RFC 8308 : Extension Negotiation in the Secure Shell (SSH) Protocol

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 21 mars 2018. Dernière mise à jour le 28 mars 2018

Date de publication du RFC : Mars 2018

https://www.bortzmeyer.org/8308.html

Le protocole SSH n'avait pas de mécanisme propre pour négocier des extensions au protocole de base (comme celle du RFC 4335 [1]). En outre, une partie de la négociation, par exemple des algorithmes cryptographiques, se déroulait **avant** le début du chiffrement et n'était donc pas protégée. Ce nouveau RFC ajoute à SSH un mécanisme pour négocier les extensions après l'échange de clés et le début de la communication sécurisée, une fois que tout est confidentiel.

Un mécanisme de négociation pour un protocole cryptographique est toujours délicat (comme TLS s'en est aperçu à plusieurs reprises). Si on n'en a pas, le client et le serveur perdent beaucoup de temps à faire des essais/erreurs « tiens, il vient de déconnecter brusquement, c'est peut-être qu'il n'aime pas SHA-512, réessayons avec SHA-1 ». Et ces deux mécanismes (négociation explicite et essai/erreur) ouvrent une nouvelle voie d'attaque, celle des attaques par repli, où l'Homme du Milieu va essayer de forcer les deux parties à utiliser des algorithmes de qualité inférieure. (La seule protection serait de ne pas discuter, de choisir des algorithmes forts et de refuser tout repli. En sécurité, ça peut aider d'être obtus.)

Notez aussi que la méthode essais/erreurs a un danger spécifique, car bien des machines SSH mettent en œuvre des mécanismes de limitation du trafic, voire de mise en liste noire, si la machine en face fait trop d'essais, ceci afin d'empêcher des attaques par force brute. D'où l'intérêt d'un vrai mécanisme de négociation (qui peut en outre permettre de détecter certaines manipulations par l'Homme du Milieu).

Vue par tshark, voici le début d'une négociation SSH, le message `SSH_MSG_KEXINIT`, normalisé dans le RFC 4253, section 7.1 :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc4335.txt`

```
SSH Protocol
    SSH Version 2
        Packet Length: 1332
        Padding Length: 5
        Key Exchange
            Message Code: Key Exchange Init (20)
            Algorithms
                Cookie: 139228cb5cfbf6c97d6b74f6ae99453d
                kex_algorithms length: 196
                kex_algorithms string: curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ec
                server_host_key_algorithms length: 290
                server_host_key_algorithms string [truncated]: ecdsa-sha2-nistp256-cert-v01@openssh.com,ecds
                encryption_algorithms_client_to_server length: 150
                encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-ctr,aes1
                encryption_algorithms_server_to_client length: 150
                encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-ctr,aes1
                mac_algorithms_client_to_server length: 213
                mac_algorithms_client_to_server string [truncated]: umac-64-etm@openssh.com,umac-128-etm@ope
                mac_algorithms_server_to_client length: 213
                mac_algorithms_server_to_client string [truncated]: umac-64-etm@openssh.com,umac-128-etm@ope
                compression_algorithms_client_to_server length: 26
                compression_algorithms_client_to_server string: none,zlib@openssh.com,zlib
                compression_algorithms_server_to_client length: 26
                compression_algorithms_server_to_client string: none,zlib@openssh.com,zlib
                languages_client_to_server length: 0
                languages_client_to_server string: [Empty]
                languages_server_to_client length: 0
                languages_server_to_client string: [Empty]
                KEX First Packet Follows: 0
                Reserved: 00000000
            Padding String: 0000000000
```

Le message était en clair (c'est pour cela que tshark a pu le décoder). L'autre machine envoie un message équivalent. Suite à cet échange, les deux machines sauront quels algorithmes sont disponibles.

Notre RFC 8308 (section 2.1) ajoute à ce message le nouveau mécanisme d'extension. Pour préserver la compatibilité avec les anciennes mises en œuvre de SSH, et comme il n'y a pas de place « propre » disponible dans le message, le nouveau mécanisme se place dans la liste des algorithmes cryptographiques (champ `kex_algorithms`, celui qui commence par `curve25519-sha256@libssh.org,ecdh-sha2-nistp256`. On ajoute à cette liste `ext-info-c` si on est client et `ext-info-s` si on est serveur. (Le nom est différent dans chaque direction, pour éviter que les deux parties ne se disent « cool, cet algorithme est commun, utilisons-le ».) Vous pouvez donc savoir si votre SSH gère le nouveau mécanisme en capturant ce premier paquet SSH et en examinant la liste des algorithmes d'échange de clé. (C'était le cas ici, avec OpenSSH 7.2. Vous l'aviez repéré ?)

Une fois qu'on a proposé à son pair d'utiliser le nouveau mécanisme de ce RFC 8308, on peut s'attendre, si le pair est d'accord, à recevoir un message SSH d'un nouveau type, `SSH_MSG_EXT_INFO`, qui sera, lui, chiffré. Il contient la liste des extensions gérées avec ce mécanisme. Notez qu'il peut être envoyé plusieurs fois, par exemple avant et après l'authentification du client, pour le cas d'un serveur timide qui ne voulait pas révéler sa liste d'extensions avant l'authentification. Ce nouveau type de message figure désormais dans le registre des types de message <`https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xml#ssh-parameters-1`>, avec le numéro 7.

La section 3 définit les quatre extensions actuelles (un registre accueillera <`https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xml#extension-names`> les extensions futures) :

— L'extension `server-sig-algs` donne la liste des algorithmes de signature acceptés par le serveur.

— `delay-compression` indique les algorithmes de compression acceptés. Il y en a deux, un du client vers le serveur et un en sens inverse. Ils étaient déjà indiqués dans le message `SSH_MSG_KEXINIT`, dans les champs `compression_algorithms_client_to_server` et `compression_algorithms_server_to_client` mais, cette fois, ils sont transmis dans un canal sécurisé (confidentiel et authentifié).
— `no-flow-control`, dont le nom indique bien la fonction.
— `elevation` sert aux systèmes d'exploitation qui ont un mécanisme d'élévation des privilèges (c'est le cas de Windows, comme documenté dans le blog de Microsoft <https://blogs.msdn.microsoft.com/winsdk/2013/03/22/how-to-launch-a-process-as-a-full-administrator>).

Les futures extensions, après ces quatre-là, nécessiteront un examen par l'IETF, via un RFC IETF (cf. RFC 8126, politique « "*IETF review*" »). Elle seront placées dans le registre des extensions <https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xml#extension-names>.

Quelques petits problèmes de déploiement et d'incompatibilité ont été notés avec ce nouveau mécanisme d'extensions. Par exemple OpenSSH 7.3 et 7.4 gérait l'extension `server-sig-algs` mais n'envoyait pas la liste complète des algorithmes acceptés. Un client qui considérait cette liste comme ferme et définitive pouvait donc renoncer à utiliser certains algorithmes qui auraient pourtant marché. Voir ce message <https://www.ietf.org/mail-archive/web/curdle/current/msg00982.html> pour une explication détaillée.

Autre gag, les valeurs des extensions peuvent contenir des octets nuls et un logiciel maladroit qui les lirait comme si c'était des chaînes de caractères en C aurait des problèmes. C'est justement ce qui est arrivé à OpenSSH, jusqu'à la version 7.5 incluse, ce qui cassait brutalement la connexion. Le RFC conseille de tester la version du pair, et de ne pas utiliser les extensions en cause si on parle à un OpenSSH [Caractère Unicode non montré [2]] 7.5.

Aujourd'hui, ce mécanisme d'extension est mis en œuvre dans OpenSSH, Bitvise SSH <https://www.bitvise.com/>, AsyncSSH <https://github.com/ronf/asyncssh> et SmartFTP. (Cf. ce tableau de comparaison <http://ssh-comparison.quendi.de/comparison/hostkey.html>, mais qui n'est pas forcément à jour.)

Merci à Manuel Pégourié-Gonnard pour avoir détecté une erreur dans la première version.

2. Car trop difficile à faire afficher par LaTeX