RFC 5746 : Transport Layer Security (TLS) Renegotiation Indication Extension

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 13 février 2010

Date de publication du RFC : Février 2010

http://www.bortzmeyer.org/5746.html

Les failles de sécurité font partie du folklore de l'Internet mais la plupart touchent une mise en œuvre particulière, qu'il suffit de corriger par un "*patch*". La faille de renégociation <http://www.bortzmeyer.org/tls-renego.html> de TLS, révélée en novembre 2009 était au contraire une faille d'un protocole. Il fallait donc modifier celui-ci pour la corriger, ce que vient de faire notre RFC (dont l'un des auteurs, Marsh Ray, est le découvreur de la faille). Ce RFC a été écrit et publié de manière particulièrement rapide, compte-tenu de l'urgence de combler cette faille. (Cette rapidité, inhabituelle à l'IETF, a d'ailleurs suscité des critiques <http://www.ietf.org/mail-archive/web/ietf/current/msg64445.html>.)

Un petit rappel : le protocole de sécurité TLS, normalisé dans le RFC 5246[1], contient une phase de **négociation** pendant laquelle les deux systèmes peuvent décider de choses comme les algorithmes de cryptographie à utiliser. Ils peuvent aussi présenter un certificat X.509 pour s'authentifier. Un point important de TLS est que la négociation ne survient pas uniquement au début de la session. À tout moment, chacune des deux parties peut **renégocier** (commande `R` si on utilise `openssl s_client`). Mais, et c'est le cœur de la faille, il n'existe pas de **liaison** ("*binding*") entre l'état de la session avant renégociation et après. Un attaquant peut donc commencer une session avec un serveur, envoyer des données, lancer une renégociation, puis laisser la place au vrai client qui va s'authentifier. (Ce problème, et l'attaque qu'il rend possible, sont détaillés dans la section 1 du RFC.) Avec des protocoles utilisant TLS, comme HTTP qui ne fait pas la différence entre avant et après la renégociation, les données envoyées par l'attaquant seront considérées comme authentifiées par le vrai client... Même avec des protocoles comme SMTP et IMAP, où la transition entre état authenfifié et non authentifié est davantage marquée, certaines attaques restaient possibles. Bref, l'absence de toute liaison cryptographique fiable entre l'≪ ancienne ≫ session et la ≪ nouvelle ≫ fait que le serveur et le client n'avaient aucun moyen de détecter l'attaque.

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

Qu'est-ce qu'une liaison, à ce sujet ? C'est une preuve cryptographique que deux éléments sont liés, qu'une attaque contre l'un invaliderait l 'autre. C'est un composant essentiel de tout protocole de cryptographie, dont l'oubli ou les défaillances sont à l'origine de plusieurs failles de sécurité. Un exemple de liaison est celle que fait SSH avec le MAC qui est calculé pour les paquets (RFC 4253, section 6.4). Sans ce MAC, l'authentification effectuée au début d'une connexion SSH ne servirait pas à grand'chose puisqu'il n'y aurait pas de liaison entre cette authentification et les paquets : un méchant pourrait laisser l'authentification se faire, puis injecter ses propres paquets. (Sur la question de la liaison cryptographique, on peut lire le RFC 5056 pour plus de détails.)

Bref, TLS manquait d'une liaison entre la session avant et après renégociation. Comment résoudre cette faille ? Le plus simple était de supprimer complètement la renégociation dans la serveur, et c'est ce qu'a fait OpenSSL en urgence. Mais cela ne résoud pas le problème des utilisateurs qui ont besoin de la rénégociation (par exemple pour l'authentification X.509 sur HTTP, si on ne veut pas avoir deux machines, une pour les sessions authentifiées et une pour les autres).

Notre RFC 5746 crée donc une autre solution, qui va nécessiter la mise à jour des mises en œuvre de TLS. Son principe est d'avoir une extension TLS `renegotiation_info` qui lie cryptographiquement l'ancienne session (avant la rénégociation) et la nouvelle. Elle est décrite plus en détail en section 3. Elle nécessite que chaque pair TLS garde quelques informations supplémentaires :

— Un booléen `secure_renegotiation` qui indique si la nouvelle option est utilisable sur cette connexion TLS,
— Une valeur `client_verify_data` qui indiquait les données de vérification envoyées par le client à la dernière négociation (et que le client doit donc connaître, pour authentifier la renégociation),
— Une valeur `server_verify_data`, l'équivalent pour le serveur.

L'extension elle-même figure en section 3.2. `renegotiation_info` est définie comme :

```
struct {
    opaque renegotiated_connection<0..255>;
} RenegotiationInfo;
```

où la valeur `renegotiated_connection` vaut zéro au début puis, en cas de renégociation, vaut `*_verify_data`.

Le changement à TLS est donc trivial. Mais il y a quelques détails pénibles. Par exemple, la section 3.3 fait remarquer que, bien qu'une extension inconnue doive être ignorée, certaines mises en œuvre de TLS avortent la connexion dans ce cas. Face à un TLS récent qui tente de leur envoyer une `renegotiation_info`,il y aura donc un problème. Une astuce a donc été créé : comme TLS permet d'indiquer les algorithmes de chiffrement acceptés, un algorithme bidon a été normalisé, `TLS_RENEGO_PROTECTION_REQUEST`. L'envoyer dans la liste des algorithmes ne plantera personne (les algorithmes inconnus sont légion et ne perturbent aucune implémentation) et permettra d'indiquer qu'on gère la nouvelle extension. (Oui, c'est un bricolage mais, dans le monde réel, on doit souvent utiliser de tels bricolages pour assurer un déploiement réussi, malgré la présence de logiciels bogués.)

Après ce petit détour sur la compatibilité, le RFC décrit en détail le comportement que doivent adopter clients et serveurs, aussi bien pour la connexion initiale (sections 3.4 et 3.6) que pour une éventuelle renégotiation (sections 3.5 et 3.7). Le client doit envoyer une extension `renegotiation_info` vide au début, pour indiquer sa volonté de protéger la rénégotiation (ou bien utiliser le truc de l'algorithme de chiffrement bidon). Si cette extension n'est pas dans la réponse, cela peut signifier que le serveur utilise un ancien logiciel, antérieur à notre RFC, ou bien qu'une attaque de l'homme du milieu est en cours. Le client doit donc décider s'il continue en notant juste que `secure_renegotiation` est faux, ou bien il doit être exigeant et couper la session (après tout, autrement, la nouvelle extension de sécurisation de la

renégociation pourrait être rendue inutile par des attaques par repli). Mais la décision est difficile car, au début tout au moins, les clients TLS nouveaux n'auront guère de serveurs « sécurisés » à qui parler. La section 4.1 décrit plus en détail ce problème cornélien.

La situation est plus simple pour le serveur qui peut toujours, s'il rencontre un vieux client, refuser une éventuelle renégociation. La méthode recommandée en section 4.3 (et 4.4 et 5) est donc de n'accepter de renégociation qu'avec un client compatible avec ce RFC 5746.

La renégociation, même sécurisée par une liaison cryptographique avec l'état antérieur de la session, soulève d'amusants problèmes de sécurité, que traite la section 5. Ainsi, beaucoup de logiciels croient qu'en appelant `getPeerCertificates()`, ils obtiennent une liste de certificats immuable, qui est valide tout le temps de la session. Mais ce n'est pas le cas s'il y a renégociation.

Notre RFC recommande donc aux bibliothèques TLS de fournir un moyen d'autoriser ou d'interdire la renégociation (ce n'était pas le cas d'OpenSSL avant la découverte de la faille de sécurité.) Il y a d'autres recommandations comme celle d'offrir une possibilité de permettre la rénégociation mais sans changement des certificats (d'autant plus que, avec la plupart des bibliothèques existantes, il n'existe aucun moyen de savoir quelles données ont été transmises avant et après la renégociation).

Aujourd'hui, parmi les deux mises en œuvre de TLS en logiciel libre, OpenSSL a intégré le code pour notre RFC 5746 dans sa version 0.9.8m alors que GnuTLS l'a en développement (l'essentiel du code est dans `lib/ext_safe_renegotiation.c` et est écrit par un des auteurs du RFC).