RFC 7858 : Specification for DNS over Transport Layer Security (TLS)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 18 mai 2016

Date de publication du RFC : Mai 2016

http://www.bortzmeyer.org/7858.html

Traditionnellement, les requêtes DNS voyageaient sur le réseau en clair, visibles par tous. Ce n'est évidemment plus acceptable, notamment depuis les révélations de Snowden, et ce RFC normalise donc un mécanisme qui permet de chiffrer les requêtes DNS pour en assurer la confidentialité vis-à-vis d'un éventuel surveillant.

Le chiffrement est le deuxième pilier de la protection des données, après la minimisation des données (qui, pour le DNS, est spécifiée dans le RFC 7816 [1]). Ce chiffrement est motivé notamment par le développement de la surveillance de masse (RFC 7258), par le souci de protéger la vie privée (RFC 6973) et par la prise de conscience des risques spécifiques au DNS (RFC 7626).

Il y avait plusieurs choix possibles pour le groupe de travail DPRIVE <https://tools.ietf.org/wg/dprive> qui a produit ce RFC. Il pouvait utiliser une technique extérieure à l'IETF comme DNSCrypt. Il pouvait s'appuyer sur un protocole existant comme IPsec ou TLS. Et il pouvait développer un nouveau protocole (plusieurs propositions avaient été faites en ce sens). Finalement, le choix s'est porté sur un protocole facile à déployer, bien connu, et éprouvé, TLS. Pour résumer techniquement ce RFC (section 1) : on met les requêtes DNS sur un canal TLS lui-même évidemment transporté sur TCP. (Le groupe DPRIVE travaille également sur une future solution utilisant DTLS et donc UDP.) Outre la confidentialité, TLS permet de protéger les données contre une modification en cours de route.

Notons au passage que DNSSEC ne résout pas ces problèmes : son but est d'authentifier les réponses, pas de protéger contre l'espionnage.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7816.txt

Il y a deux liens à protéger dans une requête DNS, celui entre la machine de l'utilisateur et le résolveur, et celui entre le résolveur et les serveurs faisant autorité. La charte du groupe DPRIVE précisait qu'il fallait d'abord s'attaquer au premier problème, plus facile (une machine ne parle qu'à peu de résolveurs, et elle a une relation avec eux, alors qu'un résolveur parle à beaucoup de serveurs faisant autorité, qu'il ne connait pas.)

La section 3 décrit la gestion des connexions TLS. TLS tourne sur TCP (un de ses copains, DTLS, tourne sur UDP), et il faut donc utiliser ce protocole (notez que l'évolution récente du DNS fait de plus en plus de place à TCP, cf. RFC 7766). DNS-sur-TLS tourne sur un port fixe, le port 853 (les premières versions de ce protocole utilisaient un protocole de négociation de TLS, comme pour SMTP mais, trop compliqué et trop dangereux, il a été abandonné). Le serveur DNS-sur-TLS écoute donc sur le port 853, le client se connecte à ce port, et démarre tout de suite une session TLS (RFC 5246). Ensuite, il y fait circuler le DNS, encodé comme sur TCP (le message DNS précédé de deux octets indiquant sa taille, cf. RFC 1035, section 4.2.2). Comme pour tout usage de DNS sur TCP (RFC 7766), le client et le serveur ne doivent évidemment pas couper la connexion après une seule requête. S'ils le faisaient, les performances seraient catastrophiques, vu le temps qu'il faut pour établir la connexion TCP **et** la session TLS. Au contraire, la session doit être réutilisée (section 3.4) pour les requêtes ultérieures (et le RFC demande aussi qu'un serveur traite les requêtes en parallèle, pas séquentiellement dans leur ordre d'arrivée, section 3.3.) Un autre moyen de gagner du temps lors de l'établissement d'une nouvelle connexion TCP, moyen recommandé par notre RFC, est d'utiliser TCP Fast Open (RFC 7413).

Notez bien la règle : jamais de contenu chiffré sur le port 53, jamais de contenu en clair sur le port 853. Si le serveur ne répond pas aux requêtes sur le port 853, cela peut être parce qu'une affreuse "*middlebox*" bloque tout le trafic vers le port 853, ou tout simplement parce que ce résolveur ne gère pas encore DNS sur TLS. Le client doit alors décider, selon sa politique de sécurité, s'il se rabat sur du trafic DNS en clair ou bien s'il renonce à communiquer. Le RFC recommande que le client se souvienne des serveurs qui répondent sur le port 853, pour détecter un filtrage récemment apparu.

À noter aussi que je n'ai pas parlé encore d'authentification du serveur. Comment être sûr qu'on parle bien au serveur à qui on veut parler et pas au terrible Homme du Milieu ? Pour l'instant, gardez la question de côté, on y reviendra.

Dit comme cela, c'est peut-être un peu compliqué, mais l'un des gros avantages du choix de TLS est qu'il existe déjà des bibliothèques toutes faites pour ce protocole. Ajouter TLS à un client ou un serveur existant est donc relativement simple. (Plus simple que la gestion du RFC 7766, qui est un pré-requis pour DNS-sur-TLS.)

Revenons à l'authentification. A priori, elle est indispensable. Si on n'authentifie pas, on risque d'envoyer ses requêtes DNS à un serveur qui n'est pas le bon, et, dans ce cas, le chiffrement n'aura servi à rien. Pour se faire passer pour le bon serveur DNS, les attaquants actifs ont bien des moyens à leur disposition, comme d'injecter des fausses routes, comme en Turquie <`http://www.bortzmeyer.org/dns-routing-hijack-turkey.html`>. Un chiffrement sans authentification ne protégerait que contre des attaquants strictement passifs.

Oui, mais l'authentification, c'est difficile. Il y a plein de pièges, de problèmes possibles, et d'argent à dépenser (par exemple pour acheter un certificat). Imposer dès le début une authentification stricte dans tous les cas aurait tué le projet dans l'œuf. Le choix a donc été de prévoir un certain nombre de **profils** d'authentification, que le client choisit, en fonction du degré de sécurité souhaité. (Le serveur, lui, n'authentifie pas le client, en tout cas aucun mécanisme n'est prévu pour cela.) Notre RFC propose deux profils d'authentification (section 4), et d'autres ont décrits par la suite dans le RFC 8310.

Le premier profil est celui « opportuniste » (*"opportunistic privacy"*). Le client DNS qui utilise ce profil va tenter de chiffrer, il peut même essayer d'authentifier mais, si cela échoue, il continue quand même à envoyer des requêtes DNS, se disant qu'il est au moins protégé contre les attaques passives.

Le second profil est celui de l'« épinglage de clé » (*"out-of-band key-pinned"* »). La clé publique (SPKI pour *"Subject Public Key Info"*) du serveur DNS est connue du client (par un mécanisme non spécifié dans ce RFC), et il ne se connecte à un serveur DNS-sur-TLS que s'il utilise une des clés correspondant à son adresse. C'est un mécanisme analogue à l'épinglage HTTP du RFC 7469. Une syntaxe possible, pour un `/etc/resolv.conf` sur Unix serait, par exemple (l'annexe A propose une autre syntaxe) :

```
domain example.net
nameserver 2001:db8:90e:241a::1  NjAwZGI5YTVhMzBiY2EzOWY1OTY5YzM0ZGYzMTllYTVkNmYxNjAwYjJmNzZhOGFhYTNhMDEyMzU4MTI
```

Où le dernier champ est le condensat de la clé publique.

Au passage, si vous voulez trouver le condensat de votre clé, vous pouvez le faire avec `gnutls-cli` (cela dépend de comment est fait l'échange de clés) ou bien, si le certificat est dans le fichier `tmp.pem`, avec cette commande utilisant OpenSSL :

```
openssl x509 -pubkey -in tmp.pem | openssl pkey -pubin -outform der | openssl dgst -sha256 -binary | base64
```

Ce mécanisme est sûr, empêchant toute attaque de l'Homme du Milieu. Il est peut-être même trop sûr, par exemple pour les réseaux qui utilisent ces horribles portails captifs qui font des attaques de l'Homme du Milieu pour vous rediriger vers le portail. Il peut donc être préférable de ne pas activer ce profil avant de s'être authentifié auprès du portail. (dnssec-trigger <`http://www.bortzmeyer.org/dnssec-trigger.html`> utilise un mécanisme analogue.)

J'ai déjà parlé des performances au moment de l'établissement de la connexion, et des problèmes de latence qui peuvent survenir en raison de l'utilisation de TCP et de TLS. La section 5 revient sur ces histoires (il est également recommandé de lire le rapport « *"T-DNS : Connection-Oriented DNS to Improve Privacy and Security"* <`ftp://ftp.isi.edu/isi-pubs/tr-688.pdf`> »). Évidemment, une connexion TCP plus une requête DNS, c'est plus coûteux qu'une requête DNS tout court. La requête et la réponse DNS nécessitent deux voyages entre client et serveur, l'établissement de la connexion TCP en nécessite trois à lui seul, et TLS deux de plus. On pourrait en déduire que DNS-sur-TLS-sur-TCP aura une latence <`http://www.bortzmeyer.org/latence.html`> plus élevée de 250 %. On peut réduire ce coût avec des trucs TCP qui réduisent le temps d'établissement de la connexion, comme le *"TCP Fast Open "* du RFC 7413 et avec des trucs TLS comme la reprise de session du RFC 5077. Mais, de toute façon, ce calcul n'est vrai que si on établit une connexion TCP par requête DNS, ce qui n'est pas conseillé, la bonne méthode étant au contraire, dans l'esprit du RFC 7766, de réutiliser les connexions.

D'autre part, TCP nécessite de stocker un état dans le serveur. Pour éviter que beaucoup de clients n'écroulent celui-ci, il faut donc ajuster les délais d'inactivité, pour couper les connexions TCP inutilisées.

La section 7 de notre RFC intéressera les ingénieurs qui se demandent pourquoi les choses sont comme elles sont et pourquoi un autre choix n'a pas été fait. Elle est consacrée à l'évolution de la solution de chiffrement du DNS, au sein du groupe de travail DPRIVE. Comme indiqué plus haut, le premier projet prévoyait de tout faire passer sur le port 53, avec un passage en TLS à la demande du client,

lorsqu'il envoyait une requête DNS « bidon » avec un nouveau bit EDNS, TO ("*TLS OK*"), mis à un, et avec le QNAME ("*Query NAME*") `STARTTLS` (reprenant un mot-clé utilisé par SMTP, dans le RFC 3207).

Cette méthode avait des avantages : elle permettait par exemple à un serveur d'accepter TLS quand il n'était pas trop chargé, et de le refuser autrement. Et, réutilisant le port existant, elle ne nécessitait pas de changer les ACL sur les pare-feux. Mais elle avait aussi des inconvénients : les affreuses "*middleboxes*" ont une longue habitude d'interférence avec EDNS et il n'était pas du tout sûr qu'elles laissent passer le bit TO. Et puis, même si le RFC ne le mentionne pas, il y avait un risque qu'une "*middlebox*" trop zélée ne fasse du DPI sur le port 53, s'aperçoive que ce qui passe n'est pas du DNS, et coupe la communication. Mais le principal problème de cette approche était qu'elle rendait les attaques par repli triviales. Un attaquant actif n'avait qu'à supprimer le bit TO et le client n'avait alors plus aucun moyen de savoir si l'absence de TLS était due à un serveur trop ancien, à une "*middlebox*" boguée... ou bien à une attaque par repli.

Une proposition alternative amusante avait été de mêler le trafic chiffré et non chiffré sur le port 53 sans signalisation : la structure de l'en-tête TLS est telle qu'une machine interprétant le TLS comme étant du DNS en clair aurait vu une réponse DNS (bit QR à 1) et il n'y aurait donc pas eu de confusion avec le trafic DNS en clair. Astucieux mais évidemment très fragile.

La section 8 de notre RFC synthétise les questions de sécurité. D'abord, TLS n'est évidemment pas une formule magique. Il y a eu plein d'attaques contre TLS (par exemple pour forcer un repli vers des algorithmes de chiffrement faibles), ou contre ses mises en œuvre (on pense évidemment tout de suite à OpenSSL). Pour éviter cela, outre le respect des bonnes pratiques TLS (RFC 7525), le client prudent tâchera de se souvenir quels serveurs acceptaient DNS-sur-TLS. Si un serveur qui l'acceptait ne répond tout à coup plus sur le port 853, c'est peut-être qu'un attaquant tente de forcer un repli sur le port 53, en clair. Le client prudent peut ainsi détecter une attaque possible. Si c'est un nouveau serveur, que le client ne connait pas, la marche à suivre dépend de la politique du client (sécurisé ou laxiste).

Quant aux attaques non-TLS (comme le blocage du port 853 mentionné ci-dessus), c'est également au client, en fonction de son profil de sécurité, de décider ce qu'il va faire (renoncer à communiquer, essayer un mécanisme de résolution alternatif, s'en foutre et tout passer en clair, etc).

Revenons à TLS pour noter que ce protocole ne fait pas d'effort pour dissimuler la taille des paquets. Un attaquant passif peut donc, en observant cette taille, et d'autres informations comme le temps écoulé entre deux paquets, en déduire certaines informations, malgré le chiffrement. L'option de remplissage du RFC 7830 permet de remplir les paquets avec des données bidons, afin de rendre cette analyse plus difficile.

Pour un bilan d'étape du projet « DNS et vie privée » à l'IETF, vous pouvez regarder mon exposé "*State of the "DNS privacy" project : running code*" à l'OARC <`https://indico.dns-oarc.net/event/22/session/2/contribution/2/material/slides/1.pdf`>.

Question mises en œuvre de ce RFC, où en est-on ? Aujourd'hui, le résolveur Unbound a le code nécessaire depuis longtemps (depuis la version 1.4.14). On peut génerer les clés nécessaires avec OpenSSL ainsi :

```
openssl req -x509 -newkey rsa:4096 \
      -keyout privatekeyfile.key -out publiccertfile.pem \
      -days 1000 -nodes
```

et configurer le serveur ainsi :

```
server:
  interface: 2001:db8:1::dead:beef@853
  ssl-service-key: "/etc/unbound/privatekeyfile.key"
  ssl-service-pem: "/etc/unbound/publiccertfile.pem"
  ssl-port: 853
```

Unbound va alors activer DNS sur TLS au démarrage et annoncer fièrement « *"setup TCP for SSL [sic] service"* ». Les clients pourront alors l'interroger en DNS sur TLS.

Bon, mais quel client utiliser ? Dans la bibliothèque getdns <https://getdnsapi.net/>, le logiciel d'exemple `getdns_query` sait faire du DNS sur TLS :

```
% ./getdns_query @2001:db8:1::dead:beef -s -a -A -l L www.bortzmeyer.org
...
Response code was: GOOD. Status was: At least one response was returned
```

(C'est l'option `-l L` qui lui indique de faire du TLS.)

Si on capture le trafic entre `getdns_query` et Unbound, on peut afficher le résultat avec tshark :

```
% tshark -n -d tcp.port==853,ssl -r /tmp/dnstls.pcap
 4   0.002996  2001:db8:1::63a:671 -> 2001:db8:1::dead:beef  SSL Client Hello
 6   0.594206  2001:db8:1::dead:beef -> 2001:db8:1::63a:671  TLSv1.2 Server Hello, Certificate, Server Key Excha
 8   0.734094  2001:db8:1::63a:671 -> 2001:db8:1::dead:beef  TLSv1.2 Client Key Exchange
16   0.751614  2001:db8:1::dead:beef -> 2001:db8:1::63a:671  TLSv1.2 Application Data
17   0.759223  2001:db8:1::63a:671 -> 2001:db8:1::dead:beef  TLSv1.2 Application Data
```

On voit un trafic TLS classique, chiffré. Notez que tshark, par défaut, ne sait pas que c'est du TLS sur le port 853 (cela sera fait lors de la prochaine version majeure). On lui indique donc explicitement (`-d tcp.port==853,ssl`).

Et si j'ai la flemme d'installer un Unbound configuré pour TLS, est-ce que je peux quand même tester un client DNS-sur-TLS ? Oui, si les serveurs de test publics listés en veulent bien répondre.

Question client, vous avez aussi le démon Stubby <https://portal.sinodun.com/wiki/display/TDNS/DNS+Privacy+daemon+-+Stubby>. Vous le lancez en indiquant le résolveur à qui il va tout relayer en DNS-sur-TLS :

```
%  sudo stubby @145.100.185.16 -L
```

Et vous pouvez alors l'utiliser comme résolveur (il écoute par défaut sur `127.0.0.1`.)

Et vous pouvez aussi utiliser Unbound comme client ! Par rapport à Stubby, il a l'avantage d'être un vrai résolveur, validant et avec un cache, et l'inconvénient de ne pas savoir authentifier le résolveur auquel il va parler. Pour configurer Unbound en client DNS-sur-TLS (et non plus en serveur comme dans l'exemple précédent) :

```
server:
  ...
  ssl-upstream: yes

forward-zone:
  name: "."
  forward-addr: 2001:4b98:dc2:43:216:3eff:fea9:41a@853
  forward-first: no
```

Android a une mise en œuvre de DNS-sur-TLS (cf. ce commit <https://android-review.googlesource.com/#/c/platform/system/netd/+/380593/>).

Si vous préférez développer en Go, l'excellente bibliothèque GoDNS <https://miek.nl/2014/August/16/go-dns-package/> gère DNS sur TLS (depuis janvier 2016) et vous permet de faire vos propres programmes. Par exemple, ce code Go :

```
c := new(dns.Client)
c.Net = "tcp-tls"
if *insecure {
        c.TLSConfig = new(tls.Config)
        c.TLSConfig.InsecureSkipVerify = true
}
in, rtt, err := c.Exchange(m, net.JoinHostPort(myResolver, "853"))
```

va ouvrir une connexion avec le serveur `myResolver`, sur le port 853, et utiliser TLS. Par défaut, la bibliothèque TLS de Go vérifie le certificat, et que le nom (ou l'adresse IP) dans le certificat correspond bien (ce qui est la bonne approche <http://soat.developpez.com/tutoriels/securite/utiliser-tls-s>). Mais cela peut être embêtant si on n'a pas acheté de certificat. D'où l'option (**dangereuse !**) pour débrayer la vérification (les trois lignes qui commencent par `if *insecure`). Voici un exemple d'utilisation (au fait, le programme est en (en ligne sur `http://www.bortzmeyer.org/files/dns-tls.go`), `-k` active l'option dangereuse) :

```
%  ./dns-tls  my-resolver internautique.fr
Error in query: x509: certificate signed by unknown authority

%  ./dns-tls -k my-resolver internautique.fr
(time 43051 micro-s) 2 keys. TC=false
```