Quad9, un résolveur DNS public, et avec sécurité

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 16 novembre 2017

https://www.bortzmeyer.org/quad9.html

Aujourd'hui a été annoncé la disponibilité du résolveur DNS Quad9 <https://www.quad9.net/#/> (prononcer « quoi de neuf » en français). C'est un résolveur DNS public, mais dont l'originalité est d'être accessible de manière sécurisée, avec TLS (RFC 7858 [1]). ("*There is also an english version of this article*" <https://labs.ripe.net/Members/stephane_bortzmeyer/quad9-a-public-dns-resolver-with-s>.)

Des résolveurs DNS publics, il y en a plein. Le plus connu est Google Public DNS <https://www.bortzmeyer.org/google-dns.html> mais il en existe beaucoup d'autres, avec des politiques et des caractéristiques techniques diverses. Le fait que tant d'utilisateurs se servent aveuglément de Google Public DNS, malgré l'énorme quantité de données que Google connait déjà sur nous <https://www.shaftinc.fr/arretez-google-dns.html> est inquiétant. Mais il y aussi un problème technique, commun à la plupart des résolveurs publics <https://www.bortzmeyer.org/dns-resolveurs-publics.html> : le lien avec eux n'est pas sécurisé. Cela permet les détournements, comme vu en Turquie <https://www.bortzmeyer.org/dns-routing-hijack-turkey.html>, ainsi que la surveillance par un tiers.

Au contraire, le nouveau service Quad9 <https://www.quad9.net/>, géré par l'organisme sans but lucratif bien connu PCH, qui gère une bonne partie de l'infrastructure du DNS, Quad9, donc, permet un accès par DNS sur TLS (RFC 7858). Cela permet d'éviter l'écoute par un tiers, et cela permet d'authentifier le résolveur (je n'ai pas encore testé ce point, Quad9 ne semble pas distribuer de manière authentifiée ses clés publiques).

Question politique, notons encore que Quad9 s'engage à ne pas stocker votre adresse IP <https://www.quad9.net/#/privacy>. Et que leur résolveur est un résolveur menteur : il ne répond pas (délibérement) pour les noms de domaines considérés comme lié à des activités néfastes comme la distribution de logiciel malveillant. On peut avoir un résolveur non-menteur en utilisant d'autres adresses

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

<https://www.quad9.net/#/faq> mais on perd DNSSEC et surtout Quad9 se met à utiliser alors l'indication du réseau du client (RFC 7871), une mauvaise pratique pour la vie privée. Espérons qu'on aura bientôt une adresse pour les réponses non-menteuses, avec DNSSEC et sans l'indication du réseau du client.

Bon, passons maintenant à la pratique, sur une machine Unix. L'adresse IPv4 de Quad9, comme son nom l'indique, est 9.9.9.9. Son adresse IPv6 est 2620:fe::fe (cf. la FAQ <https://www.quad9.net/#/faq>). D'abord, un accès classique en UDP en clair :

```
% dig +nodnssec @9.9.9.9 AAAA irtf.org

; <<>> DiG 9.10.3-P4-Ubuntu <<>> +nodnssec @9.9.9.9 AAAA irtf.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11544
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;irtf.org. IN AAAA

;; ANSWER SECTION:
irtf.org. 1325 IN AAAA 2001:1900:3001:11::2c

;; Query time: 4 msec
;; SERVER: 9.9.9.9#53(9.9.9.9)
;; WHEN: Thu Nov 16 09:49:41 +08 2017
;; MSG SIZE  rcvd: 65
```

On y voit que Quad9 valide avec DNSSEC (la réponse a bien le bit AD - "*Authentic Data*").

Si le domaine est sur la liste noire de Quad9 (merci à Xavier Claude <https://framasphere.org/u/claudex> pour avoir trouvé un nom), le résolveur répond NXDOMAIN ("*No Such Domain*", ce domaine n'existe pas) :

```
%  dig @9.9.9.9  www.hjaoopoa.top

; <<>> DiG 9.10.3-P4-Debian <<>> @9.9.9.9 www.hjaoopoa.top
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 1143
;; flags: qr rd ad; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;www.hjaoopoa.top. IN A

;; Query time: 17 msec
;; SERVER: 9.9.9.9#53(9.9.9.9)
;; WHEN: Sat Nov 18 20:30:41 CET 2017
;; MSG SIZE  rcvd: 45
```

(Avec un résolveur non-menteur, on aurait eu le code de retour NOERROR et l'adresse IP `54.213.138.248`.)

Maintenant, testons la nouveauté importante de ce service, DNS sur TLS (RFC 7858). C'est du TLS donc on peut y aller avec openssl :

```
% openssl s_client -connect \[2620:fe::fe\]:853 -showcerts
depth=2 O = Digital Signature Trust Co., CN = DST Root CA X3
verify return:1
depth=1 C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3
verify return:1
depth=0 CN = dns.quad9.net
verify return:1
---
Certificate chain
 0 s:/CN=dns.quad9.net
   i:/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
-----BEGIN CERTIFICATE-----
...
1 s:/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
   i:/O=Digital Signature Trust Co./CN=DST Root CA X3
...
Server certificate
subject=/CN=dns.quad9.net
issuer=/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
---
Peer signing digest: SHA512
Server Temp Key: ECDH, P-256, 256 bits
...
New, TLSv1.2, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
...
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
...
```

On voit que Quad9 répond bien en TLS, et a un certificat Let's Encrypt.

Testons ensuite avec un client DNS, le programme `getdns_query` distribué avec getdns <`https://getdnsapi.net/`> :

```
% getdns_query @9.9.9.9 -s -l L www.afnic.fr AAAA
{
  "answer_type": GETDNS_NAMETYPE_DNS,
  "canonical_name": <bindata for lb01-1.nic.fr.>,
  "just_address_answers":
  [
    {
      "address_data": <bindata for 2001:67c:2218:30::24>,
      "address_type": <bindata of "IPv6">
    }
...
```

(Oui, `getdns_query` est très bavard.) L'option `-l L` lui dit d'utiliser DNS sur TLS.

On va d'ailleurs utiliser tshark pour vérifier qu'on est bien en TLS :

```
% tshark -n -i wlp2s0  -d tcp.port==853,ssl host 9.9.9.9
Capturing on 'wlp2s0'
1 0.000000000 31.133.136.116 → 9.9.9.9      TCP 74 37874 → 853 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_P
2 0.002518390      9.9.9.9 → 31.133.136.116 TCP 74 853 → 37874 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=
3 0.002551638 31.133.136.116 → 9.9.9.9      TCP 66 37874 → 853 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=233
4 0.002642065 31.133.136.116 → 9.9.9.9      SSL 371 Client Hello
5 0.022008585      9.9.9.9 → 31.133.136.116 TLSv1.2 1514 Server Hello
6 0.022042645 31.133.136.116 → 9.9.9.9      TCP 66 37874 → 853 [ACK] Seq=306 Ack=1449 Win=32128 Len=0 TSva
7 0.022050371      9.9.9.9 → 31.133.136.116 TLSv1.2 108 [TCP Previous segment not captured] , Ignored Unkno
8 0.022054712 31.133.136.116 → 9.9.9.9      TCP 78 [TCP Window Update] 37874 → 853 [ACK] Seq=306 Ack=1449
9 0.022667110      9.9.9.9 → 31.133.136.116 TCP 1514 [TCP Out-Of-Order] 853 → 37874 [ACK] Seq=1449 Ack=306
10 0.022679278 31.133.136.116 → 9.9.9.9      TCP 66 37874 → 853 [ACK] Seq=306 Ack=2939 Win=37888 Len=0 TSv
11 0.023537602 31.133.136.116 → 9.9.9.9      TLSv1.2 192 Client Key Exchange, Change Cipher Spec, Encrypted
12 0.037713598      9.9.9.9 → 31.133.136.116 TLSv1.2 117 Change Cipher Spec, Encrypted Handshake Message
13 0.037888417 31.133.136.116 → 9.9.9.9      TLSv1.2 225 Application Data
14 0.093441153      9.9.9.9 → 31.133.136.116 TCP 66 853 → 37874 [ACK] Seq=2990 Ack=591 Win=31232 Len=0 TSv
15 0.742375719      9.9.9.9 → 31.133.136.116 TLSv1.2 178 Application Data
...
```

Le `-d tcp.port==853,ssl` était là pour dire à tshark d'interpréter ce qui passe sur le port 853 (celui de DNS-sur-TLS) comme étant du TLS. On voit bien le dialogue TLS mais évidemment pas les questions et réponses DNS puisque tout est chiffré.

Bien, maintenant que les tests se passent bien, comment utiliser Quad9 pour la vraie résolution de noms ? On va utiliser stubby <https://dnsprivacy.org/wiki/display/DP/DNS+Privacy+Daemon+-+Stubby> pour parler à Quad9. Le fichier de configuration Stubby sera du genre :

```
listen_addresses:
  - 0::1@8053
# https://github.com/getdnsapi/getdns/issues/358

dns_transport_list:
  - GETDNS_TRANSPORT_TLS

upstream_recursive_servers:
# Quad9
   - address_data: 9.9.9.9
     tls_auth_name: "dns.quad9.net"
   - address_data: 2620:fe::fe
     tls_auth_name: "dns.quad9.net"
```

On indique à stubby d'écouter sur l'adresse locale `::1`, port 8053, et de faire suivre les requêtes en DNS sur TLS à `9.9.9.9` ou `2620:fe::fe`. On lance stubby :

```
% stubby
[12:28:10.942595] STUBBY: Read config from file /usr/local/etc/stubby/stubby.yml
[12:28:10.942842] STUBBY: Starting DAEMON....
```

Et on peut le tester, en utilisant dig pour interroger à l'adresse et au port indiqué :

```
% dig @::1 -p 8053 A www.catstuff.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> @::1 -p 8053 A www.catstuff.com
```

```
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20910
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 65535
;; QUESTION SECTION:
;www.catstuff.com. IN A

;; ANSWER SECTION:
www.catstuff.com. 600 IN A 216.157.88.24

;; Query time: 974 msec
;; SERVER: ::1#8053(::1)
;; WHEN: Thu Nov 16 20:29:26 +08 2017
;; MSG SIZE  rcvd: 77
```

Et on peut vérifier avec tshark ou tcpdump que Stubby parle bien avec Quad9, et en utilisant TLS.

Si, à ce stade, vous obtenez une réponse DNS `FORMERR` (”*FORmat ERRor*”) au lieu du `NOERROR` qu’on voit ci-dessus, c’est à cause de cette bogue `<https://github.com/getdnsapi/getdns/issues/357>` et il faut mettre à jour la bibliothèque getdns utilisée par Stubby.

Stubby a l’avantage de bien gérer TCP, notamment en réutilisant les connexions (il serait très coûteux d’établir une connexion TCP pour chaque requête DNS, surtout avec TLS par dessus). Mais il n’a pas de cache des réponses, ce qui peut être ennuyeux si on est loin de Quad9. Pour cela, le plus simple est d’ajouter un vrai résolveur, ici Unbound. On le configure ainsi :

```
server:
   interface: 127.0.0.1
   do-not-query-localhost:  no
forward-zone:
  name: "."
    forward-addr: ::1@8053
```

Avec cette configuration, Unbound va écouter sur l’adresse `127.0.0.1` (sur le port par défaut, 53, le port du DNS) et relayer les requêtes pour lesquelles il n’a pas déjà une réponse dans son cache vers Stubby (`::1`, port 8053). Interrogeons Unbound :

```
% dig @127.0.0.1 A mastodon.gougere.fr

; <<>> DiG 9.10.3-P4-Ubuntu <<>> @127.0.0.1 A mastodon.gougere.fr
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40668
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;mastodon.gougere.fr. IN A
```

```
;; ANSWER SECTION:
mastodon.gougere.fr. 600 IN A 185.167.17.10

;; Query time: 2662 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Nov 16 20:36:09 +08 2017
;; MSG SIZE  rcvd: 64
```

Unbound a une mémoire (le cache) donc si on recommance la requête aussitôt, la réponse arrivera bien plus vite et on verra le TTL (600 secondes ici) diminué.

Si vous trouvez que tout cela est bien compliqué à installer/configurer, vous pouvez vous servir d'une image Docker, faite par Juzam <https://hub.docker.com/r/juzam/docker-getdns-stubby/> (voir aussi sur Github <https://github.com/juzam/docker-getdns-stubby>).

On note qu'il n'est pas évident de trouver une adresse IPv4 facilement mémorisable comme 9.9.9.9. L'examen de DNSDB <https://www.bortzmeyer.org/dnsdb.html> montre que cette adresse a beaucoup servi avant d'arriver chez PCH, et pour des activités...que certains peuvent trouver contestables. Cette adresse, sérieusement marquée, est donc noirlistée à plusieurs endroits. Si cela ne marche pas de chez vous, essayez d'ailleurs, ou alors en IPv6. On voit bien cette imparfaite connectivité en testant avec les sondes RIPE Atlas <https://atlas.ripe.net/> et le programme atlas-reach <https://labs.ripe.net/Members/stephane_bortzmeyer/using-ripe-atlas-to-debug-network-connecti comparant Quad9 et Google Public DNS :

```
% atlas-resolve -r 200 -e 9.9.9.9 --nsid -t AAAA irtf.org
Nameserver 9.9.9.9
[ERROR: SERVFAIL] : 1 occurrences
[TIMEOUT] : 9 occurrences
[2001:1900:3001:11::2c] : 177 occurrences
Test #10205081 done at 2017-11-16T01:41:40Z

% atlas-resolve -r 200 -e 8.8.8.8 -g 10205081 --nsid -t AAAA irtf.org
Nameserver 8.8.8.8
[TIMEOUT] : 1 occurrences
[2001:1900:3001:11::2c] : 186 occurrences
Test #10205089 done at 2017-11-16T01:46:38Z
```

Et merci à Sara Dickinson pour son aide technique.

Si vous vous intéressez aux questions plus politiques, vous pouvez consulter l'article critique de sha-5_1_2 <https://www.reddit.com/r/privacy/comments/7lwt6l/psa_please_stop_using_quad9_9999_dont_make_the/> ainsi que la la réponse très détaillée du directeur de Quad9 <https://www.reddit.com/r/privacy/comments/7lwt6l/psa_please_stop_using_quad9_9999_dont_make_the/drqcap4/>.