La faille TLS de (non-)vidage des tampons

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 11 mars 2011

https://www.bortzmeyer.org/tls-vidage-tampon.html

Les failles de sécurité sont nombreuses sur l'Internet et CVE 2011-0411 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0411> n'est pas la plus grave. Mais elle est rigolote car elle ne résulte pas d'une faille du protocole (comme c'était le cas pour la renégociation TLS <https://www.bortzmeyer.org/tls-renego.html>) ni d'une bourde isolée présente uniquement dans un logiciel (comme la faille d'IOS dite « attribut 99 <https://www.bortzmeyer.org/bgp-attribut-99.html> »). La faille d'« injection de texte » de TLS ne met pas en cause ce protocole mais elle touche plusieurs mises en œuvre, car elle découle de suppositions qu'avaient faites plusieurs programmeurs indépendants.

D'abord, un avertissement, cet article est largement pompé de l'excellente synthèse de Wietse Venema, « "*Plaintext command injection in multiple implementations of STARTTLS*" <http://www.postfix.org/CVE-2011-0411.html> ». Si vous lisez l'anglais, vous pouvez laisser tomber mon article et voir celui de Venema. Sinon, continuons. La faille est liée à la façon dont plusieurs protocoles utilisent TLS. Ce protocole de cryptographie, normalisé dans le RFC 5246 [1], permet de protéger une communication, et assure la confidentialité et l'intégrité de celle-ci (et, en prime, si on croit en X.509, une certaine authentification). Tout va bien si on utilise TLS depuis le début, ce que fait la plupart du temps HTTP (si on a un URL de plan `https:`, la session sera protégée par TLS depuis le début, cf. RFC 2818). Mais beaucoup de protocoles utilisent une autre méthode, celle qui consiste à démarrer la session d'abord, puis à la monter en TLS, en général avec une commande nommée `STARTTLS`. C'est ainsi que fonctionnent SMTP (RFC 3207), IMAP (RFC 3501), ManageSieve (RFC 5804), etc. Ici, je me connecte à un serveur SMTP, je vérifie (réponse à la commande `EHLO`) qu'il gère TLS et je passe en TLS :

```
% telnet localhost smtp
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 horcrux ESMTP Postfix (Ubuntu)
EHLO test.example
```

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

```
250-horcrux
250-PIPELINING
250-SIZE 20000000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
STARTTLS
220 2.0.0 Ready to start TLS
```

Et, ensuite, je ne peux plus lui envoyer de commandes car je ne sais évidemment pas faire du TLS à la main. Pour déboguer des sessions TLS, on utilise en général `openssl`, inclus dans le paquetage du même nom :

```
% openssl s_client -quiet -starttls smtp -connect localhost:smtp
...
250 DSN
```

Et, à la fin, je peux taper des commandes dans une session protégée par TLS.

C'est là, dans cette promotion d'une session « normale » en session TLS qu'il y a un piège. Dès que le serveur reçoit `STARTTLS`, il commence la négociation TLS. Puis il continue à lire les données du client, et les traite désormais comme « protégées », comme « plus sûres ». Mais, pour cela, il faut être sûr qu'elles ont été envoyées **après** que la négociation TLS soit terminée avec succès. Or, plusieurs programmeurs (comme ceux du MTA Postfix, qui ont découvert la faille en auditant leur code) ont oublié ce détail, Leur code reprenait la lecture après la négociation TLS, en lisant des données qui avaient été envoyées **avant** et qui trainaient dans les tampons d'entrée-sortie.

On ne peut pas reproduire cette bogue avec telnet car la frappe est trop lente. Le temps de tenter l'injection de texte en clair, la négociation TLS est largement finie. Il faut donc modifier `openssl` La modification est simple : là où `openssl` envoie `STARTTLS\r\n` (les deux caractères \r et \n sont le Retour Chariot et le Saut de Ligne qui, ensemble, forment la fin d'une ligne dans la plupart des protocoles TCP/IP), on va envoyer `STARTTLS\r\nXXXXX\r\n` où XXXXX est une commande qui sera exécutée alors que le serveur se croit protégé, alors qu'elle avait été tapée avant. Ici, bien sûr, le client est le même, et n'a donc aucune raison de tricher, mais, dans le cas d'une vraie attaque, on peut imaginer un homme du milieu qui introduirait les commandes après `STARTTLS` mais avant que TLS soit réellement en route et n'empêche cette injection.

Voyons un exemple concret avec IMAP. Le protocole est un tout petit peu plus compliqué que SMTP car il faut préfixer chaque commande d'une étiquette, permettant d'associer une réponse à chaque question (RFC 3501, section 2.2.1). OpenSSL ne se fatigue pas, il utilise simplement un point comme étiquette. Dans la fonction `main` on trouve donc :

```
BIO_printf(sbio,". STARTTLS\r\n");
```

Comme commande à injecter, comme nous ne sommes pas de réels attaquants, nous allons simplement utiliser `NOOP` (RFC 3501, section 6.1.2) :

```
BIO_printf(sbio,". STARTTLS\r\n. NOOP\r\n");
```

Normalement, la commande `NOOP` ne doit **pas** être exécutée, puisque injectée en clair, après un `STARTTLS`. Testons avec notre OpenSSL modifié, sur un serveur Dovecot :

```
% ./apps/openssl s_client -quiet -starttls imap -connect imap.example.net:imap
...
. OK Capability completed.
```

Effectivement, rien n'indique que la commande ait été exécutée. C'est la même chose avec Zimbra. Mais, avec Courier :

```
% ./apps/openssl s_client -quiet -starttls imap -connect imap.example.net:imap
...
. OK CAPABILITY completed
. OK NOOP completed
```

Le `NOOP` a été exécuté. Le serveur Courier est donc vulnérable. Est-ce exploitable ? Sans doute. Par exemple, si le serveur authentifie le client sur la seule base d'un certificat fourni lors de la négociation TLS, un `. DELETE INBOX` ainsi injecté peut détruire la boîte `INBOX` de l'utilisateur... (Notez que l'attaque résumée ici n'est pas complète, je vous laisse trouver la faille.)

L'article de Venema, cité plus haut, explique très bien la cause de l'erreur dans le programme. Une leçon à en tirer pour les programmeurs ? Ne mélangez pas deux canaux d'entrée, un protégé et un qui ne l'est pas, ou, au moins, remettez à zéro le tampon d'entrée avec une promotion vers un protocole protégé.