RFC 7129 : Authenticated Denial of Existence in the DNS

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 13 février 2014

Date de publication du RFC : Février 2014

https://www.bortzmeyer.org/7129.html

Voilà une expression un peu effrayante, « la négation d'existence authentifiée ». C'est en fait un terme utilisé dans le DNS et notamment dans DNSSEC, pour indiquer qu'un résolveur DNS a pu prouver qu'un nom n'existe pas, ou bien qu'il existe mais qu'il n'a pas de données de tel type. Dans DNSSEC, cela se fait typiquement avec des enregistrements NSEC et NSEC3, dont la définition n'est pas toujours claire pour tout le monde. Ce RFC a donc pour but d'expliquer « tout ce qu'il faut savoir sur la négation d'existence authentifiée ». C'est une lecture indispensable pour les programmeurs qui vont mettre en œuvre DNSSEC, mais aussi pour les étudiants ou ingénieurs qui veulent comprendre DNSSEC y compris dans ses tréfonds les plus obscurs.

C'est que NSEC et, surtout, NSEC3 sont très compliqués, reconnait ce RFC dès sa première section. C'est même une des parties les plus compliquées de DNSSEC, qui n'en manque pourtant pas.

DNSSEC fonctionne en signant cryptographiquement les données retournées. Mais il ne signe pas les messages. Il y a deux codes de retour pour une réponse DNS lorsque le nom n'existe pas ou bien qu'il n'y a pas de données du type demandé. Dans le premier cas, le serveur DNS répond NXDOMAIN ("*No Such Domain*"), dans le second NOERROR mais avec un champ "*Answer Count*" à zéro, ce qu'on nomme une réponse NODATA. Dans les deux cas, aucun enregistrement n'est renvoyé. Comment signer le vide ? C'est pourtant indispensable, autrement un attaquant pourrait, par exemple, fabriquer une réponse prétendant qu'un enregistrement DS (délégation signée) n'existe pas, débrayant ainsi la sécurité qu'offre DNSSEC (section 6 de notre RFC).

Tous les exemples du RFC sont faits avec une petite zone d'exemple, example.org dont voici le contenu non signé :

```
example.org.        SOA ( ... )
                    NS  a.example.org.
a.example.org.      A 192.0.2.1
                    TXT "a record"
d.example.org.      A 192.0.2.1
                    TXT "d record"
```

Chargeons-la dans NSD <https://www.bortzmeyer.org/nsd.html> pour voir et pouvoir reproduire les tests (le serveur va écouter sur le port 9053). D'abord (section 2 du RFC), avec une zone non signée. Premier test, le résolveur demande les enregistrements de type `TXT` pour `a.example.org` (le fichier de configuration de dig contient l'option `+dnssec` et donc impose EDNS) :

```
% dig @::1 -p 9053 TXT a.example.org

; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsg.P2-4ubuntu1.1 <<>> @::1 -p 9053 TXT a.example.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46585
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;a.example.org. IN TXT

;; ANSWER SECTION:
a.example.org. 3600 IN TXT "a record"

;; AUTHORITY SECTION:
example.org. 3600 IN NS a.example.org.

;; ADDITIONAL SECTION:
a.example.org. 3600 IN A 192.0.2.1

;; Query time: 1 msec
;; SERVER: ::1#9053(::1)
;; WHEN: Mon Jan 20 18:46:25 CET 2014
;; MSG SIZE  rcvd: 93
```

On a bien le code de retour `NOERROR` et un seul enregistrement en réponse. On récupère aussi des informations facultatives dans les sections *"Authority"* et *"Additional"*. Si on demande par contre `b.example.org` :

```
% dig @::1 -p 9053 TXT b.example.org

; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsg.P2-4ubuntu1.1 <<>> @::1 -p 9053 TXT b.example.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 10555
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;b.example.org. IN TXT

;; AUTHORITY SECTION:
example.org. 3600 IN SOA a.example.org. root.example.org. (
2014012000 ; serial
604800     ; refresh (1 week)
86400      ; retry (1 day)
2419200    ; expire (4 weeks)
86400      ; minimum (1 day)
```

```
)

;; Query time: 0 msec
;; SERVER: ::1#9053(::1)
;; WHEN: Mon Jan 20 18:48:18 CET 2014
;; MSG SIZE  rcvd: 85
```

On a bien un `NXDOMAIN`. La section "*Authority*" nous donne le SOA de la zone.

Il y a un autre cas où la négation d'existence est nécessaire : lorsque le nom existe mais qu'il n'y a pas de données de ce type. Par exemple, si je demande une adresse IPv6 (qui n'est pas dans la zone) :

```
%  dig @::1 -p 9053 AAAA a.example.org

; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsg.P2-4ubuntu1.1 <<>> @::1 -p 9053 AAAA a.example.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55053
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1
;; WARNING: recursion requested but not available

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

;; AUTHORITY SECTION:
example.org. 3600 IN SOA a.example.org. root.example.org. (
2014012000 ; serial
604800     ; refresh (1 week)
86400      ; retry (1 day)
2419200    ; expire (4 weeks)
86400      ; minimum (1 day)
)

;; Query time: 0 msec
;; SERVER: ::1#9053(::1)
;; WHEN: Thu Jan 23 10:07:06 CET 2014
;; MSG SIZE  rcvd: 83
```

Ici, `a.example.org` a bien une adresse IPv4 dans la zone (et donc, le nom existe) mais pas d'adresse IPv6. Le code de retour est `NOERROR` (pas de problème) mais le compteur de réponses (`ANSWER:`) indique zéro. C'est ce qu'on appelle une réponse `NODATA` même si ce code n'apparait jamais dans les normes DNS.

Maintenant, passons en DNSSEC (section 3). D'abord, le rappel de quelques principes :

— Les signatures DNSSEC peuvent être calculées à la volée (lors d'une requête) ou bien pré-calculées puis mises dans la zone. Les deux modes sont possibles (DNSSEC a été soigneusement conçu pour que les serveurs de noms ne soient pas obligés de faire de la cryptographie en temps réel) mais la plupart des déploiements actuels utilisent le pré-calcul. Pourquoi ne pas avoir imposé la signature dynamique (elle aurait simplifié certaines choses) ? Parce que le matériel de l'époque de la conception de DNSSEC ne permettait pas de signer à ce rythme et, même aujourd'hui, il faut se rappeler que des serveurs travaillent fréquemment à plus de 10 000 requêtes par seconde, avec des pics plus élevés. D'autre part, la signature à la volée nécessite que la clé privée soit distribuée à tous les serveurs faisant autorité, ce qui peut être difficile et est certainement peu sûr.

— La réponse DNS n'est jamais signée, ce sont les données qui le sont. Ce qui veut dire que tous les en-têtes de la réponse DNS (comme le code de retour `NXDOMAIN`, ou comme le bit AD - "*Authentic Data*") peuvent être mensongers.
— Les alias (`CNAME`) et surtout les jokers (cf. RFC 4592 [1]) compliquent sérieusement les choses. Sans eux, le RFC 5155, par exemple, serait bien plus court! (Une bonne leçon lorsqu'on conçoit des protocoles : des fonctions qui semblent cool et utiles peuvent se payer très cher plus tard, par exemple lorsqu'on tente de sécuriser le protocole.)

En raison du premier principe, il faut pouvoir pré-calculer les réponses négatives alors que la liste des questions menant à une réponse négative est quasi-infinie. Pas question donc de générer une réponse signée pour toutes les questions théoriquement possibles (si on imposait la génération dynamique de réponses à la volée, cela serait possible). On ne peut pas non plus utiliser un enregistrement générique (ne dépendant pas du nom demandé) pour toutes les réponses négatives car il pourrait facilement être utilisé dans une attaque par rejeu (on demande `foobar.example.org`, on a une réponse négative signée, on l'enregistre et on la sert ensuite à des résolveurs qui demandent `a.example.org` pour leur faire croire que ce nom n'existe pas).

La solution adoptée par DNSSEC a été de « signer les trous ». Un enregistrement spécial définit un intervalle entre deux noms, signifiant « il n'y a rien entre ces deux noms ». Et cet enregistrement est signé et peut donc être vérifié. Au début, cet enregistrement spécial se nommait le `NXT` ("*NeXT name*") et était normalisé dans le RFC 2535, abandonné depuis. Il a été remplacé par le `NSEC` ("*Next SECure*") avec le RFC 3755.

Le `NSEC` nécessite que la zone soit triée. Pour qu'un enregistrement affirmant « il n'y a rien entre `jack.example.com` et `susie.example.com` » soit utilisable, il faut savoir quels noms tomberaient entre ces deux noms. Le choix a été d'utiliser l'ordre alphabétique (RFC 4034, section 6.1). Donc, `monica.example.co` est entre ces deux noms mais pas `bob.example.com`. Dans notre zone d'exemple `example.org`, cela veut dire qu'il y aura trois NSEC : un allant de l'apex (`example.org`, l'apex est souvent notée @) à `a.example.org`, un allant de `a.example.org` à `d.example.org` et un bouclant la chaîne, en allant de `d.example.org` à l'apex. Signons la zone pour voir :

```
[On crée une clé]
% dnssec-keygen -a RSASHA256 -f KSK -b 4096 example.org
Generating key pair.....................................................................................
Kexample.org.+008+37300

[On édite le fichier de zone pour mettre "$INCLUDE
Kexample.org.+008+37300.key"]

[On signe]
% dnssec-signzone -z example.org
dnssec-signzone: warning: example.org:3: no TTL specified; using SOA MINTTL instead
Verifying the zone using the following algorithms: RSASHA256.
Zone fully signed:
Algorithm: RSASHA256: KSKs: 1 active, 0 stand-by, 0 revoked
                      ZSKs: 0 active, 0 stand-by, 0 revoked
example.org.signed
```

Examinons le fichier `example.org.signed`, il a bien trois `NSEC` (et ils sont signés) :

```
% grep NSEC example.org.signed
example.org. 86400 IN NSEC a.example.org. NS SOA RRSIG NSEC DNSKEY
example.org. 86400 IN RRSIG NSEC 8 2 86400 20140222083601 20140123083601 37300 example.org. J9kDD...
a.example.org. 86400 IN NSEC d.example.org. A TXT RRSIG NSEC
a.example.org. 86400 IN RRSIG NSEC 8 3 86400 20140222083601 20140123083601 37300 example.org. IVc/rrR...
d.example.org. 86400 IN NSEC example.org. A TXT RRSIG NSEC
d.example.org. 86400 IN RRSIG NSEC 8 3 86400 20140222083601 20140123083601 37300 example.org. b/DHgE...
```

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc4592.txt`

(J'ai un peu triché, j'ai mis des noms absolus partout avec `named-compilezone -s full`.)

Chargeons cette zone signée et interrogeons-la pour un nom inexistant :

```
% dig @::1 -p 9053 A b.example.org

; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsg.P2-4ubuntu1.1 <<>> @::1 -p 9053 A b.example.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 16186
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 6, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;b.example.org. IN A

;; AUTHORITY SECTION:
a.example.org. 86400 IN NSEC d.example.org. A TXT RRSIG NSEC
a.example.org. 86400 IN RRSIG NSEC 8 3 86400 (
20140222083601 20140123083601 37300 example.org.
IVc/rr...
example.org. 86400 IN NSEC a.example.org. NS SOA RRSIG NSEC DNSKEY
example.org. 86400 IN RRSIG NSEC 8 2 86400 (
20140222083601 20140123083601 37300 example.org.
J9kDDpefX...
example.org. 86400 IN SOA a.example.org. root.example.org. (
2014012000 ; serial
604800     ; refresh (1 week)
86400      ; retry (1 day)
2419200    ; expire (4 weeks)
86400      ; minimum (1 day)
)
example.org. 86400 IN RRSIG SOA 8 2 86400 (
20140222083601 20140123083601 37300 example.org.
QcFJIEpCs...

;; Query time: 0 msec
;; SERVER: ::1#9053(::1)
;; WHEN: Thu Jan 23 10:45:38 CET 2014
;; MSG SIZE  rcvd: 1821
```

On obtient logiquement un `NXDOMAIN` et on a en plus le NSEC (signé) affirmant « il n'existe rien entre `a.example.org` et `d.example.org` ». Le résolveur a donc la preuve (signée) de la non-existence de `b.example.org`. Un attaquant qui enregistre cette réponse ne pourra pas l'utiliser contre, disons, `k.example.org`, puisque celui-ci n'est pas couvert par le NSEC.

Et avec un nom existant mais un type de données qui ne l'est pas :

```
% dig @::1 -p 9053 AAAA a.example.org

; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsg.P2-4ubuntu1.1 <<>> @::1 -p 9053 AAAA a.example.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15504
```

```
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 1
;; WARNING: recursion requested but not available

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

;; AUTHORITY SECTION:
example.org. 86400 IN SOA a.example.org. root.example.org. (
2014012000 ; serial
604800     ; refresh (1 week)
86400      ; retry (1 day)
2419200    ; expire (4 weeks)
86400      ; minimum (1 day)
)
example.org. 86400 IN RRSIG SOA 8 2 86400 (
20140222083601 20140123083601 37300 example.org.
QcFJIE...
a.example.org. 86400 IN NSEC d.example.org. A TXT RRSIG NSEC
a.example.org. 86400 IN RRSIG NSEC 8 3 86400 (
20140222083601 20140123083601 37300 example.org.
IVc/rrRD5...

;; Query time: 0 msec
;; SERVER: ::1#9053(::1)
;; WHEN: Thu Jan 23 10:56:28 CET 2014
;; MSG SIZE  rcvd: 1228
```

Ls NSEC contiennent à la fin une liste des types de données présents pour le nom en partie gauche du NSEC. On a un NSEC qui dit « oui, a.example.org existe mais les seules données sont des types A, TXT et évidemment RRSIG et NSEC ». Le résolveur validant peut alors s'assurer que le AAAA n'existe pas.

Bon, tout cela est très joli mais les NSEC ont quand même deux inconvénients (section 3.4). Le premier est qu'ils permettent le "*zone walking*" : en testant avec un nom pris au hasard, on récupére le nom existant suivant. On teste alors ce nom avec un type inexistant et on a le nom encore suivant. Et, en itérant, on a toute la zone. À la main (mais cela s'automatise facilement) :

```
% dig @::1 -p 9053 A $RANDOM.example.org
...
example.org. 86400 IN NSEC a.example.org.
[Donc, le nom suivant est a.example.org. On continue avec un type de
données rare.]

%  dig @::1 -p 9053 NAPTR a.example.org
...
a.example.org. 86400 IN NSEC d.example.org. ...
[On a le nom suivant, d.example.org, plus qu'à continuer]
```

Ce "*zone walking*" permet d'accéder au contenu entier d'une zone, même si le transfert de zones <https://www.bortzmeyer.org/recuperer-zone-dns.html> est fermé. Cela n'est pas gênant pour certaines zones (zones très structurées comme celles sous ip6.arpa ou bien petites zones au contenu évident, l'apex et www) mais inacceptable pour d'autres. Il existe des techniques astucieuses pour le limiter (cf. RFC 4470) mais la solution la plus répandue est le NSEC3, présenté plus loin.

Le second inconvénient des NSEC ne concerne que les grandes zones composées essentiellement de délégations, typiquement les TLD. Chaque enregistrement va nécessiter un NSEC et sa signature,

augmentant considérablement la taille de la zone, même quand elle ne comporte que peu de délégations sécurisées. Cela pourrait faire hésiter à adopter DNSSEC.

La solution à ces deux inconvénients est NSEC3, normalisé dans le RFC 5155. Mais, avant de voir NSEC3, voyons les autres solutions qui avaient été envisagées puis abandonnées (section 4 du RFC). Il y avait le NO, où la chaîne qui faisait le tour de la zone n'était pas fondée sur les noms, comme dans NSEC mais sur un condensat des noms, ne permettant pas de remonter au nom. (NSEC3 a plus tard repris cette idée.) Sur le moment, le danger du "*zone walking*" n'était pas considéré comme très grave, et NO était vite tombé dans l'oubli. Le problème que posait le "*zone walking*" a resurgi par la suite, menant à la conception de NSEC2, qui utilisait également des condensats des noms protégés. NSEC2 travaillait main dans la main avec un nouvel enregistrement, EXIST, qui servait à prouver l'existence d'un nom sans avoir besoin d'indiquer des données. C'était utile en cas d'utilisation des jokers (RFC 4592). Prouver que `b.example.org` n'existe pas est une chose, mais il faut aussi prouver qu'il n'y avait pas de joker qui aurait pu l'engendrer. C'est lors du travail sur NSEC2 qu'on a compris que les jokers étaient très compliqués et très mal compris.

Par contre, NSEC2 n'avait pas de système d'"*opt-out*" qui aurait pu résoudre le second inconvénient de NSEC, l'augmentation de la taille de la zone pour un TLD. D'où le développement de DNSNR, un enregistrement expérimental qui avait été le premier à prévoir une option pour ne faire aller la chaîne que d'un nom signé à un autre nom signé (alors que la chaîne NSEC va d'un nom à un nom). DNSNR affaiblit un peu la sécurité (on ne peut plus prouver la non-existence d'un nom qui n'est pas signé) mais d'une manière cohérente avec la philosophie de DNSSEC (où la signature n'est pas obligatoire). Et il permettait de diminuer l'augmentation de taille de la zone, la rendant proprotionnelle au nombre de noms signés, pas au nombre de noms. NSEC3 a donc également repris ce concept.

Il est donc temps de passer à NSEC3 (section 5 de notre RFC). NSEC3 ajoute la condensation des noms et l'"*opt-out*", mais au prix d'une certaine complexité, qui fait qu'encore de nos jours, peu de gens arrivent à déboguer un problème NSEC3. C'est entre autre pour cela que NSEC3 ne remplace pas NSEC : les deux restent définis (par exemple, la racine est signée avec NSEC, le "*zone walking*" n'étant pas un problème, puisque les données sont publiques ; la taille n'en est pas un non plus, la majorité des TLD étant signés, l'"*opt-out*" ne ferait pas gagner grand'chose).

Donc, le principe de NSEC3 : lorsqu'on signe, on met un enregistrement NSEC3 pour le condensat du nom, pointant vers le condensat suivant. La chaîne qui couvre tous les trous est donc une chaîne de condensats et pas de noms. Lorsqu'une question arrive, et que la réponse est négative, le serveur faisant autorité doit condenser le nom pour savoir quel enregistrement NSEC3 renvoyer (donc, NSEC3 est le seul cas dans DNSSEC où un serveur faisant autorité doit faire de la crypto à chaque requête, mais il est vrai que ce n'est qu'une condensation, opération typiquement très rapide). Pour éviter qu'un attaquant n'utilise des tables pré-calculées, stockant les condensats de noms possibles, on ajoute un sel et on peut faire plus d'une itération si on veut.

Pour tester NSEC3, signons cette fois notre zone avec NSEC3. Il va falloir ajouter deux arguments, `-3` pour indiquer le sel et `-H` pour le nombre d'itérations supplémentaires à faire (`-A` permettrait d'activer l'"*opt-out*", ce que je ne fais pas ici). Les exemples du RFC utilisent un sel amusant, DEAD. C'est une mauvaise idée que de choisir un sel comme cela, car ces mots amusants sont en nombre limité et des méchants ont peut-être déjà généré des tables de condensats utilisant ces sels. Je prends donc un sel aléatoire (avec `$RANDOM` et `sha256sum` ci-dessous). Par contre, je garde les autres options du RFC, 2 pour le nombre d'itérations supplémentaires, pas d'"*opt-out*" et SHA-1 comme algorithme :

```
% dnssec-signzone -3 $(echo $RANDOM$RANDOM | sha256sum | cut -d' ' -f1) -H 2 \
                 -z example.org
dnssec-signzone: warning: example.org:3: no TTL specified; using SOA MINTTL instead
Verifying the zone using the following algorithms: RSASHA256.
Zone fully signed:
Algorithm: RSASHA256: KSKs: 1 active, 0 stand-by, 0 revoked
                      ZSKs: 0 active, 0 stand-by, 0 revoked
example.org.signed
```

Et, lorsque je charge cette zone et que je demande un nom non-existant, je récupère des NSEC3 au lieu des NSEC :

```
% dig @::1 -p 9053 A b.example.org

; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsg.P2-4ubuntu1.1 <<>> @::1 -p 9053 A b.example.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 7591
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 8, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;b.example.org. IN A

;; AUTHORITY SECTION:
ui6pc9ajfb1e6ge0grul67qnckig9bck.example.org. 86400 IN NSEC3 1 0 2 DD438FBA32EC3FFA4B1849EF2F41F64A83A17D220
ASPD8T7IP6MGQ09OPQQP3KMH9D7VVODA
A TXT RRSIG )
ui6pc9ajfb1e6ge0grul67qnckig9bck.example.org. 86400 IN RRSIG NSEC3 8 3 86400 (
20140223065356 20140124065356 37300 example.org.
PIb9Y8T...
l6m3op8qm1vr3t47jnm6dbl6s4qm2bl8.example.org. 86400 IN NSEC3 1 0 2 DD438FBA32EC3FFA4B1849EF2F41F64A83A17D220
UI6PC9AJFB1E6GE0GRUL67QNCKIG9BCK
NS SOA RRSIG DNSKEY NSEC3PARAM )
l6m3op8qm1vr3t47jnm6dbl6s4qm2bl8.example.org. 86400 IN RRSIG NSEC3 8 3 86400 (
20140223065356 20140124065356 37300 example.org.
WKf8OkW...
aspd8t7ip6mgq09opqqp3kmh9d7vvoda.example.org. 86400 IN NSEC3 1 0 2 DD438FBA32EC3FFA4B1849EF2F41F64A83A17D220
L6M3OP8QM1VR3T47JNM6DBL6S4QM2BL8
A TXT RRSIG )
aspd8t7ip6mgq09opqqp3kmh9d7vvoda.example.org. 86400 IN RRSIG NSEC3 8 3 86400 (
20140223065356 20140124065356 37300 example.org.
YrR7/0vxWT...

[SOA omis...]

;; Query time: 0 msec
;; SERVER: ::1#9053(::1)
;; WHEN: Fri Jan 24 08:57:43 CET 2014
;; MSG SIZE  rcvd: 2639
```

À quel nom correspondent ces condensats ? On va utiliser le programme nsec3hash livré avec BIND. On prend les paramètres de condensation dans l'enregistrement NSEC3PARAM (cet enregistrement est nécessaire car tous les serveurs faisant autorité pour une zone, pas seulement le ou les maîtres, doivent calculer ces condensats, ce qui implique de disposer de ces informations ; si le DNS avait été prévu avec DNSSEC et NSEC3 dès le début, on les aurait probablement mises dans le SOA) :

```
% dig +short @::1 -p 9053 NSEC3PARAM example.org
1 0 2 DD438FBA32EC3FFA4B1849EF2F41F64A83A17D220D22F57BC9903300A861BFE9
...
```

On voit donc le nombre d'itérations à faire et la valeur du sel (ce n'est pas le même sel que dans les exemples du RFC, ne vous étonnez donc pas si les valeurs sont différentes ; celles du RFC sont résumées dans l'annexe C). Testons avec l'apex :

```
% nsec3hash DD438FBA32EC3FFA4B1849EF2F41F64A83A17D220D22F57BC9903300A861BFE9 1 2 example.org
L6M3OP8QM1VR3T47JNM6DBL6S4QM2BL8 (salt=DD438FBA32EC3FFA4B1849EF2F41F64A83A17D220D22F57BC9903300A861BFE9, hash=1,
```

Et pareil pour les autres noms qu'on connait. On peut donc trouver le NSEC3 correspondant à chacun. Ici, pour l'apex, c'était :

```
l6m3op8qm1vr3t47jnm6dbl6s4qm2bl8.example.org. 86400 IN NSEC3 1 0 2 DD438FBA32EC3FFA4B1849EF2F41F64A83A17D220D22F
UI6PC9AJFB1E6GE0GRUL67QNCKIG9BCK
NS SOA RRSIG DNSKEY NSEC3PARAM )
```

Avec à la fin la liste des types qui existent pour ce nom. Et avec dans la valeur le condensat suivant, UI6PC9AJFB1E6GE0GRUL67QNCKIG9BCK.

Ici, on n'a pas utilisé l'"*opt-out*". Il n'aurait pas de sens pour cette zone car l'"*opt-out*" ne dispense de signatures et de preuves de non-existence que les délégations ou les non-terminaux vides (les domaines qui n'ont pas d'enregistrement et qui n'existent que parce qu'ils ont des sous-domaines). Mais, dans un gros TLD, l'"*opt-out*" peut aider à garder la taille des zones à des valeurs raisonnables. Regardons la zone .fr :

```
% dig A nexistesurementpasmaisquisait.fr
...
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 38993
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 8, ADDITIONAL: 1
...
9GVSHUTCOU5FH6SKL75VBHIJQ6KLB8HR.fr. 5346 IN NSEC3 1 1 1 BE174A99 9GVTDB9F17RKAOJKIK7GIH3EHS89GGJ6 NS SOA TXT NA
...
```

Le deuxième chiffre, qui est à 1 et plus à 0 comme dans l'exemple précédent, montre que .fr utilise l'"*opt-out*". Les enregistrements NSEC3 ne couvrent donc que les délégations signées (et qui ont donc un enregistrement DS). Bref, ceux qui ne voulaient pas utiliser DNSSEC ne sont pas protégés par DNSSEC, ce qui est finalement plus logique que ce qui arrive avec NSEC.

NSEC3 peut encore peut-être paraître simple comme cela mais la norme est très complexe et on y découvre encore des cas bizarres, pas ou mal prévus comme l'erratum 3441 <http://www.rfc-editor.org/errata_search.php?eid=3441>.

Maintenant, le problème difficile, les jokers (section 5.3). Ajoutons dans notre zone d'exemple :

```
*.example.org.      TXT "wildcard record"
```

Si un client DNS demande le TXT de z.example.org, il aura une réponse (il aurait eu un NXDOMAIN avec la zone précédente). Revenons aux enregistrements NSEC pour un moment :

```
% dig -p 9053 @::1 TXT z.example.org

; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsg.P2-4ubuntu1.1 <<>> -p 9053 @::1 TXT z.example.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49955
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;z.example.org. IN TXT

;; ANSWER SECTION:
z.example.org. 86400 IN TXT "wildcard record"
z.example.org. 86400 IN RRSIG TXT 8 2 86400 (
20140227062210 20140128062210 37300 example.org.
G8THJnlRf...

;; AUTHORITY SECTION:
d.example.org. 86400 IN NSEC example.org. A TXT RRSIG NSEC
d.example.org. 86400 IN RRSIG NSEC 8 3 86400 (
20140227062210 20140128062210 37300 example.org.
JoewWY...

;; Query time: 0 msec
;; SERVER: ::1#9053(::1)
;; WHEN: Tue Jan 28 08:23:49 CET 2014
;; MSG SIZE  rcvd: 1215
```

On reçoit une réponse. Comment savoir qu'elle vient d'un joker ? C'est grâce à un champ discret de la signature, dont je n'ai pas encore parlé, le champ "*Labels*" (RFC 4034, section 3.1.3), le troisième du `RRSIG`. Il vaut 3 pour `d.example.org`, ce qui est normal car ce nom comporte trois composants. Mais il ne vaut que 2 pour `z.example.org` qui a aussi trois composants, car cet enregistrement dans la réponse a été dynamiquement généré grâce au joker. Ainsi, le résolveur DNS validant a toutes les informations nécessaires pour vérifier la signature de `z.example.org`. Comme on peut le vérifier en lisant le fichier de zone, cette signature n'a pas été générée dynamiquement, c'est celle du joker, `*.example.org`.

L'enregistrement `NSEC` dans la réponse ci-dessus empêche les attaques où le méchant essaierait de faire croire que, par exemple, `a.example.org` a aussi l'enregistrement `TXT` du joker. `a.example.org` ayant son propre `TXT`, celui du joker n'est pas utilisé et le `NSEC` - obligatoire - permettrait de prouver cela. Ici, le `NSEC` dit qu'il n'y a pas de `TXT` entre `d.example.org` et `example.org` donc rien pour `z.example.org`, achevant de prouver que la réponse vient d'un joker.

Et avec `NSEC3` ? Pareil :

```
% dig -p 9053 @::1 TXT z.example.org

; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsg.P2-4ubuntu1.1 <<>> -p 9053 @::1 TXT z.example.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10172
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 1
;; WARNING: recursion requested but not available
```

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;z.example.org. IN TXT

;; ANSWER SECTION:
z.example.org. 86400 IN TXT "wildcard record"
z.example.org. 86400 IN RRSIG TXT 8 2 86400 (
20140223162550 20140124162550 37300 example.org.
i69nAYi...
;; AUTHORITY SECTION:
l6m3op8qm1vr3t47jnm6dbl6s4qm2bl8.example.org. 86400 IN NSEC3 1 0 2 DD438FBA32EC3FFA4B1849EF2F41F64A83A17D220D22F
UI6PC9AJFB1E6GE0GRUL67QNCKIG9BCK
NS SOA RRSIG DNSKEY NSEC3PARAM )
l6m3op8qm1vr3t47jnm6dbl6s4qm2bl8.example.org. 86400 IN RRSIG NSEC3 8 3 86400 (
20140223162550 20140124162550 37300 example.org.
ZbfNK6xD++...

;; Query time: 0 msec
;; SERVER: ::1#9053(::1)
;; WHEN: Tue Jan 28 08:15:15 CET 2014
;; MSG SIZE  rcvd: 1292
```

L'enregistrement `NSEC3` prouve qu'il n'y a pas de `TXT` pour `z.example.org`, dont le condensat, `TIIAIPBQRUI8LQOQVUOTC1RGS4VR3BA5`, tomberait entre `L6M3OP8QM1VR3T47JNM6DBL6S4QM2BL8` (l'apex) et `UI6PC9AJFB1E6GE0GRUL67QNCKIG9BCK` (`a.example.org`).

Je vous épargne (et à moi aussi), la section 5.4 sur les `CNAME` combinés aux jokers.

Plus utile (mais pas facile) est la section 5.5, sur la notion de « boîte la plus proche » (*"closest encloser"*). Une particularité de `NSEC3` est qu'on perd l'information sur la profondeur de la zone : les condensats sont dans un espace plat, quel que soit le nombre de composants dans le nom condensé. Il va donc falloir ajouter un enregistrement `NSEC3` pour la boîte la plus proche. Cette boîte (*"closest encloser"*, défini dans le RFC 4592) est le nœud existant dans la zone qui a le plus de composants correspondant à la requête. Si la requête portait sur `x.2.example.org` et qu'il n'existe rien se terminant en `2.example.org`, la boîte la plus proche est `example.org`. Autre notion, le « nom plus proche [de la boîte la plus proche] » est le nom (non existant, au contraire de la boîte la plus proche) ayant un composant de plus que la boîte la plus proche. Dans l'exemple précédent, ce serait `2.example.org`. Une preuve `NSEC3` doit parfois comprendre un enregistrement pour prouver la boîte la plus proche, et un pour prouver que le nom plus proche n'existe pas.

```
% dig @::1 -p 9053 A x.2.example.org

; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsg.P2-4ubuntu1.1 <<>> @::1 -p 9053 A x.2.example.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52476
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 8, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;x.2.example.org. IN A

;; AUTHORITY SECTION:
ui6pc9ajfb1e6ge0grul67qnckig9bck.example.org. 86400 IN NSEC3 1 0 2 DD438FBA32EC3FFA4B1849EF2F41F64A83A17D220D22F
```

```
ASPD8T7IP6MGQ09OPQQP3KMH9D7VVODA
A TXT RRSIG )
ui6pc9ajfb1e6ge0grul67qnckig9bck.example.org. 86400 IN RRSIG NSEC3 8 3 86400 (
20140227063446 20140128063446 37300 example.org.
eqpl3Vp...
...
l6m3op8qm1vr3t47jnm6dbl6s4qm2bl8.example.org. 86400 IN NSEC3 1 0 2 DD438FBA32EC3FFA4B1849EF2F41F64A83A17D220
UI6PC9AJFB1E6GE0GRUL67QNCKIG9BCK
NS SOA RRSIG DNSKEY NSEC3PARAM )
l6m3op8qm1vr3t47jnm6dbl6s4qm2bl8.example.org. 86400 IN RRSIG NSEC3 8 3 86400 (
20140227063446 20140128063446 37300 example.org.
GaK0HrQ...
je5k991emd1g7f9p37a3ajr96jrl0fvi.example.org. 86400 IN NSEC3 1 0 2 DD438FBA32EC3FFA4B1849EF2F41F64A83A17D220
L6M3OP8QM1VR3T47JNM6DBL6S4QM2BL8
TXT RRSIG )
je5k991emd1g7f9p37a3ajr96jrl0fvi.example.org. 86400 IN RRSIG NSEC3 8 3 86400 (
20140227063446 20140128063446 37300 example.org.
U5zFverPJT...

;; Query time: 0 msec
;; SERVER: ::1#9053(::1)
;; WHEN: Tue Feb 04 18:10:08 CET 2014
;; MSG SIZE  rcvd: 2641
```

Si vous voulez suivre les explications qui suivent, vous devez calculer les condensats des noms. Les voici déjà calculés (avec mon sel ; avec le sel du RFC, voir l'annexe C qui résume les valeurs) :

```
Noms dans la zone :
L6M3OP8QM1VR3T47JNM6DBL6S4QM2BL8 : apex
UI6PC9AJFB1E6GE0GRUL67QNCKIG9BCK : a.example.org
ASPD8T7IP6MGQ09OPQQP3KMH9D7VVODA : d.example.org
JE5K991EMD1G7F9P37A3AJR96JRL0FVI : *.example.org

Noms dans la requête :
C9PVNA4KK7P5CH0L8C5RFDDI377RMJTO : x.2.example.org
4TPLI2PU4PVISAQ3OK4BNE06FL1ESDKJ : 2.example.org
TIIAIPBQRUI8LQOQVUOTC1RGS4VR3BA5 : z.example.org
```

Ici, il a fallu pas moins de trois `NSEC3` pour prouver que le nom n'existait pas. Le premier prouve que `2.example.org` n'existe pas (son condensat tombe entre les deux valeurs de l'enregistrement `NSEC3`) et que donc `x.2.example.org` n'existe pas non plus, puisqu'il était sous `2.example.org`. Le second prouve que la boîte la plus proche, `example.org` existe (son condensat est la partie gauche du `NSEC3`). Et le troisième prouve qu'il y a un joker sous la boîte la plus proche (son condensat est la partie gauche du `NSEC3`) mais seulement pour les `TXT`. S'il n'y avait pas de joker dans la zone, le troisième `NSEC3` servirait à prouver cette absence (section 5.6).

Sinon, si vous aimez lire le code source, le RFC recommande celui d'Unbound, dans `validator/val_nsec3.c`...

Si votre cerveau n'est pas complètement frit après ce RFC, l'annexe A présente un mécanisme pour utiliser des `NSEC` en rendant plus difficile l'énumération de la zone, la technique de la « couverture minimale » (RFC 4470). Elle impose par contre une signature en ligne, en réponse à la question posée. En pratique, cette technique ne semble pas déployée. Une technique analogue existe pour `NSEC3` avec les « mensonges blancs » rappelés dans l'annexe B. Ils sont mis en œuvre dans le serveur de noms Phreebird <`http://dankaminsky.com/phreebird/`>.