RFC 9102 : TLS DNSSEC Chain Extension

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 12 août 2021

Date de publication du RFC : Août 2021

https://www.bortzmeyer.org/9102.html

Lorsqu'on authentifie un serveur TLS (par exemple un serveur DoT) avec DANE, le client TLS doit utiliser un résolveur DNS <https://www.bortzmeyer.org/resolveur-dns.html> validant et tenir compte de son résultat (données validées ou pas), ce qui n'est pas toujours facile, ni même possible dans certains environnements. Ce RFC propose une solution pour cela : une extension à TLS, dnssec_chain, qui permet au serveur TLS d'envoyer l'ensemble des enregistrements DNS pertinents au client, le dispensant ainsi de solliciter un résolveur validant. Toute la cuisine de vérification peut ainsi se faire sans autre canal que TLS.

TLS (normalisé dans les RFC 5246[1] et RFC 8446) fournit un canal de communication sécurisé, si on a authentifié le serveur situé en face. La plupart du temps, on authentifie via un certificat PKIX (RFC 5280). Dans certains cas, ce n'est pas facile d'utiliser PKIX. Par exemple, pour DoT (RFC 7858), le résolveur DoT n'est typiquement connu que par une adresse IP, alors que le certificat ne contient en général qu'un nom (oui, le RFC 8738 existe, mais n'est pas toujours activé par les autorités de certification). Les serveurs DoT sont souvent authentifiables par DANE (RFC 6698 et RFC 7671). Par exemple, mon résolveur DoT public, dot.bortzmeyer.fr, est authentifiable avec DANE (regardez les données TLSA de _853._tcp.dot.bortzmeyer.fr). Mais DANE dépend de la disponibilité d'un résolveur DNS <https://www.bortzmeyer.org/resolveur-dns.html> validant. Diverses raisons font que le client TLS peut avoir du mal à utiliser un résolveur validant, et à récupérer l'état de la validation (toutes les API de résolution de noms ne donnent pas accès à cet état). L'article « "*Discovery method for a DNSSEC validating stub resolver*" <https://www.nlnetlabs.nl/downloads/publications/os3-2015-rp2-xavier-torrent-gorjon.pdf> » donne des informations sur ces difficultés. Bref, devoir récupérer ces enregistrements DANE de manière sécurisée peut être un problème. Et puis utiliser le DNS pour récupérer des données permettant d'authentifier un résolveur DNS pose un problème d'œuf et de poule.

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

D'où l'idée centrale de ce RFC : le client TLS demande au serveur de lui envoyer ces enregistrements. Le client n'a plus « que » à les valider à partir d'une clé de confiance DNSSEC (typiquement celle de la racine, commune à tout le monde). Le serveur peut aussi, s'il n'utilise pas DANE, renvoyer la preuve de non-existence de ces enregistrements, le client saura alors, de manière certaine, qu'il peut se rabattre sur un autre mécanisme d'authentification. L'extension TLS de notre RFC peut servir à authentifier un certificat complet, ou bien une clé brute (RFC 7250). Dans ce dernier cas, cette extension protège en outre contre les attaques dites « "*Unknown Key-Share*" ».

Notez le statut de ce RFC : cette spécification est pour l'instant expérimentale, elle n'a pas vraiment réuni de consensus à l'IETF, mais elle fait partie des solutions proposées pour sécuriser DoT, dans le RFC 8310 (section 8.2.2).

Bon, maintenant, l'extension elle-même (section 2). C'est une extension TLS (ces extensions sont spécifiées dans la section 4.2 du RFC 8446), elle est nommée `dnssec_chain` et enregistrée dans le registre IANA <`https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xml#tls-extensiontype-values-1`>, avec le code 59. Son champ de données (`extension_data`) vaut, dans le `ClientHello`, le port de connexion et, dans le `ServerHello`, un ensemble d'enregistrements DNS au format binaire du DNS (pour TLS 1.2 ; en 1.3, cet ensemble est attaché au certificat du serveur). Dans le langage de description des données TLS, cela donne, du client vers le serveur :

```
struct {
    uint16 PortNumber;
} DnssecChainExtension;
```

Et du serveur vers le client :

```
struct {
    uint16 ExtSupportLifetime;
    opaque AuthenticationChain<1..2^16-1>
} DnssecChainExtension;
```

(Le `ExtSupportLifetime` indique que le serveur s'engage à continuer à gérer cette extension pendant au moins ce temps - en heures. Si ce n'est plus le cas avant l'expiration du délai, cela peut indiquer une usurpation par un faux serveur.) Cette extension est donc une forme, assez spéciale, de « DNS sur TLS ». Le client ne doit pas oublier d'envoyer un SNI (RFC 6066), pour que le serveur sache quel nom va être authentifié et donc quels enregistrements DNS renvoyer. Quant au serveur, s'il ne gère pas cette extension, il répond au client avec un `ServerHello` n'ayant pas l'extension `dnssec_chain`. Même chose si le serveur accepte l'extension mais, pour une raison ou pour une autre, n'a pas pu rassembler tous les enregistrements DNS.

Et pourquoi le client doit-il indiquer le port auquel il voulait se connecter ? Le serveur le connait, non ? Mais ce n'est pas forcément le cas, soit parce qu'il y a eu traduction de port quelque part sur le chemin, soit parce que le client a été redirigé, par exemple via le type d'enregistrement SVCB (RFC 9460). Le port indiqué par le client doit être le port original, avant toute traduction ou redirection.

Revenons à la « chaine », l'ensemble d'enregistrements DNS qui va permettre la validation DANE. Le format est le format binaire habituel du DNS, décrit dans le RFC 1035, section 3.2.1. L'utilisation de ce format, quoiqu'il soit inhabituel dans le monde TLS, permet, entre autres avantages, l'utilisation des bibliothèques logicielles DNS existantes. Le RFC recommande d'inclure dans l'ensemble d'enregistrements une chaine complète, y compris la racine et incluant donc les clés DNSSEC de celle-ci (enregistrements DNSKEY), même si le client les connait probablement déjà. Cela donne par exemple la chaine suivante, pour le serveur `www.example.com` écoutant sur le port 443. Notez qu'on inclut les signatures (RRSIG) :

— _443._tcp.www.example.com TLSA
— RRSIG(_443._tcp.www.example.com TLSA)
— example.com. DNSKEY
— RRSIG(example.com DNSKEY)
— example.com DS
— RRSIG(example.com DS)
— com DNSKEY
— RRSIG(com DNSKEY)
— com DS
— RRSIG(com DS)
— . DNSKEY
— RRSIG(. DNSKEY)

(L'exemple suppose que _443._tcp.www.example.com et example.com sont dans la même zone.) Voilà, avec tous ces enregistrements, le client peut, sans faire appel à un résolveur DNS validant, vérifier (s'il a confiance dans la clé de la racine) l'authenticité de l'enregistrement DANE (type TLSA) et donc le certificat du serveur. De la même façon, si on n'a pas d'enregistrement TLSA, le serveur peut envoyer les preuves de non-existence (enregistrements NSEC ou NSEC3). Bon, évidemment, dans ce cas, c'est moins utile, autant ne pas gérer l'extension dnssec_chain…Face à ce déni, le client TLS n'aurait plus qu'à se rabattre sur une autre méthode d'authentification.

Le serveur TLS construit la chaine des enregistrements comme il veut, mais pour l'aider, la section 3 du RFC fournit une procédure possible, à partir d'interrogations du résolveur de ce serveur. Une autre possibilité, plus simple pour le serveur, serait d'utiliser les requêtes chainées du RFC 7901 (mais qui n'ont pas l'air très souvent déployées aujourd'hui).

Comme tous les enregistrements DNS, ceux inclus dans l'extension TLS dnssec_chain ont une durée de vie. Et les signatures DNSSEC ont une période de validité (en général plus longue que la durée de vie). Le serveur TLS qui construit l'ensemble d'enregistrements qu'il va renvoyer peut donc mémoriser cet ensemble, dans la limite du TTL et de l'expiration des signatures DNSSEC. Le client TLS peut lui aussi mémoriser ces informations, dans les mêmes conditions.

On a vu plus haut que les données de l'extension TLS incluaient un ExtSupportLifetime qui indiquait combien de temps le client pouvait s'attendre à ce que le serveur TLS continue à gérer cette extension. Car le client peut épingler ("*pinning*") cette information. Cela permet de détecter certaines attaques (« ce serveur gérait l'extension dnssec_chain et ce n'est maintenant plus le cas ; je soupçonne un détournement vers un serveur pirate »). Cet engagement du serveur est analogue au HSTS de HTTPS (RFC 6797) ; dans les deux cas, le serveur s'engage à rester « sécurisé » pendant un temps minimum (mais pas infini, car on doit toujours pouvoir changer de politique). À noter que « gérer l'extension dnssec_chain » n'est pas la même chose que « avoir un enregistrement TLSA », on peut accepter l'extension mais ne pas avoir de données à envoyer (il faudra alors envoyer la preuve de non-existence mentionnée plus haut). Bien sûr, le client TLS est libre de sa politique. Il peut aussi décider d'exiger systématiquement l'acceptation de l'extension TLS dnssec_chain (ce qui, aujourd'hui, n'est réaliste que si on ne parle qu'à un petit nombre de serveurs).

Si un serveur gérait l'extension dnssec_chain mais souhaite arrêter, il doit bien calculer son coup : d'abord réduire ExtSupportLifetime à zéro puis attendre que la durée annoncée dans le précédent ExtSupportLifetime soit écoulée, afin que tous les clients aient arrêté de l'épingler comme serveur à dnssec_chain. Il peut alors proprement stopper l'extension (cf. section 10 du RFC).

Un petit problème se pose si on fait héberger le serveur TLS chez un tiers. Par exemple, imaginons que le titulaire du domaine boulanger.example ait un serveur chez la société JVL (Je Vous Loge) et que server.boulanger.example soit un alias (enregistrement de type CNAME) vers clients.jvl.example.

Le client TLS va envoyer `server.boulanger.example` comme SNI. Il ne sera pas pratique du tout pour JVL de coordonner la chaine d'enregistrements DNS et le certificat. Il est donc conseillé que l'enregistrement TLSA du client soit lui aussi un alias vers un enregistrement TLSA de l'hébergeur. Cela pourrait donner (sans les signatures, pour simplifier la liste) :

— `server.boulanger.example` CNAME (vers `clients.jvl.example`)
— `_443._tcp.server.boulanger.example` CNAME (vers `_dane443.node1.jvl.example`)
— `clients.jvl.example` AAAA
— `_dane443.node1.jvl.example` TLSA

Les deux premiers enregistrements sont gérés par l'hébergé, les deux derniers par l'hébergeur. (La section 9 du RFC explique pourquoi l'enregistrement TLSA de l'hébergeur n'est pas en `_443._tcp`...)

Comme dit plus haut, le client TLS est maitre de sa politique : il peut exiger l'extension TLS, il peut l'accepter si elle existe, il peut l'ignorer. Si l'authentification par DANE échoue mais que celle par PKIX réussit, ou le contraire, c'est au client TLS de décider, en fonction de sa politique, ce qu'il fait.

Est-ce que le serveur TLS qui gère cette extension doit envoyer la chaine complète de certificats ? S'il veut pouvoir également être identifié avec PKIX, oui. Si non, s'il se contente de DANE et plus précisément de DANE-EE ou DANE-TA (ces deux termes sont définis dans le RFC 7218), il peut envoyer uniquement le certificat du serveur (pour DANE-EE).

Question mise en œuvre, l'excellente bibliothèque ldns <`https://www.nlnetlabs.nl/projects/ldns/`> a (je n'ai pas testé...) tout ce qu'il faut pour générer et tester ces chaines <`https://www.nlnetlabs.nl/documentation/ldns/structldns__dnssec__data__chain__struct.html`> d'enregistrements DNS. Si vous voulez développer du code pour gérer cette extension, l'annexe A du RFC contient des vecteurs de test qui vous seront probablement bien utiles.