
— Si les détails du protocole TLS ne vous font pas peur, il existe un article très détaillé sur l'examen de traces TLS <http://max.euston.net/d/tip_ssldump.html>.
— Pour apprendre la PFS et le fonctionnement de DHE, un excellent article <http://vincent.bernat.im/fr/blog/2011-ssl-perfect-forward-secrecy.html> en français (très technique).
— Si on veut quand même examiner une session TLS utilisant du Diffie-Hellman éphémère, il existe quelques solutions qui marchent mais qui sont complexes à mettre en œuvre (et ne modifient donc pas la tonalité générale de cet article). Le principe est de faire une attaque de l'homme du milieu contre soi-même <http://security.stackexchange.com/questions/33374/whats-an-easy-way-to-perform-a-man-in-the-middle-attack-on-ssl>. On peut utiliser ZAP <https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project> ou Ncat <http://nmap.org/ncat/>. Cédric Trachsel suggère de mettre en place un serveur relais "*reverse proxy*" (par exemple un nginx), qu'on attaque en HTTP et qui relaie en HTTPS. Ainsi, le trafic entre le client et le serveur sera en HTTP et donc analysable. Même idée chez Kim-Minh Kaplan qui dirige vers Squid : "*Bumping CONNECT tunnels*" <http://wiki.squid-cache.org/Features/HTTPS#Bumping_CONNECT_tunnels> ou bien "*Bumping direct SSL/TLS connections*" <http://wiki.squid-cache.org/Features/HTTPS#Bumping_direct_SSL.2BAC8-TLS_connections>.

— Manuel Pégourié-Gonnard propose plutôt de convaincre la bibliothèque client TLS d'afficher le "*pre-master secret*", comme suggéré dans la documentation de Wireshark <http://wiki.wireshark.org/SSL> ou sur StackExchange <http://security.stackexchange.com/questions/35639/decrypting-tls-in-wireshark-when-using-dhe-rsa-ciphersuites> ou encore dans un excellent article pratique de Jim Shaver <https://jimshaver.net/2015/02/11/decrypting-tls-browser-traffic-with-wireshark-the-easy-way/>, qui explique comment lire cette clé depuis Wireshark. NSS sait faire (voir aussi « "*NSS Key Log Format*" <https://developer.mozilla.org/en-US/docs/NSS_Key_Log_Format> »). Avec OpenSSL, cela ne semble pas possible avec la commande `openssl s_client` mais peut-être l'est-ce avec la bibliothèque elle-même. Sinon, on peut tenter d'utiliser Panda <https://github.com/moyix/panda/blob/master/docs/panda_ssltut.md>.