RFC 8701 : Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 29 janvier 2020

Date de publication du RFC : Janvier 2020

https://www.bortzmeyer.org/8701.html

Ce nouveau RFC s'attaque à un problème fréquent dans l'Internet : des programmeurs paresseux, incompétents ou pressés par les délais imposés mettent en œuvre un protocole normalisé (comme TLS) sans bien lire les normes, et notamment sans tenir compter des variations que la norme permet. Ils programment rapidement, testent avec une ou deux implémentations trouvées et, si ça marche, en déduisent que c'est bon. Mais dès qu'une autre implémentation introduit des variantes, par exemple un paramètre optionnel et qui n'était pas utilisé avant, la bogue se révèle. Le cas s'est produit de nombreuses fois. Notre RFC propose une solution disruptive : utiliser délibérément, et au hasard, plein de variantes d'un protocole, de façon à détecter rapidement les programmes écrits avec les pieds. C'est le principe de GREASE ("*Generate Random Extensions And Sustain Extensibility*"), la graisse qu'on va mettre dans les rouages de l'Internet pour que ça glisse mieux. Ce RFC 8701 [1] commence par appliquer ce principe à TLS.

Le problème n'est évidemment pas spécifique à TLS, on l'a vu arriver aussi dans BGP lorsqu'on s'est aperçu que la simple annonce d'un attribut BGP inconnu <https://www.bortzmeyer.org/bgp-attribut-99.html> pouvait planter les routeurs Cisco. Là aussi, le « graissage » (tester systématiquement des valeurs non allouées pour les différents paramètres, pour vérifier que cela ne plante rien) aurait bien aidé. D'où le projet « "*Use it or lose it*" », décrit dans l'"*Internet-Draft*" draft-iab-use-it-or-lose-it dont GREASE est un cas particulier. Le "*draft*" draft-iab-use-it-or-lose-it analyse le problème des options non utilisées et recommande de les utiliser systématiquement, pour habituer les logiciels à voir ces options.

Le principe de GREASE ("*Generate Random Extensions And Sustain Extensibility*") est donc de faire en sorte que clients et serveurs TLS (RFC 8446) annoncent, pour différents paramètres de la connexion,

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc8701.txt

des valeurs qui ne correspondent à rien. Ainsi, les "*middleboxes*" boguées, installées au milieu de la communication parce que le commercial qui les vendait était convaincant, seront vite détectées, au lieu que le problème demeure dormant pendant des années et soit subitement révélé le jour où on essaie des valeurs légales mais nouvelles, comme dans le cas de l'attribut 99 <https://www.bortzmeyer.org/bgp-attribut-99.html>.

Qu'est-ce qui est variable dans TLS ? Beaucoup de choses, comme la liste des algorithmes de cryptographie ou comme les extensions. Dans TLS, le client annonce ce qu'il sait faire, et le serveur sélectionne dans ce choix (RFC 8446, section 4.1.3). Voici un exemple vu par tshark, d'abord le message du client (`Client Hello`), puis la réponse du serveur (`Server Hello`) :

```
Secure Sockets Layer
    TLSv1 Record Layer: Handshake Protocol: Client Hello
        Content Type: Handshake (22)
        Handshake Protocol: Client Hello
            Version: TLS 1.2 (0x0303)
            Cipher Suites (28 suites)
                Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
                Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
                ...
            Extension: ec_point_formats (len=4)
                Type: ec_point_formats (11)
                EC point formats Length: 3
                Elliptic curves point formats (3)
                    EC point format: uncompressed (0)
                    ...
            Extension: SessionTicket TLS (len=0)
                Type: SessionTicket TLS (35)
            Extension: encrypt_then_mac (len=0)
                Type: encrypt_then_mac (22)
            Extension: signature_algorithms (len=48)
                Type: signature_algorithms (13)
                Signature Hash Algorithms (23 algorithms)
                    Signature Algorithm: ecdsa_secp256r1_sha256 (0x0403)
                        Signature Hash Algorithm Hash: SHA256 (4)
                        Signature Hash Algorithm Signature: ECDSA (3)
                        ...


Secure Sockets Layer
    TLSv1.2 Record Layer: Handshake Protocol: Server Hello
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Handshake Protocol: Server Hello
            Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
            Compression Method: null (0)
            Extensions Length: 13
            Extension: renegotiation_info (len=1)
                Type: renegotiation_info (65281)
                Length: 1
                Renegotiation Info extension
                    Renegotiation info extension length: 0
            Extension: ec_point_formats (len=4)
                Type: ec_point_formats (11)
            ...
```

Le client propose de nombreux algorithmes de cryptographie, comme `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`, et plusieurs extensions comme le format pour les courbes elliptiques (normalisé dans le RFC 8422), les tickets du RFC 5077, le chiffrement avant le MAC du RFC 7366 et des algorithmes de signature. Le serveur choisit l'algorithme de chiffrement `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`, accepte l'extension sur le format des courbes elliptiques, et, puisque le client

était d'accord (via l'indication d'un algorithme de chiffrement spécial), le serveur utilise l'extension de renégociation.

Les valeurs inconnues (par exemple une nouvelle extension) **doivent** être ignorées (RFC 8446, section 4.1.2). Si ce n'était pas le cas, si une valeur inconnue plantait la partie située en face, il ne serait pas possible d'introduire de nouveaux algorithmes ou de nouvelles extensions, en raison des déploiements existants. Prenons les algorithmes de cryptographie, enregistrés à l'IANA <https://www.iana.org/assignments/tls-parameters/tls-parameters.xml#tls-parameters-4>. Si un nouvel algorithme apparait, et reçoit une valeur, comment vont réagir, non seulement les pairs avec qui on communique en TLS, mais également tous ces boitiers intermédiaires installés souvent sans raison sérieuse, et pour lesquels il n'existe pas de mécanisme pratique de remontée des problèmes ? Si leurs programmeurs avaient lu la norme, ils devraient ignorer ce nouvel algorithme, mais on constate qu'en pratique, ce n'est souvent pas le cas, ce qui rend difficile l'introduction de nouveaux algorithmes. Dans le pire des cas, le boitier intermédiaire jette les paquets portant les valeurs inconnues, sans aucun message, rendant le déboguage très difficile.

D'où les métaphores mécaniques : dans l'Internet d'aujourd'hui, bien des équipements sur le réseau sont rouillés, et il faut les graisser, en faisant travailler les parties qui ne sont normalement pas testées. C'est le principe de GREASE que d'envoyer des valeurs inconnues pour certains paramètres, dans l'espoir de forcer les mises en œuvre de TLS, surtout celles dans les boitiers intermédiaires, à s'adapter. Une méthode darwinienne, en somme.

La section 2 de notre RFC indique les valeurs choisies pour ces annonces. C'est délibérement qu'elles ne sont pas contiguës, pour limiter le risque que des programmeurs paresseux ne testent simplement si une valeur est incluse dans tel intervalle. Il y a un jeu de valeurs pour les algorithmes de cryptographie et les identificateurs ALPN (RFC 7301), un pour les extensions, un pour les versions de TLS, etc. Toutes sont enregistrées à l'IANA, dans le registre respectif. Par exemple, pour les extensions TLS, (cf. leur liste <https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xml#tls-extensiontype-values-1>), les valeurs, 2570, 6682, 10794 et plusieurs autres sont réservées pour le graissage. (Il fallait les réserver pour éviter qu'une future extension TLS ne reçoive le même numéro, ce qui aurait cassé la compatibilité avec les logiciels GREASE.)

Une fois ces valeurs réservées par notre RFC, le client TLS peut, au hasard, ajouter ces valeurs dans, par exemple, la liste des algorithmes de cryptographie qu'il gère, ou la liste des extensions qu'il propose. Si jamais le serveur les accepte (dans son `ServerHello`), le client doit rejeter la connexion ; le but de ces valeurs était de tester les logiciels, elles ne doivent jamais être sélectionnées. Notez que c'est le comportement normal d'un client TLS d'aujourd'hui de refuser proprement les valeurs inconnues. De même, un serveur normal d'aujourd'hui va ignorer ces valeurs inconnues (et donc ne jamais les sélectionner). Si tout le monde suit la norme, l'introduction des valeurs GREASE ne va rien changer. Les règles de la section 3 ne sont qu'un rappel de règles TLS qui ont toujours existé.

La section 4 de notre RFC traite le cas un peu plus difficile où le serveur propose et le client accepte. C'est par exemple ce qui arrive quand le serveur demande au client de s'authentifier, en envoyant un `CertificateRequest`. Le serveur peut là aussi utiliser GREASE et indiquer des extensions ou des algorithmes de signature inconnus, et le client doit les ignorer (sinon, si le client sélectionne ces valeurs, le serveur doit rejeter un tel choix).

La section 5 du RFC précise dans quels cas utiliser les valeurs GREASE et lesquelles. Le but étant de révéler les problèmes, le RFC recommande de choisir les valeurs aléatoirement. (Si un programme envoyait toujours la même valeur GREASE, il y aurait un risque que des programmes en face ignorent spécifiquement cette valeur, alors qu'il faut ignorer toutes les valeurs inconnues.) Par contre, pour un

même partenaire TLS, il vaut mieux un certain déterminisme, sinon les problèmes seront difficiles à déboguer (parfois, ça marche, parfois, ça ne marche pas…)

Enfin, la section 7 du RFC discute des conséquences de l'absence de graissage sur la sécurité. Si certaines mises en œuvre de TLS résistent mal aux options inconnues, cela peut encourager le repli, c'est-à-dire à réessayer sans les options. Ainsi, un attaquant actif pourrait facilement forcer une machine à ne pas utiliser des options qui rendraient des attaques ultérieures plus compliquées. Les attaques par repli étant particulièrement dangereuses, il est important de ne pas se replier, et donc de s'assurer qu'on peut réellement utiliser toutes les possibilités du protocole. Cela veut dire entre autres que, si une machine TLS utilisant GREASE a du mal à se connecter, elle ne devrait pas essayer sans GREASE : cela annnulerait tous les bénéfices qu'on attend du graissage. Le principe de robustesse <https://www.bortzmeyer.org/principe-robustesse.html> est mauvais pour la sécurité.

À noter que Chrome a déjà mis en œuvre ce principe, et que ça semble bien fonctionner. L'article « HTTPS : de SSL à TLS 1.3 <https://openweb.eu.org/articles/https-de-ssl-a-tls-1-3> », surtout consacré à la version 1.3 de TLS, montre à quoi ressemble les options GREASE pour Wireshark (« "*Unknown*" »).

À noter qu'un concept équivalent existe dans le futur HTTP/3, "*Reserved Stream Types*" et "*Reserved Frame Types*". Pour HTTP/2 (RFC 7540), où les trames de type inconnu devraient être ignorées, l'expérience a déjà prouvé qu'elles ne l'étaient pas toujours <https://bugs.chromium.org/p/chromium/issues/detail?id=1019410>.