RFC 6528 : Defending Against Sequence Number Attacks

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 2 février 2012

Date de publication du RFC : Février 2012

https://www.bortzmeyer.org/6528.html

Ce court RFC spécifie les précautions que doit prendre une mise en œuvre de TCP pour éviter qu'un attaquant ne puisse deviner le numéro de séquence initial. Rien de nouveau, les précautions en question étant programmées dans TCP depuis de nombreuses années, juste un rappel et une mise à jour du précédent RFC sur ce sujet, le RFC 1948 [1]. Il a depuis été intégré dans le RFC 9293.

Pour comprendre le problème de sécurité des numéros de séquence, et les solutions présentées dans ce RFC, il vaut mieux revenir à l'attaque qui exploitait les anciennes vulnérabilités. C'est fait dans l'annexe A du RFC, que je reprends ici.

Le but de l'attaquant est de s'en prendre à un serveur qui autorise par adresse IP. À l'époque du RFC 1948, cela concernait rsh mais, de nos jours, SSH a complètement remplacé cet archaïque protocole. Aujourd'hui, ce serait plutôt un serveur DNS qui authentifie le transfert de zones ainsi, ou bien un serveur SMTP qui autorise seulement certaines adresses IP à passer par lui - même s'il ferait mieux de se servir du RFC 4954, ou encore un serveur HTTP qui ne permet l'invocation de certaines actions qu'aux utilisateurs de certaines adresses. En deux mots, dès que l'autorisation se fait par adresse IP uniquement, il faut s'assurer que vous utilisez un TCP qui mette en œuvre les recommandations de ce RFC 6528.

On le sait, il est trivial pour un attaquant d'envoyer des paquets avec une fausse adresse IP source (le RFC 2827, censé l'empêcher, n'étant quasiment pas déployé). Mais, dans ce cas, l'attaquant ne reçoit pas les réponses. Or, si on veut maintenir une connexion TCP avec sa victime, il faut lui envoyer des paquets avec les données qu'elle attend, notamment le numéro de port et les numéros de séquence. Sinon, la victime va jeter ces paquets TCP qui lui sembleront anormaux. Pour les numéros de port, le RFC 6056 s'en occupe. Pour les numéros de séquence, voyons ce qui se passe. Tout paquet TCP doit inclure ces numéros, un pour chaque sens de la conversation (RFC 793, section 3.3). Ce numéro identifie l'octet suivant à envoyer. Une ouverture de connexion TCP normale ressemble donc, pour le cas où Alice veut parler à Bob, à :

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc1948.txt

— Alice-¿Bob : [SYN] ISNa, null (Alice envoie un paquet TCP où le bit `SYN` est mis, avec un numéro de séquence initial ISNa - ISN = "*Initial Sequence Number*".)
— Bob-¿Alice : [SYN] ISNb, ACK(ISNa+1) (Bob répond qu'il est d'accord, envoie son propre ISN - rappelez-vous qu'il y a deux séquences d'octets, une dans chaque direction - et accuse réception de l'ISN d'Alice.)
— Alice-¿Bob : [] ISNa, ACK(ISNb+1) (Alice est d'accord et cela termine la « triple poignée de mains » qui ouvre une connexion TCP.)

Voici d'ailleurs, vu par tcpdump, un exemple d'ouverture de connexion (vers le dépôt git de Linux) :

```
% tcpdump -S -n -r tmp/3wayhandshake.pcap
reading from file tmp/3wayhandshake.pcap, link-type EN10MB (Ethernet)
09:47:53.086347 IP 192.134.6.69.39859 > 149.20.4.72.9418: Flags [S], seq 3366079112, ...
09:47:53.242880 IP 149.20.4.72.9418 > 192.134.6.69.39859: Flags [S.], seq 3075029842, ack 3366079113, ...
09:47:53.242910 IP 192.134.6.69.39859 > 149.20.4.72.9418: Flags [.], ack 3075029843, ...
```

Le `S` entre crochets indique un paquet `SYN`. L'option `-S` est là pour indiquer à tcpdump d'afficher les numéros de séquence absolus, tels qu'ils sont envoyés sur le câble (par défaut, il montre des numéros relatifs, sauf pour les paquets `SYN`). On note aussi que tcpdump n'affiche pas le premier numéro de séquence s'il n'y a pas de données, et n'affiche pas le second si le bit `ACK` n'est pas mis.

Le deuxième paquet n'a été accepté par Alice que parce qu'il contenait un numéro de séquence attendu. Même chose pour le troisième. Un attaquant qui procéderait au hasard, envoyant des numéros de séquence quelconques, n'aurait aucune chance que ses paquets soient acceptés. Imaginons maintenant que l'attaquant puisse connaître les numéros de séquence utilisés. Bob fait confiance à Alice mais Mallory, l'attaquant, va se faire passer pour Alice. Mallory n'est pas situé sur le chemin entre Alice et Bob (sinon, cela serait trivial, il lui suffirait d'écouter le réseau) :

— AliceM-¿Bob : [SYN] ISNa, null (Mallory, utilisant l'adresse source d'Alice, envoie un paquet TCP où le bit `SYN` est mis, avec un numéro de séquence initial ISNa.)
— Bob-¿Alice : [SYN] ISNb, ACK(ISNa+1) (Bob répond qu'il est d'accord, envoie son propre ISN et accuse réception de l'ISN qu'il croit venir d'Alice. Mallory ne recevra **pas** ce paquet, qui ira à la vraie Alice. Mallory ne sait donc pas ce que vaut ISNb, et doit deviner. En cas de divination juste, son paquet suivant est accepté.)
— AliceM-¿Bob : [] ISNa, ACK(ISNb+1) (Mallory, utilisant l'adresse source d'Alice, termine la triple poignée de mains. Bob croit qu'il est connecté à Alice alors qu'il va accepter les paquets de Mallory.)

Et comment Mallory a-t-il trouvé le numéro de séquence que va utiliser Bob ? Le RFC originel, le RFC 793 ne mettait pas du tout en garde contre cette attaque, inconnue à l'époque (1981). Il ne se préoccupait que du risque de collision accidentel entre deux connexions et suggérait donc un algorithme où l'ISN était simplement incrémenté toutes les quatre micro-secondes, ce qui le rendait très prévisible (l'attaquant ouvre une connexion, regarde l'ISN, et sait quel va être celui de la prochaine connexion). Le premier article décrivant une faiblesse dans les mises en œuvre de TCP, faiblesse permettant de trouver le prochain numéro de séquence, a été publié en 1985 (Morris, R., "*A Weakness in the 4.2BSD UNIX TCP/IP Software*" `<http://pdos.csail.mit.edu/rtm/papers/117.pdf>`, CSTR 117, AT&T Bell Laboratories). À l'époque, elle était suffisante pour établir des sessions rsh avec la victime.

À noter qu'Alice, elle, reçoit l'accusé de réception de Bob, auquel elle ne s'attend pas. Elle va donc émettre un paquet `RST` qui va fermer la connexion. Diverses techniques permettent à l'attaquant de réussir quand même (par exemple en montant au même moment une attaque DoS contre Alice pour ralentir ses paquets, donnant à l'attaque le temps de réussir).

C'est pour répondre à cette attaque que le RFC 1948 avait été écrit en 1996, documentant la pratique qui était devenue courante, de générer des numéros de séquence, sinon aléatoires, en tout cas imprévisibles. (Un article plus récent et plus détaillé sur la vulnérabilité initiale était celui de Shimomura, T., "*Technical details of the attack described by Markoff in NYT*" `<http://www.gont.com.ar/docs/post-shimomura-usenet.txt>`, envoyé sur Usenet, en `comp.security.misc`, Message-ID : `<3g5gkl$5j1@ariel.sdsc.edu>`, en 1995.) Notre RFC 6528 met à jour ce RFC 1948. La section 2 expose les principes d'un bon algorithme de génération d'ISN ("*Initial Sequence Number*"), la section 3 décrit l'algorithme proposé. Commençons par celui-ci. Il tient en une formule :

```
ISN = M + F(localip, localport, remoteip, remoteport)
```

où M est un compteur qui s'incrémente toutes les quatre micro-secondes, et F une fonction pseudo-aléatoire, qui va prendre comme paramètre le tuple qui identifie une connexion TCP (adresse IP locale, port local, adresse IP distante, port distant). F ne doit pas être prévisible de l'extérieur et le RFC suggère une fonction de hachage cryptographique, comme MD5 (RFC 1321), hachant le tuple ci-dessus après l'avoir concaténé à une clé secrète. La sécurité reposera sur cette clé, l'algorithme de hachage utilisé étant sans doute connu. On peut envisager qu'elle soit générée aléatoirement (vraiment aléatoirement, en suivant le RFC 4086), ou configurée à la main (en espérant que l'ingénieur système ne choisira pas « toto »). Il est recommandé de la changer de temps en temps.

Notez que, bien que MD5 ait de nombreuses faiblesses cryptographiques pour d'autres utilisations, il est parfaitement sûr pour celle-ci (et bien plus rapide que ses concurrents).

La section 2, elle, définissait le cahier des charges : qu'attend t-on d'un algorithme de sélection de l'ISN ? D'abord, il n'est pas forcé d'empêcher des vieux paquets IP qui traînent dans le réseau d'être acceptés. Pour cela, il vaut mieux compter sur l'état `TIME_WAIT` de TCP (après la fin d'une connexion, garder l'état pendant deux MSL - "*Maximum Segment Life*", cf. RFC 7323). Ça ne marche pas forcément si les connexions sont créées à un rythme très rapide mais c'est la seule solution sûre (mais consommant de la mémoire). On peut lire à ce sujet deux articles sur les générateurs d'ISN, leurs avantages et inconvénients, « "*Strange Attractors and TCP/IP Sequence Number Analysis*" <`http://lcamtuf.coredump.cx/oldtcp/tcpseq.html`> » et « "*Strange Attractors and TCP/IP Sequence Number Analysis - One Year Later*" <`http://lcamtuf.coredump.cx/newtcp/`> ».

Compte-tenu de ces risques, une génération aléatoire de l'ISN suffirait. Mais elle aurait l'inconvénient de de ne pas permettre de deviner si un nouveau paquet TCP appartient à une ancienne connexion ou pas. L'idée de base de ce RFC est donc de créer un espace de numéros de séquence par identificateur TCP (le tuple {adresse IP locale, port local, adresse IP distante, port distant}) et, dans chacun de ces espaces, de suivre les règles d'incrémentation progressive de l'ISN du RFC 793.

Pour ceux qui veulent pousser l'analyse de sécurité, la section 4 prend de la hauteur et rappelle quelques points importants. D'abord que la vraie solution serait évidemment de protéger les paquets IP par de la cryptographie comme le fait IPsec, ou au niveau TCP comme le fait le TCP-AO du RFC 5925 (TLS ou SSH ne suffisent pas, ils n'empêchent pas une attaque par déni de service avec des faux paquets TCP `RST`).

La section 4 fait aussi remarquer que rien n'est parfait et que l'algorithme proposé dans notre RFC a des effets de bord, comme de permettre le comptage du nombre de machines qui se trouvent derrière un routeur NAT, ce qui peut être une information qu'on souhaiterait garder privée.

Et, surtout, elle rappelle que toutes les précautions de notre RFC 6528 ne s'appliquent que si l'attaquant est en dehors du chemin entre les deux pairs TCP. S'il est sur le chemin et qu'il peut écouter le réseau, elles ne servent à rien.

Quels sont les changements depuis le RFC 1948 ? Il n'y a pas de changement sur le fond (l'algorithme est le même donc un TCP compatible avec le RFC 1948 l'est également avec le RFC 6528). Les changements sont résumés dans l'annexe B. Le statut du RFC a changé (désormais sur le chemin des normes), les exemples maintenant dépassés par l'évolution des techniques ont été mis à jour ou supprimés,

On l'a dit, les précautions décrites dans ce RFC sont présentes dans les implémentations depuis longtemps. Pour un exemple de mise en œuvre de cet algorithme, on peut regarder le code source de NetBSD. Il se trouve en `sys/netinet/tcp_subr.c`. Son comportement est contrôlé par la variable `sysctl iss_hash`. Mise à 1, elle indique qu'il faut utiliser l'algorithme ci-dessus. À 0, qu'il faut utiliser une valeur aléatoire. Prenons le premier cas, celui de notre RFC. On voit la génération du secret (un nombre aléatoire, issu de `cprng_strong()`) puis le hachage MD5 de l'identifiant de connexion avec le secret :

```
/*
 * If we haven't been here before, initialize our cryptographic
 * hash secret.
 */
if (tcp_iss_gotten_secret == false) {
cprng_strong(kern_cprng,
     tcp_iss_secret, sizeof(tcp_iss_secret), 0);
tcp_iss_gotten_secret = true;
}

        /*
 * Compute the base value of the ISS.  It is a hash
 * of (saddr, sport, daddr, dport, secret).
 */
MD5Init(&ctx);
MD5Update(&ctx, (u_char *) laddr, addrsz);
MD5Update(&ctx, (u_char *) &lport, sizeof(lport));
MD5Update(&ctx, (u_char *) faddr, addrsz);
MD5Update(&ctx, (u_char *) &fport, sizeof(fport));
MD5Update(&ctx, tcp_iss_secret, sizeof(tcp_iss_secret));
MD5Final(hash, &ctx);
memcpy(&tcp_iss, hash, sizeof(tcp_iss));

        return (tcp_iss);
```

Pour Linux (remerciements au passage à @ackrst <http://twitter.com/ackrst>), il faut regarder le `net/core/secure_seq.c` qui contient à peu près la même chose (la variable `net_secret` est initialisée au tout début de ce fichier) :

```
__u32 secure_tcp_sequence_number(__be32 saddr, __be32 daddr,
 __be16 sport, __be16 dport)
{
u32 hash[MD5_DIGEST_WORDS];

hash[0] = (__force u32)saddr;
hash[1] = (__force u32)daddr;
hash[2] = ((__force u16)sport << 16) + (__force u16)dport;
hash[3] = net_secret[15];

md5_transform(hash, net_secret);
```