RFC 8467 : Padding Policies for Extension Mechanisms for DNS (EDNS(0))

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 2 décembre 2018

Date de publication du RFC : Octobre 2018

https://www.bortzmeyer.org/8467.html

Chiffrer pour assurer la confidentialité, c'est bien. Pour le DNS, c'est ce que permet le RFC 7858 [1] (DNS sur TLS). Mais un problème de TLS et de pas mal d'autres protocoles cryptographiques est qu'il ne dissimule pas les métadonnées, et notamment la **taille** des messages échangés sur le réseau. Dans un monde public comme celui du DNS, c'est un problème. En effet, l'attaquant peut facilement mesurer la taille des réponses chiffrées (en envoyant lui-même une requête), voir la taille des réponses, et en déduire les questions qui avaient été posées. La solution classique en cryptographie face à ce risque est le remplissage, normalisé, pour le DNS, dans le RFC 7830. Mais le RFC 7830 ne normalisait que le format, pas le mode d'emploi. Il faut remplir jusqu'à telle taille ? Comment concilier un remplissage efficace pour la confidentialité avec le désir de limiter la consommation de ressources réseaux ? Ce RFC décrit plusieurs stratégies possibles, et recommande un remplissage jusqu'à atteindre une taille qui est le multiple suivant de 468 (octets).

Ce nouveau RFC tente de répondre à ces questions, en exposant les différentes politiques possibles de remplissage, leurs avantages et leurs inconvénients. Le RFC 7830 se limitait à la syntaxe, notre nouveau RFC 8467 étudie la sémantique.

D'abord, avant de regarder les politiques possibles, voyons les choses à garder en tête (section 3 du RFC). D'abord, ne pas oublier de mettre l'option EDNS de remplissage (celle du RFC 7830) en dernier dans la liste des options (car elle a besoin de connaitre la taille du reste du message).

Ensuite, il faut être conscient des **compromis** à faire. Remplir va améliorer la confidentialité mais va réduire la durée de vie de la batterie des engins portables, va augmenter le débit qu'on injecte dans le réseau, voire augmenter le prix si on paie à l'octet transmis. Lors des discussions à l'IETF, certaines

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7858.txt

personnes ont d'ailleurs demandé si le gain en confidentialité en valait la peine, vu l'augmentation de taille. En tout cas, on ne remplit les messages DNS que si la communication est chiffrée : cela ne servirait à rien sur une communication en clair.

Enfin, petit truc, mais qui montre l'importance des détails quand on veut dissimuler des informations, le remplissage doit se faire sans tenir compte des deux octets qui, avec certains protocoles de transport du DNS, comme TCP, peut faire fuiter des informations. Avec certaines stratégies de remplissage, les deux octets en question peuvent faire passer de l'autre côté d'un seuil et donc laisser fuiter l'information qu'on était proche du seuil.

Ensuite, après ces préliminaires, passons aux stratégies de remplissage, le cœur de ce RFC (section 4). Commençons par celle qui est la meilleure, et recommandée officiellement par notre RFC : remplissage en blocs de taille fixe. Le client DNS remplit la requête jusqu'à atteindre un multiple de 128 octets. Le serveur DNS, si le client avait mis l'option EDNS de remplissage dans la requête, et si la communication est chiffrée, remplit la réponse de façon à ce qu'elle soit un multiple de 468 octets. Ainsi, requête et réponse ne peuvent plus faire qu'un nombre limité de longueurs, la plupart des messages DNS tenant dans le bloc le plus petit. Voici un exemple vu avec le client DNS dig, d'abord sans remplissage :

```
% dig +tcp +padding=0 -p 9053 @127.0.0.1 SOA foobar.example
...
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29832
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
...
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
; COOKIE: dda704b2a06d65b87f0493105c03ca4d2b2c83f2d4e25680 (good)
;; QUESTION SECTION:
;foobar.example. IN SOA

;; ANSWER SECTION:
foobar.example. 600 IN SOA ns1.foobar.example. root.foobar.example. (
2015091000 ; serial
604800     ; refresh (1 week)
86400      ; retry (1 day)
2419200    ; expire (4 weeks)
86400      ; minimum (1 day)
)
...
;; MSG SIZE  rcvd: 116
```

La réponse fait 116 octets. On demande maintenant du remplissage, jusqu'à 468 octets :

```
% dig +tcp +padding=468 -p 9053 @127.0.0.1 SOA foobar.example
...
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2117
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
; COOKIE: 854d2f29745a72e5fdd6891d5c03ca4b5d5287daf716e327 (good)
; PAD (348 bytes)
;; QUESTION SECTION:
;foobar.example. IN SOA
```

```
;; ANSWER SECTION:
foobar.example. 600 IN SOA ns1.foobar.example. root.foobar.example. (
2015091000 ; serial
604800     ; refresh (1 week)
86400      ; retry (1 day)
2419200    ; expire (4 weeks)
86400      ; minimum (1 day)
)

;; MSG SIZE  rcvd: 468
```

La réponse fait 468 octets, grâce aux 348 octets de remplissage (notez la ligne `PAD (348 bytes)`).

Les avantages de cette méthode est qu'elle est facile à mettre en œuvre, assure une confidentialité plutôt bonne, et ne nécessite pas de générateur de nombres aléatoires. Son principal inconvénient est qu'elle permet de distinguer deux requêtes (ou deux réponses) si elles ont le malheur d'être remplies dans des blocs de taille différente. Mais il ne faut pas chercher une méthode idéale : rappelez-vous qu'il faudra faire des compromis. Cette méthode a un faible coût pour le défenseur, et élève les coûts sensiblement pour l'attaquant, c'est ça qui compte.

Notez que les chiffres 128 et 468 ont été obtenus empiriquement, en examinant du trafic DNS réel. Si DNSSEC continue à se répandre, les tailles des réponses moyennes augmenteront, et il faudra peut-être réviser ces chiffres.

Une autre statégie est celle du remplissage maximal. On met autant d'octets qu'on peut. Si un serveur a une taille maximale de réponse de 4 096 octets (la valeur par défaut la plus courante) et que le client accepte cette taille, on remplit la réponse jusqu'à ce qu'elle fasse 4 096 octets. L'avantage évident de cette méthode est qu'elle fournit la meilleure confidentialité : toutes les réponses ont la même taille. L'inconvénient évident est qu'elle est la méthode la plus consommatrice de ressources. En outre, ces grandes réponses vont souvent excéder la MTU, pouvant entrainer davantage de problèmes liés à la fragmentation.

Autre stratégie envisageable : remplissage aléatoire. On tire au sort le nombre d'octets à ajouter. Cela fournit une bonne distribution des tailles (par exemple, une réponse courte peut désormais être plus grande qu'une réponse longue, ce qui n'arrive jamais avec les deux stratégies précédentes). Inconvénient : comme ça ne change pas la limite de taille inférieure, un attaquant qui voit beaucoup de messages pourrait en déduire des informations. Et cela oblige à avoir un générateur de nombres aléatoires, traditionnellement un problème délicat en cryptographie.

Enfin, une dernière méthode raisonnable est de combiner le remplissage dans des blocs et le tirage au sort : on choisit au hasard une longueur de bloc et on remplit jusqu'à atteindre cette longueur. Contrairement à la précédente, elle n'a pas forcément besoin d'une source aléatoire à forte entropie. Mais c'est sans doute la technique la plus compliquée à mettre en œuvre.

La section 7 du RFC ajoute quelques points supplémentaires qui peuvent mettre en péril la confidentialité des requêtes. Par exemple, si le client DNS remplit correctement sa requête, mais que le serveur ne le fait pas, un attaquant pourra déduire la requête de la réponse (c'est d'autant plus facile, avec le DNS, que la question est répétée dans la réponse). Dans une communication de client à résolveur DNS, il faut bien choisir son résolveur.

Et le remplissage ne brouille qu'une seule des métadonnées. Il y en a d'autres comme l'heure de la question, le temps de réponse ou comme la succession des requêtes/réponses, qui restent accessibles à un éventuel attaquant. La protection contre la fuite d'informations via ces métadonnées nécessiterait d'injecter « gratuitement » du trafic de couverture (qui, lui aussi, éleverait la consommation de ressources réseau).

Et pour terminer le RFC, l'annexe A est consacrée aux mauvaises politiques de remplissage, celles qui non seulement ne sont pas recommandées mais sont activement déconseillées. (Mais on les trouve parfois dans du code réel.) Il y a l'évidente stratégie « pas de remplissage du tout ». Son principal intérêt est qu'elle fournit le point de comparaison pour toutes les autres stratégies. Avantages : triviale à implémenter, il suffit de ne rien faire, et aucune consommation de ressources supplémentaires. Inconvénient : la taille des requêtes et des réponses est exposée, et un observateur malveillant peut en déduire beaucoup de choses.

Une autre méthode inefficace pour défendre la vie privée est celle du remplissage par une longueur fixe. Elle est simple à implémenter mais ne protège rien : une simple soustraction suffit pour retrouver la vraie valeur de la longueur.

Ces différentes stratégies ont été analysées empiriquement (il n'y a pas vraiment de bon cadre pour le faire théoriquement) et le travail est décrit dans l'excellente étude de Daniel Kahn Gillmor (ACLU), « "*Empirical DNS Padding Policy*" <https://dns.cmrg.net/ndss2017-dprive-empirical-DNS-traffic-s pdf> » présentée à NDSS <https://www.ndss-symposium.org/> en 2017. Si vous aimez les chiffres et les données, c'est ce qu'il faut regarder !

Testons un peu les mises en œuvres du remplissage des messages DNS (une liste plus complète figure sur le site du projet <http://edns0-padding.org/implementations>).

Essayons avec BIND version 9.13.4. Il fournit le client de débogage dig et son option `+padding`. Avec un `+padding=256`, le datagramme va faire 264 octets, incluant les ports source et destination d'UDP). Vu par tshark, cela donne :

```
<Root>: type OPT
    Name: <Root>
    Type: OPT (41)
    UDP payload size: 4096
    Higher bits in extended RCODE: 0x00
    EDNS0 version: 0
    Z: 0x8000
        1... .... .... .... = DO bit: Accepts DNSSEC security RRs
        .000 0000 0000 0000 = Reserved: 0x0000
    Data length: 213
    Option: COOKIE
        Option Code: COOKIE (10)
        Option Length: 8
        Option Data: 722ffe96cd87b40a
        Client Cookie: 722ffe96cd87b40a
        Server Cookie: <MISSING>
    Option: PADDING
        Option Code: PADDING (12)
        Option Length: 197
        Option Data: 00000000000000000000000000000000000000000000000000...
        Padding: 00000000000000000000000000000000000000000000000000...
```

Cela, c'était la requête du client. Mais cela ne veut pas dire que le serveur va accepter de répondre avec du remplissage. D'abord, il faut qu'il soit configuré pour cela (BIND 9.13.4 ne le fait pas par défaut). Donc, côté serveur, il faut :

```
options {
...
response-padding {any;} block-size 468;
};
```

(`any` est pour accepter le remplissage pour tous les clients.) Mais cela ne suffit pas, BIND ne répond avec du remplissage que si l'adresse IP source est raisonnablement sûre (TCP ou biscuit du RFC 7873, pour éviter les attaques par amplification). C'est pour cela qu'il y a une option `+tcp` dans les appels de dig plus haut. On verra alors dans le résultat de dig le `PAD (392 bytes)` indiquant qu'il y a eu remplissage. La taille indiquée dans `response-padding` est une taille de bloc : BIND enverra des réponses qui seront un multiple de cette taille. Par exemple, avec `response-padding {any;} block-size 128;`, une courte réponse est remplie à 128 octets (notez que la taille de bloc n'est pas la même chez le serveur et chez le client) :

```
% dig +tcp +padding=468 -p 9053 @127.0.0.1 SOA foobar.example
...
; PAD (8 bytes)
...
;; MSG SIZE  rcvd: 128
```

Alors qu'une réponse plus longue (notez la question `ANY` au lieu de `SOA`) va faire passer dans la taille de bloc au dessus (128 octets est clairement une taille de bloc trop petite, une bonne partie des réponses DNS, même sans DNSSEC, peuvent la dépasser) :

```
% dig +tcp +padding=468 -p 9053 @127.0.0.1 ANY foobar.example
...
; PAD (76 bytes)
...
;; MSG SIZE  rcvd: 256
```

On voit bien ici l'effet de franchissement du seuil : une taille de bloc plus grande doit être utilisée.

BIND n'est pas forcément que serveur DNS, il peut être client, quand il est résolveur et parle aux serveurs faisant autorité. La demande de remplissage dans ce cas se fait dans la configuration par serveur distant, avec `padding`. (Je n'ai pas testé.)

Le résolveur Knot <`https://www.knot-resolver.cz/`> fait également le remplissage (option `net.tls_padding`) mais, contrairement à BIND, il ne le fait que lorsque le canal de communication est chiffré (ce qui est logique).

La bibliothèque pour développer des clients DNS getdns <`https://getdnsapi.net/`> a également le remplissage <`https://getdnsapi.net/pipermail/users/2015-November/000137.html`>. En revanche, Unbound, dans sa version 1.8.1, ne fait pas encore de remplissage. Et, comme vous avez vu dans l'exemple tshark plus haut, Wireshark sait décoder l'option de remplissage.