RFC 9267 : Common Implementation Anti-Patterns Related to Domain Name System (DNS) Resource Record (RR) Processing

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 14 septembre 2022

Date de publication du RFC : Juillet 2022

https://www.bortzmeyer.org/9267.html

Comme chacun·e sait, l'Internet est une jungle (les politiciens ajouteraient « une jungle Far-West de non-droit qu'il faut civiliser »). Par exemple, les logiciels qui parlent avec les vôtres ne sont pas toujours correctement écrits, voire ils sont franchement malveillants. Le code de votre côté doit donc être robuste, voire paranoïaque, et penser à tout. Ce RFC décrit quelques problèmes qui ont été observés dans des logiciels mettant en œuvre le DNS et explique comment ne pas refaire la même erreur.

Le RFC ne parle pas de failles DNS mais de failles dans les programmes DNS, ce qui est très différent (mais la différence est souvent oubliée dans les médias). Le protocole lui-même n'était pas en cause dans ces cas, ce sont juste les logiciels qui avaient des bogues. Les cas sont nombreux, par exemple SIGRed (CVE-2020-1350) <https://nvd.nist.gov/vuln/detail/CVE-2020-1350#vulnCurrentDescriptionTitle> ou DNSpooq (CVE-2020-25681 à CVE-2020-25687) <https://www.jsof-tech.com/wp-content/uploads/2021/01/DNSpooq-Technical-WP.pdf>. Ces problèmes frappent notamment souvent dnsmasq (personnellement, je n'ai jamais compris pourquoi ce logiciel était si utilisé, mais c'est une autre histoire).

Plusieurs vulnérabilités ont concerné l'analyse des enregistrements DNS (RR = "*Resource Record*"). Les risques lors de cette analyse devraient être bien connus, la première faille documentée l'ayant été en 2000 (CVE-2000-0333 <https://nvd.nist.gov/vuln/detail/CVE-2000-0333>)! Tout logiciel qui analyse des enregistrements DNS (client, serveurs, mais aussi pare-feux, IDS, etc) peut tomber dans ces pièges, d'où l'importance de les documenter. C'était déjà fait dans l'"*Internet-Draft*" draft-ietf-dnsind-local-com et dans le RFC 5625[1] mais c'était perdu au milieu d'autres choses donc notre RFC choisit d'enfoncer le clou.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc5625.txt

Il commence par le grand classique des bogues de logiciels DNS : le traitement des pointeurs de compression. Pour gagner quelques octets, à l'époque où ça comptait, le DNS prévoit (RFC 1035, section 4.1.4) qu'on peut, dans un enregistrement DNS, remplacer tout ou partie d'un nom de domaine par un **pointeur** vers un autre endroit du paquet. Ainsi, si on a tous ses serveurs qui se terminent par les mêmes composants <https://www.bortzmeyer.org/parties-nom-domaine.html> (comme c'est le cas, par exemple, de la racine <https://dns.bortzmeyer.org/root/NS>), on peut ne mettre ces composants qu'une fois, et pointer vers eux partout ailleurs. Évidemment, si vous êtes programmeuse ou programmeur, vous avez déjà vu le piège : les pointeurs, c'est dangereux. Ils peuvent pointer en dehors du paquet, ou pointer vers eux-même, par exemple. Si on suit aveuglément un pointeur, un déni de service est possible, si on tape en dehors de la mémoire allouée ou bien si on se lance dans une boucle sans fin.

Rentrons dans les détails. Le RFC 1035 nous dit que l'octet qui indique la longueur d'un composant doit valoir moins de 64 (la taille maximale d'un composant), donc avoir les deux bits de plus fort poids à zéro. S'ils sont tous les deux à un, cela indique qu'on a un pointeur. Cet octet et le suivant (privés des deux bits de plus fort poids) sont alors un pointeur, le nombre d'octets depuis le début du message. On voit qu'on peut atteindre 16 383 octets (2 ** 14 - 1), largement assez pour sortir de la mémoire allouée pour le paquet, avec les conséquences qu'on imagine. Cela s'est produit en vrai (CVE-2020-25767, CVE-2020-24339 et CVE-2020-24335).

Autre exemple amusant cité par le RFC, le cas d'un pointeur pointant sur lui-même. Soit un message DNS minimal. Le premier composant est à 12 octets du début du message. Si on y met les octets 0xC0 et 0x0C, on aura un pointeur (0xC0 = 11000000, les deux bits de plus fort poids à un) qui vaut 12 (0xC0 0x0C moins les deux bits les plus significatifs = 0000000000001100 = 12). Le pointeur pointera alors sur lui-même, entrainant le logiciel DNS imprudent dans une boucle sans fin. Ça aussi, ça s'est déjà produit (CVE-2017-9345 <https://nvd.nist.gov/vuln/detail/CVE-2017-9345>).

Dernier exemple amusant avec des pointeurs, le pointeur qui va mener à un nombre infini de composants. L'attaquant (ou le programmeur maladroit) met dans le message un composant suivi d'un pointeur qui revient au début de ce composant. Si le composant était `test`, un analyseur DNS imprudent va créer le nom de domaine `test.test.test.test...` avant de tomber à court de mémoire, ou bien, dans un langage de programmation comme C, d'écrire dans une autre zone de la mémoire, ce qui peut mener à un RCE. Notez que les pointeurs ne devraient pas pointer vers un pointeur : ça n'a aucun intérêt pratique et c'est dangereux, mais le RFC 1035 ne l'interdit pas explicitement. Une alternative est de tester le nombre de fois qu'on a suivi un pointeur ou, encore mieux, de vérifier que le pointeur ne pointe qu'en avant de lui-même.

Notez que le problème de la répétition infinie d'un composant pourrait également être évité en s'assurant que le nom de domaine reste en dessous de sa taille maximale, 255 octets (section 3 du RFC).

Mais il n'y a pas que les pointeurs. Un nom de domaine doit être terminé par un octet nul, indiquant la racine. Ainsi, le nom `afnic.fr` est encodé 0x05 0x61 0x66 0x6E 0x69 0x63 ("afnic") 0x02 0x66 0x72 ("fr") 0x00 (la racine). Que se passe-t-il si l'octet nul final manque ? Certains programmes en C ont été assez imprudents pour tenter de traiter le nom de domaine avec des routines comme `strlen`, qui comptent sur un octet nul pour terminer les données (section 4 de notre RFC). Le développeur qui ne veut pas refaire des failles comme CVE-2020-25107, CVE-2020-17440, CVE-2020-24383 ou CVE-2020-27736 devrait utiliser `strnlen` ou une autre méthode sûre.

On n'a pas terminé. Lorsqu'on traite une réponse DNS, les enregistrements incluent des données. Leur longueur est donné par un champ RDLENGTH de deux octets, qui précède les données (section 5). Si un analyseur DNS ne fait pas attention, la longueur indiquée peut être supérieure à la taille du paquet. Un programme qui, bêtement, lirait RDLENGTH puis ferait un `read()` de la longueur indiquée aurait

de fortes chances de taper en dehors de la mémoire accessible (CVE-2020-25108, CVE-2020-24336 et CVE-2020-27009).

Ah, et le RFC n'en parle pas, mais l'analyse des options EDNS (RFC 6891) offre également ce genre de pièges. Une option est encodée en TLV et une longueur invalide peut mener à lire les valeurs en dehors du paquet.

Un problème du même genre est décrit dans la section 6 du RFC. Un message DNS comprend plusieurs sections (Question, Réponse, Autorité, Additionnelle) et le nombre d'enregistrements par section est indiqué au début du message DNS. Ces nombres ("*count*") peuvent également être faux et ne doivent pas être utilisés aveuglément, sinon, on aura des malheurs du genre CVE-2020-25109, CVE-2020-24340, CVE-2020-24334 ou CVE-2020-27737.

C'est en raison de tous ces risques qu'il faut tester que ses programmes résistent bien à des messages DNS mal formés. Par exemple, le serveur faisant autorité Drink <https://www.bortzmeyer.org/drink.html>, écrit en Elixir, a dans ses jeux de tests de tels messages. Regardez par exemple `drink_parsing_test.exs`. Un exemple d'un tel test, pour le problème de boucle sans fin sur des pointeurs, décrit en section 2 du RFC, est :

```
  test "pointerloop1" do # RFC 9267, section 2
    result = Drink.Parsing.parse_request(<<@id::unsigned-integer-size(16),
      @request::unsigned-integer-size(16),
      1::unsigned-integer-size(16), # QDcount
      0::unsigned-integer-size(16), # ANcount
      0::unsigned-integer-size(16), # NScount
      0::unsigned-integer-size(16), # ARcount
      0xc0::unsigned-integer-size(8), # First label of the question
      # section starts with a
      # compression pointer to itself.
      0x0c::unsigned-integer-size(8),
      @mx::unsigned-integer-size(16),
      @in_class::unsigned-integer-size(16)
      >>,
      false)
    assert result == {:error, :badDNS}
  end
```

On fabrique un paquet DNS avec le pointeur invalide, et on l'analyse. Le sous-programme `parse_request` ne doit pas tourner sans fin, et doit renvoyer une erreur.

Un test du même genre en Python, où on envoie un message DNS avec un pointeur pointant sur lui-même (et on espère que ça n'aura pas tué le serveur qui le reçoit) :

```
id = 12345
misc = 0 # opcode 0, all flags zero
data = struct.pack(">HHHHHHBB", id, misc, 1, 0, 0, 0, 0xc0, 0x0c)
s.sendto(data, sockaddr)
```