RFC 7507 : TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 25 avril 2015

Date de publication du RFC : Avril 2015

https://www.bortzmeyer.org/7507.html

Le protocole TLS offre un grand nombre de choix : choix de la version, choix des algorithmes cryptographiques, plein d'options... Tous ces choix ne sont pas équivalents en terme de sécurité : les versions anciennes de TLS ont des vulnérabilités absentes des plus récentes, par exemple. Dès qu'un protocole cryptographique a des choix, il y a un risque de sécurité : que l'attaquant tente de perturber la négociation initiale entre les deux parties, pour les amener à choisir les variantes les moins sécurisées, en faisant croire à chacun que son partenaire ne peut pas gérer les meilleures variantes. C'est ce qu'on nomme une **attaque par repli** ("*downgrade attack*") et c'est une plaie classique des protocoles cryptographiques (sauf de ceux qui n'offrent aucun choix, mais qui ont d'autres problèmes, notamment l'impossibilité d'évoluer). Ce RFC présentait une technique TLS pour empêcher certaines attaques par repli : annoncer un algorithme cryptographique qui n'est pas un vrai algorithme mais qui signale qu'on a effectué un repli, permettant au partenaire de détecter que quelqu'un a interféré avec la communication. Cette technique se nomme SCSV, pour "*Signaling Cipher Suite Value*". Elle a été abandonnée par le RFC 8996 [1], qui retirait du service les versions de TLS vulnérables.

Normalement, un tel signalement n'est pas nécessaire (c'est très bien expliqué dans cet article sur StackExchange <http://crypto.stackexchange.com/a/19674/11528>), les règles de négociation de TLS se chargent de tout (RFC 5246, sections 7.3 et 7.4). Mais il y a des serveurs TLS bogués qui réagissent mal avec certaines variantes. Au lieu d'indiquer proprement qu'ils ne gèrent pas ces variantes, ils coupent la communication. Un client TLS qui veut donc maximiser ses chances de se connecter va donc réessayer, en ne proposant pas les choix qui ont pu mener au précédent échec. C'est sympa du point de vue de l'interopérabilité mais cette "*downgrade dance*" <https://www.openssl.org/~bodo/ssl-poodle.pdf> est dangereuse du point de vue de la sécurité : le serveur, lors de la deuxième tentative, ne sait pas qu'il s'agit d'un repli. Prenons l'exemple d'un attaquant qui a trouvé une faille dans

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc8996.txt

TLS 1.0 mais qui n'existe plus dans TLS 1.2 (celui du RFC 5246). Si Alice et Bob savent tous les deux parler 1.2, c'est cette version qu'ils choisiront, par défaut. Il faut donc les convaincre de se replier en 1.0. Un attaquant actif peut perturber la négotiation TLS (par exemple en ne transmettant pas le `ServerHello`, donnant au client l'impression que le serveur a ignoré son `ClientHello`). Certains clients vont alors se dire « zut, demander 1.2 perturbe le serveur, je vais réessayer directement en 1.0 ». Et l'attaquant aura atteint son but. Le serveur, qui sait faire du TLS 1.2, verra une demande en 1.0 et se dire juste « tiens, un vieux client qui ne sait pas faire de 1.2 ». Jusqu'à ce nouveau RFC, la seule protection contre ces attaques était de configurer le serveur pour refuser certains choix vraiment trop mauvais, question sécurité. Cette solution était trop rigide (pour reprendre l'exemple précédent, TLS 1.0 n'est pas catastrophique : il est raisonnable d'autoriser les vieux clients à se connecter en 1.0, mais on veut empêcher que les clients récents ne soient amenés à se limiter à 1.0). Au passage, un exemple d'une telle démarche est discutée dans l'article présentant la configuration TLS de mon blog <`https://www.bortzmeyer.org/https-blog.html`> (la ligne `GnuTLSPriorities SECURE:-VERS-SSL3.0:-ARCFOUR-128:-ARCFOUR-40`).

L'idéal serait bien sûr qu'il n'y ait pas de serveurs bogués, que la négociation TLS normale suffise, et que les clients n'aient jamais à faire de "*downgrade dance*". Mais, dans le monde cruel où nous vivons, cela n'arrivera pas, les programmeurs ne lisent pas les normes et ne testent pas leur travail. À noter que, même en l'absence d'un attaquant actif, on voit parfois des replis malencontreux : un problème temporaire de réseau peut mener le client TLS à se dire « tiens, pas de réponse, je vais réessayer en TLS 1.0 et sans les extensions, pour voir ».

Donc, pour permettre au serveur de détecter que le client a fait un repli à tort, notre RFC introduit un nouveau pseudo-algorithme de chiffrement, `TLS_FALLBACK_SCSV`, valeur 0x56,0x00 (désormais dans le registre IANA <`https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4`>). Il sera envoyé uniquement par le client, jamais par le serveur. Ce n'est pas un vrai algorithme, juste un signalement par le client qu'une première connexion a échoué et que le client tente un repli (section 4 du RFC). Si le client envoie le pseudo-algorithme `TLS_FALLBACK_SCSV` dans son `ClientHello`, et indique un protocole de version inférieure à ce que le serveur peut gérer, le serveur peut alors se rendre compte que le client a effectué un repli à tort (cf. section 3). Il peut alors répondre avec un nouveau message d'erreur, l'alerte TLS `inappropriate_fallback` (valeur 86 <`https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-6`>). Voici ce que ça donne avec Firefox :

Le signalement par le client qu'il s'est replié est un (pseudo-)algorithme de chiffrement, pas une extension TLS, car le but est qu'il ne pose de problème avec aucun serveur, même le plus ancien et le plus bogué. Le mécanisme d'indication des algorithmes de chiffrement date des débuts de SSL et on peut donc certainement compter dessus. Au passage, la technique de notre nouveau RFC est la deuxième utilisation de ce principe du SCSV, après la `TLS_EMPTY_RENEGOTIATION_INFO_SCSV` de la section 3.3 du RFC 5746.

La section 6 de notre RFC explore les cas où le client a intérêt à envoyer un `TLS_FALLBACK_SCSV` ou pas. Si le client n'a pas pu se connecter en TLS v1 ou supérieur, et se replie sur SSL v3, alors, le `TLS_FALLBACK_SCSV` est vraiment nécessaire car SSL v3 a de nombreuses faiblesses (la faille Poodle par exemple). Par contre, quand un client a une toute nouvelle version de TLS (à l'heure de la publication de ce RFC, la version 1.3 est en cours de développement), des difficultés avec les vieux serveurs sont à prévoir et il peut être préférable de ne pas envoyer `TLS_FALLBACK_SCSV`, et d'accepter le repli.

Des mesures faites en novembre 2014 indiquaient que 14 % des serveurs TLS sur l'Internet géraient déjà ce pseudo-algorithme `TLS_FALLBACK_SCSV`. Si vous voulez tester vous-même, vous pouvez par exemple vous servir d'OpenSSL en ligne de commande :

```
% openssl s_client -connect google.com:443 -state -fallback_scsv -tls1_1
CONNECTED(00000003)
SSL_connect:before/connect initialization
SSL_connect:SSLv3 write client hello A
SSL3 alert read:fatal:unknown
SSL_connect:failed in SSLv3 read server hello A
1077786776:error:1409443E:SSL routines:SSL3_READ_BYTES:tlsv1 alert inappropriate fallback:s3_pkt.c:1260:SSL aler
1077786776:error:1409E0E5:SSL routines:SSL3_WRITE_BYTES:ssl handshake failure:s3_pkt.c:598:
```

Ici, on a tenté de se connecter à Google en envoyant le SCSV (option `-fallback_scsv`) et en indiquant comme numéro de version TLS 1.1 (alors que Google et OpenSSL savent tous les deux faire du 1.2). Google a donc envoyé l'alarme (le message "*alert inappropriate fallback*"). Avec un serveur qui ne gère pas SCSV, on aurait au contraire eu une connexion sans alerte. Si, au lieu du programme `openssl`, on veut le faire depuis une application qu'on écrit, il y a instructions et code sur le Wiki d'OpenSSL. <`https://wiki.openssl.org/index.php/SSL_MODE_SEND_FALLBACK_SCSV`> Notez que le test des SSL Labs <`https://www.ssllabs.com/`> regarde aussi SCSV (message « "*This server supports TLS_FALLBACK_SCSV to prevent protocol downgrade attacks*" » ou bien aucune indication dans le cas contraire). Vu avec tshark, le test avec `openssl` montrera :

```
client -> serveur:

        Handshake Protocol: Client Hello
            Version: TLS 1.1 (0x0302)
                Cipher Suite: Unknown (0x5600) [version trop ancienne
de Wireshark, qui n'affiche pas le nom de la SCSV]

serveur -> client:

    TLSv1.1 Record Layer: Encrypted Alert
        Content Type: Alert (21)
        Version: TLS 1.1 (0x0302)
        Length: 2
        Alert Message: Encrypted Alert
```

(L'alerte est chiffrée et donc pas décodée par défaut par Wireshark.)

Côté navigateurs Web, Firefox et Chrome gèrent cette technique SCSV mais il semble qu'Internet Explorer ne le fera pas <`https://connect.microsoft.com/IE/feedback/details/1002874/internet-explorer-should-send-tls-fallback-scsv`>.

Et, pour finir, une bonne explication de SCSV, avec des exemples Wireshark <`https://alpacapowered.wordpress.com/2014/10/20/ssl-poodle-attack-what-is-this-scsv-thingy/`>. Et merci à Florian Maury pour sa relecture.