Architecture DNS sécurisée

Guillaume Valadon et Yves-Alexis Perez

ANSSI 51 bd. de La Tour-Maubourg 75700 Paris Cedex 07 France

Résumé Le protocole DNS est primordial au bon fonctionnement de l'Internet. C'est l'un des éléments clés des communications actuelles. Depuis sa création en 1983 par la RFC883, le protocole n'a cessé d'évoluer afin de palier ses différentes limitations. L'objectif de cet article est double. Tout d'abord, il fait le point sur les problèmes et faiblesses connus de l'architecture DNS comme l'empoisonnement de cache, la corruption de trafic. Dans un second temps, il présente différentes solutions permettant de s'en prémunir. Afin d'illustrer ces solutions, des exemples concrets de configuration sont donnés.

Plus spécifiquement, nous cherchons ici à décrire différentes possibilités pour sécuriser les enregistrements de bout en bout entre un serveur DNS autoritaire et le client final, protéger les échanges en confidentialité et finalement assurer la disponibilité de l'architecture DNS.

Cet article est notamment l'occasion de présenter les avantages et les inconvénients des protocoles DNSSEC et DNSCurve. L'article se focalise sur le protocole DNS mais pas sur d'autres mécanismes de protection comme chroot, la descente de privilège ou bien encore les vues.

Deux solutions originales (l'une pour la protection du dernier lien, l'autre pour un système de résolution de nom dans un segment intranet maîtrisé) sont enfin présentées à titre d'expérimentation.

Mots-clés: DNSSEC, DNSCurve, TSIG

Introduction

Le protocole DNS est primordial au bon fonctionnement de l'Internet. C'est l'un des éléments clés des communications actuelles. Depuis sa création en 1983 par la RFC883 [36], le protocole n'a cessé d'évoluer afin de pallier ses différentes limitations. L'objectif de cet article est double. Tout d'abord, il fait le point sur les problèmes et faiblesses connus de l'architecture DNS comme l'empoisonnement de cache et la corruption de trafic. Ensuite, il présente différentes solutions permettant de s'en prémunir. Afin d'illustrer ces solutions, des exemples concrets de configuration sont donnés.

Plus spécifiquement, nous cherchons ici à décrire différentes possibilités pour sécuriser les enregistrements de bout en bout entre un serveur DNS autoritaire et le client final, protéger les échanges en confidentialité et finalement assurer la disponibilité de l'architecture DNS.

Cet article est notamment l'occasion de présenter les avantages et les inconvénients des protocoles DNSSEC et DNSCurve. Il se focalise sur le protocole DNS et ne prend pas en compte les aspects systèmes des implantations.

Dans la première section, une description d'une résolution DNS et de l'architecture DNS est donnée. La seconde section présente ensuite les faiblesses associées. Les troisième et quatrième sections décrivent les solutions que l'on peut déployer dès aujourd'hui pour se protéger des attaques connues. Enfin, une solution originale de résolution de nom basée sur LDAP est présentée à titre d'expérimentation.

1 Le DNS

Les exemples présentés par la suite ne prétendent pas être exhaustifs mais cherchent à être les plus représentatifs de la majorité des usages. En particulier, on s'intéresse principalement à la résolution des noms en adresse IP (enregistrements A et AAAA) et non aux autres enregistrements (MX, CNAME, ou NS).

1.1 Le DNS côté client

On appelle *résolution DNS* l'action qui consiste à trouver l'adresse IP associée à un nom en utilisant le protocole DNS. Dans le cas général, la résolution DNS permet à un client de récupérer un enregistrement DNS : c'est le nom de la donnée DNS élémentaire qui associe un nom à une adresse IP et une adresse IP à un nom.

Afin de bien cerner les différentes communications et entités intervenant dans l'architecture DNS, nous prenons l'exemple d'un client cherchant à connaître l'adresse IPv4 de `www.example.net`. En plus du client, cette résolution fait intervenir :

1. un serveur DNS cache : c'est le serveur DNS du fournisseur d'accès du client. Il se charge de répondre à ses requêtes en interrogeant des serveurs DNS autoritaires.
2. un serveur DNS racine : il possède la zone racine qui contient les adresses IP de serveurs DNS de tous les domaines de premiers niveaux (`net`, `com`, ou `fr`).
3. les serveurs DNS des zones `net` et `example.net`

On dit que ce sont des serveurs autoritaires pour les zones racine, `net` et `example.net` car ils en assurent la gestion.

Côté client, deux éléments sont importants : l'adresse du serveur DNS cache et un résolveur DNS [1]. Ce résolveur est habituellement intégré dans la `libc`. Les logiciels demandent une résolution DNS à l'aide des fonctions `gethostbyname()` et

1. Il est appelé résolveur stub (bout d'un stylo, en anglais) car il ne parle qu'à des serveurs DNS cache.

`getaddrinfo()`. Il arrive parfois que des logiciels embarquent leur propre résolveur DNS pour des questions d'efficacité comme le fait qmail [9]. C'est une pratique marginale qui peut poser problème si le resolveur ad-hoc n'est pas assez robuste et ne traite pas correctement tous les paquets DNS.

Afin de trouver l'adresse IP associée à `www.example.net`, le résolveur du client va donc envoyer un message DNS à destination de son serveur cache lui demandant de renvoyer l'information. On considère ici que le serveur cache est vide et qu'il connait les adresses IP des serveurs racine. Il va donc demander au serveur racine s'il connait l'adresse IP associée à `www.example.net`. Le serveur racine n'est pas autoritaire pour la zone `example.net` mais va cependant indiquer au serveur cache la liste des serveurs autoritaires pour la zone `net` ainsi que leurs adresses IP.

Le serveur cache va stocker cette information puis interroger l'un des serveurs autoritaires pour `net` en lui demandant s'il connait l'adresse IP associée à `www.example.net`. Ce serveur de la zone `net` n'étant pas autoritaire pour la zone `example.net` répond avec le nom et l'adresse IP du serveur DNS autoritaire pour `example.net`.

Fort de cette nouvelle information en cache, le serveur cache va interroger le serveur autoritaire de la zone `example.net` qui va lui fournir l'enregistrement DNS correspondant à l'adresse IPv4 de `www.example.net`. À la réception de cette réponse, le serveur cache va l'enregistrer dans sa base interne et la transmettre au client. Le client possède maintenant l'adresse de `www.example.net` et peut désormais tenter de se connecter au serveur web.

L'exemple 13 reproduit à l'aide de la commande `dig` l'ensemble des requêtes DNS envoyées par le serveur DNS cache afin d'apprendre l'adresse IP de `www.example.net`. L'option @ est successivement utilisée pour envoyer la requête à des serveurs DNS possédant une information plus spécifique.

```
                    # La sortie de dig a ete tronquee. #

$ dig @m.root-servers.net. +norecurse www.example.net. A
net. NS a.gtld-servers.net
a.gtld-servers.net. A 192.5.6.30

$ dig @192.5.6.30 +norecurse www.example.net. A
example.net. NS a.iana-servers.net.
a.iana-servers.net. A 199.43.132.53

$dig @199.43.132.53 +norecurse www.example.net. A
www.example.net. A 192.0.32.10
```

Listing 1.1. Résolution DNS complète à l'aide de la commande dig

1.2 Le DNS côté administrateur

L'administrateur d'une zone autoritaire doit quant à lui travailler sur différents éléments. A minima, il doit écrire le fichier de zone et gérer un serveur DNS autoritaire. Pour plus de redondance, des serveurs DNS esclaves sont très souvent déployés. Il s'agit de serveurs possédant une copie exacte du fichier de zone qui peuvent ainsi décharger le serveur maître et assurer la disponibilité du service en cas de panne du maître. Pour obtenir un service fiable et redondant, les serveurs esclaves sont présents sur des réseaux différents du maître.

Depuis plusieurs années, de gros serveurs DNS autoritaires, comme ceux de la racine, sont distribués à travers l'Internet grâce au routage *anycast* [2]. Différentes instances d'un serveur racine sont présentes dans des lieux différents mais répondent pour la même adresse. Ainsi, un fournisseur d'accès japonais recevra une réponse de l'instance de Tokyo lorsqu'il interrogera le serveur racine M. De même, un fournisseur d'accès français recevra une réponse de l'instance de Paris. Ce procédé permet de réduire les délais et la charge sur les serveurs. En Mars 2007, il a permis de réduire les effets d'une attaque en déni de service sur les serveurs racine [28], certaines instances n'ayant pas vu d'augmentation significatives de trafic.

Le fichier de zone est chargé manuellement dans le serveur maître. Il est rechargé chaque fois qu'une entrée de la zone est ajoutée, supprimée ou mise à jour. Les serveurs esclaves doivent posséder une copie à jour du fichier de zone afin d'éviter des incohérences entre les réponses du maître et des esclaves. La synchronisation du fichier de zone entre le maître et les esclaves et les mises à jour dynamiques se font généralement via le protocole DNS. Il arrive également que les administrateurs décident d'employer des solutions ad-hoc pour synchroniser le fichier de zone via `rsync` et `ssh` pour plus de flexibilité.

2 Points faibles de l'infrastructure DNS

Nous présentons ici les différents vecteurs d'attaques envisageables contre l'infrastructure DNS. L'attaquant cherche à faire parvenir au client des informations fausses :

- en falsifiant l'information correcte, pour rediriger le trafic du client vers un serveur qu'il maîtrise,
- en supprimant l'information correcte pour faire croire au client qu'elle n'existe pas, et le forcer à se connecter à une autre machine.

L'attaquant peut également attaquer la disponibilité de l'architecture DNS par le biais d'attaques en déni de service.

2.1 Le serveur DNS cache

Compte tenu des communications ayant lieu lors d'une résolution DNS par un serveur cache, les vecteurs d'attaques peuvent être multiples.

Tout d'abord, si l'attaquant est présent sur le même lien que le client légitime, il peut renvoyer de fausses réponses en répondant plus rapidement que le serveur cache légitime. Cette attaque est implantée par la commande `dnsspoof` de la suite dsniff [41] et par les répondeurs de Scapy [10]. La section 4 présente différentes solutions pour se protéger de cette attaque.

Un administrateur d'un serveur de la zone racine a la possibilité de répondre aux questions posées par un serveur DNS cache. Un serveur répondant à une requête pour une zone dont il n'est pas autoritaire est appelée un DNS menteur. L'utilisation par le gouvernement chinois de la zone racine pour contrôler les accès à des sites a été mis en évidence en mars 2010 [45]. Une instance de la zone racine administrée par la Chine renvoyait des réponses pour des sites comme Twitter ou Facebook. Cette instance habituellement inaccessible hors de Chine a été rendu visible par le biais du routage `BGP`. Le mécanisme *Response Policy Zone* a récemment été ajouté au logiciel `BIND` pour faciliter la création de DNS menteurs.

Les gestionnaires des serveurs gérant les domaines de premiers niveaux peuvent décider de renvoyer une réponse même si le domaine n'existe pas. C'est ce que l'on appelle le *DNS wildcard*. De Septembre à Octobre 2003 [27], la société Verisign gérant les domaines `com` et `net` a ainsi redirigé les requêtes concernant des domaines non existants vers une adresse IP appartenant à Verisign. Les navigateurs web étaient ainsi redirigés vers un service web ad-hoc appelé *site finder*. Le protocole SMTP a lui aussi été affecté car les mails destinés à des domaines inexistants étaient renvoyés à un serveur de Verisign qui les rejetaient.

Un serveur `DNS` cache peut renvoyer de fausses réponses ou ne pas renvoyer de réponse du tout. Ces dernières années, certains fournisseurs d'accès ont délibérément renvoyé leurs utilisateurs vers des pages publicitaires lorsque le domaine n'existait pas. Un serveur cache peut également être victime d'empoisonnement. Ici, le but de l'attaquant est de mettre en cache des enregistrements illicites pointant, par exemple, sur des adresses IP sous son contrôle. Le serveur cache ayant enregistré des informations fausses, l'empoisonnement touchera tous ses clients.

En 2008, une nouvelle forme d'empoisonnement de cache est apparue : l'attaque Kaminsky [20]. Elle vise à empoisonner un serveur cache DNS pour un domaine entier et non plus sur un seul enregistrement comme dans les empoisonnements classiques. Elle exploite deux problèmes d'implantation du protocole DNS : les numéros des

requêtes prévisibles car attribués linéairement, et le fait que les ports UDP source des requêtes soient toujours les mêmes.

Les dernières versions des serveurs DNS ne sont pas sensibles à cette attaque et s'en protègent en choisissant des valeurs aléatoires pour les numéros de requêtes et les ports sources comme le préconise la RFC5452 [25].

2.2 Le serveur DNS autoritaire

En cas d'altération du fichier de zone par malveillance ou par erreur, le serveur vérifie uniquement le format mais pas l'intégrité des données. Il n'est donc pas possible de détecter des problèmes au chargement de la zone.

Dans le cas d'une synchronisation entre le maître et les esclaves, un attaquant peut :

1. lister la zone sur le maître en demandant un transfert de zone à l'aide de l'enregistrement AXFR ;
2. modifier les informations échangées entre le maître et l'esclave par le biais d'une attaque active ;
3. enregistrer passivement l'échange (effectué en clair) entre le maître et l'esclave.

Il faut toutefois noter que la protection en intégrité des transferts de zone effectués avec le protocole DNS est presque systématique dans les déploiements de serveurs DNS actuels. Nous reviendrons en détails sur ces mécanismes mettant en oeuvre l'enregistrement TSIG dans la section 3.3.

3 Les moyens de protection basés sur le protocole DNS

Dans cette section, nous présentons les différents mécanismes que l'on peut utiliser pour se prémunir des attaques et problèmes précédemment décrits. Ils s'appuient tous sur le protocole DNS et permettent l'intégrité, la confidentialité des messages et des enregistrements ou l'authenticité des interlocuteurs.

3.1 Protection de l'architecture globale avec DNSSEC

Pour pallier les problématiques de corruption et de modification des enregistrements DNS, des extensions permettant de signer les enregistrements ont été proposés par l'IETF dans la RFC4034 [4]. Elles sont appelées *DNSSEC* pour *DNS Security Extensions*. Ces extensions visent à transporter dans le protocole DNS des enregistrements RRSIG correspondant à des signatures permettant de vérifier qu'un enregistrement DNS a bien été émis par le gestionnaire de la zone correspondante.

L'exemple 6 présente une vue partielle d'un fichier de zone avec la signature associée à des enregistrements A. Le RRSIG comporte une signature pour les deux enregistrements de `a.example.net`, et possède des dates de début et de fin de validité. Il contient en plus le *keytag* (ici #24363) qui permet d'identifier la clé publique DNSKEY que l'on peut utiliser pour vérifier la signature.

```
a.example.net.  A 192.168.0.1
                A 192.168.0.2
                RRSIG A 5 3 86400 20110428115700 20110329115700 24363 \
                example.net. Qfise7Z0QLb9fKTONH4hjSkVFbYBruXO7NINu5KBk7rS \
                k8Bzv6RXUpMlmRowmyNxUwxlyZnLOeexIAIh0K7esg==
```

Listing 1.2. DNSSEC: un enregistrement RRSIG en pratique

Les clés publiques DNSKEY associées sont présentes dans les fichiers de zone ; la hiérarchie des zones DNS formant une architecture de clé dont la chaîne de confiance se résume comme suit. La clé DNSKEY de la zone racine signe l'empreinte cryptographique DS de la clé de la zone `net`. La clé DNSKEY de la zone `net` signe l'empreinte DS de la clé de la zone `example.net`. Finalement, la clé de la zone `example.net` signe l'enregistrement `www.example.net`. Avec DNSSEC, la chaîne de confiance ne peut être brisée. Si la zone `nodnssec.example`.net n'est pas signée, il n'est pas possible de protéger la zone `sub.nodnssec.example.net` avec DNSSEC.

Le déploiement de DNSSEC sur Internet est actuellement en cours. La zone racine est signée depuis Juillet 2010. En Mars 2011, 65 domaines de premiers niveaux possèdent un enregistrement DS dans la racine mais toutes n'acceptent pas des enregistrements DS pour leur sous zones. La plupart des bureaux d'enregistrement ne permettent pas encore d'ajouter des enregistrements DS mais les plus importants devraient faire le pas courant 2011.

Il convient dès maintenant de noter qu'avec DNSSEC seuls les serveurs DNS cache font de la vérification de signatures : ce sont des serveurs DNS validant. La sécurité du lien entre un serveur DNS cache et ses clients n'est donc pas traité par DNSSEC. De plus, la signature des enregistrements se fait hors ligne : aucune clé privée n'a besoin d'être présente sur les serveurs DNS autoritaires. Un administrateur souhaitant déployer DNSSEC sur une zone qu'il possède doit :

1. générer la clé DNSKEY de la zone ;
2. signer le fichier de zone avec cette clé ;
3. charger ce fichier dans le serveur de nom maître ;

4. fournir l'empreinte de la clé DS de la zone à son bureau d'enregistrement comme il le fait pour les enregistrements NS de la zone.

Le choix des algorithmes et des tailles de clés est laissé aux administrateurs. Actuellement, il est possible d'utiliser RSA/SHA-1 [19], RSA/SHA-2 [30], DSA/SHA-1 [17] et GOST [16] pour protéger ses zones. En pratique, 90% des zones sont protégées par RSA/SHA-1 [40]. Il convient de noter qu'une implantation ne pourra pas valider une zone signée par un algorithme qu'elle ne supporte pas.

Preuve de non-existence Comme nous venons de le voir, un enregistrement RRSIG permet de vérifier l'authenticité et l'existence d'un enregistrement associé. Signer les enregistrements existants n'est pourtant pas suffisant car un attaquant pourrait, en supprimant un enregistrement et sa signature, faire croire que l'enregistrement en question n'existe pas en renvoyant un message NXDOMAIN. Pour régler ce problème, DNSSEC possède un mécanisme, appelé preuve de non-existence, dont le but est de présenter une signature valide pour des enregistrements n'existant pas dans une zone. En effet, comme les zones sont signées statiquement, il n'est pas possible de signer des enregistrements NXDOMAIN pour l'intégralité des enregistrements qui n'existent pas et il faut donc procéder autrement.

Le premier mécanisme mis en place par DNSSEC s'appelle NSEC (Next SECure). Il est défini dans la RFC3845 [39] et consiste à trier les noms des enregistrements d'une zone dans un ordre dit canonique (proche de l'ordre alphabétique), et à signer les intervalles. Étant donnés les noms `a.example.net` et `d.example.net`, la non-existence de `b.example.net` et `c.example.net` sera fournie par un enregistrement NSEC ayant comme nom `a.example.net` et `d.example.net` comme donnée. La preuve de non-existence est quant à elle fournie par l'enregistrement NSEC et son RRSIG associé. La réponse a une requête DNS portant sur le nom `b.example.net` sera donc un enregistrement NSEC et un enregistrement RRISG comme l'illustre l'exemple 5.

```
a.example.net. NSEC d.example.net. A RRSIG NSEC
               RRSIG   NSEC 5 3 86400 20110428115700 20110329115700 24363 \
               example.net. s53BqShj7mY3wuJVqPmVMNUUgD6fWKaBLk4mAvt2wGq9J \
               fNzxfBBNoWGGDc82dV1sIQGTuUbw+TPczVTjigRcA==
```

Listing 1.3. DNSSEC: preuve de non-existence

Cette solution fournit une preuve de non-existence mais pose un autre problème : il est désormais trivial d'énumérer une zone car les noms apparaissent en clair dans

l'enregistrement NSEC et un serveur autoritaire répond aux requêtes portant sur des enregistrements NSEC. Le projet `DNSWALK` [5] fournit une implantation efficace d'une énumération de zone utilisant NSEC.

Pour améliorer les choses, un second mécanisme a été défini dans la RFC5155 [34] : DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. L'objectif est de fournir un nouvel enregistrement NSEC3 dont les mécanismes sous-jacent résistent à l'énumération de zone. Des condensats des noms plutôt que les noms eux-mêmes sont désormais utilisés. Afin de se prémunir contre des attaques par dictionnaire, un sel et le nombre d'itérations d'une fonction de hachage sont spécifiques à chaque zone. On peut récupérer ces informations via l'enregistrement NSEC3PARAM du nom de domaine. Le fonctionnement est similaire à NSEC si ce n'est que l'on hache les noms avant de les classer. Étant donné le sel 0x414243, une itération de l'algorithme SHA-1, l'exemple 7 donne la réponse à une requête portant sur le nom `b.example.net` sera alors un enregistrement NSEC3 incluant le haché de `b.example.net` (`22383b9ih85v0gj3u3epkot1nv6vqjkv`).

```
tvvgq44a75ecklia53o50qfcstpto6ob.example.net. NSEC3 1 0 1 \
                    414243 hp667bc9imetnnakj9mj9s8vm2dtb90e CNAME RRSIG
tvvgq44a75ecklia53o50qfcstpto6ob.example.net. RRSIG NSEC3 7 3 86400     \
                    20110428124240 20110329124240 10838 \
                    example.net. C57FKJOG8ZlyITR7bFbv38DlEMKCi2u1Bp5ZcS \
                    c6NKbyznw8cN7yTTatoeGucTz/avPiB0fKyeYRKKZar2ewAQ==
```

Listing 1.4. DNSSEC: preuve de non-existence avec NSEC3

Avec NSEC3, l'énumération de zone devient en théorie plus difficile qu'avec NSEC car il est interdit de demander à un serveur autoritaire si un enregistrement NSEC3 existe. En pratique, l'énumération de zone reste possible et repose sur la capacité de l'attaquant à calculer les associations entre noms de domaine valides et hachés. Tout d'abord, on récupère le sel et le nombre d'itérations en récupérant l'enregistrement NSEC3PARAM de la zone. On calcule ensuite un ensemble de hachés correspondant à des noms de domaine valides. Ces informations permettent désormais de commencer l'énumération de zone. On récupère un premier NSEC3 en interrogeant le serveur DNS à l'aide d'un nom inexistant ; `unknown.example.net` par exemple. Le serveur nous répond avec un premier enregistrement NSEC3 qui prouve la non-existence de `unkown.example.net`. Ensuite, il convient de trouver dans le dictionnaire un nom de domaine dont le haché se situe juste après le second nom couvert par le NSEC3 puis d'interroger le serveur avec ce nom pour récupérer un nouveau NSEC3. En répétant cette opération, il est possible d'énumérer la zone hachée. L'opération suivante consiste à retrouver hors ligne les noms associés à chaque haché. L'outil `nsec3walker` [7] met en

œuvre cette méthode d'énumération et fournit des outils permettant de *bruteforcer* les hachés pour retrouver les noms en clair. Les noms de services étant relativement communs (`www`, `mx`, `mail` ou `ftp`) calculer des hachés pour une zone standard est assez aisé.

En pratique, la seule méthode permettant actuellement de se protéger contre l'énumération NSEC et NSEC3 est d'aller à l'encontre de la philosophie de DNSSEC en signant dynamiquement une entrée NSEC ou NSEC3 portant uniquement sur le nom inexistant. Cette technique appelée NSEC narrow [2] est uniquement implanté dans le serveur `PowerDNS` [26] et le proxy `PhreeBird` [31]. Elle est définie dans la RFC4470 [48]. L'exemple 5 montre les enregistrements minimaux que l'on peut renvoyer lorsque que l'on demande le nom inexistant `b.example.net`.

```
aa.example.net. NSEC cc.example.net. A RRSIG NSEC

22383b9ih85v0gj3u3epkot1nv6vqjku.example.net.  NSEC3 1 0 1 \
          414243 22383b9ih85v0gj3u3epkot1nv6vqjkw CNAME RRSIG
```

Listing 1.5. DNSSEC: preuve de non-existence dynamique

Il faut toutefois noter que l'énumération de zone ne pose pas forcément de problème pour tout le monde, dans certains cas les données stockées dans la zone ne sont pas considérées comme privées. Il est tout de même important de prévoir un mécanisme permettant d'assurer la confidentialité de celles-ci, même si tout le monde ne l'utilise pas.

Clés et bonnes pratiques En pratique, deux types de clés protègent une zone signée :

1. Zone Signing Key (ZSK) : cette clé est utilisée pour signer les enregistrements de la zone comme dans l'exemple 6 ;
2. Key Signing Key (KSK) : c'est le condensat de cette clé qui est présent dans l'enregistrement DS correspondant à la zone signée. Elle ne signe que les RRSIG protégeant la ZSK. Cela permet le renouvellement de la ZSK sans modification de la zone du dessus. La clé KSK est habituellement plus grande que la ZSK.

Avec DNSSEC, le temps devient un élément important du protocole DNS. Avec le TTL des enregistrements, le temps était relatif aux données et à l'instant de leur récupération. Désormais, les signatures ont une durée de vie limitée et il faut les renouveler. La maintenance régulière d'une zone signée est nécessaire et critique.

Une fois signée avec DNSSEC, il est important que la zone soit valide et que les clés soient correctement distribuées. Dans le cas contraire, les vérifications de

2. ou NSEC white lies.

signature par les serveurs validant échoueront et la zone ne sera plus accessible.

La RFC4641 [32] fournit des bonnes pratiques pour un administrateur de zones. Elle indique notamment que le TTL d'un enregistrement doit être plus petit que la durée de validité des RRSIG, et supérieur à 10 minutes pour éviter que des signatures puissent expirer durant les processus de validation. Cette RFC recommande que la KSK soit changée tous les 13 mois.

Les renouvellements des clés lors d'une mise à jour ou d'une compromission doivent être les plus transparents possibles pour éviter que la zone ne soit inaccessible. Si une clé est subitement retirée de la zone et remplacée par une clé plus récente, cela peut amener à des situations délicates dans lesquels un serveur veut valider un RRSIG avec une clé qui n'existe plus. Afin d'éviter tout problème, il convient de modifier la zone progressivement. A la charge de l'administrateur, deux mécanismes de changement de ZSK clés ont été prévu :

1. la `pré-publication` de la nouvelle clé en avance. Dans un premier temps la nouvelle clé est ajoutée à la zone. Celle ci sera mise en cache avec l'ancienne clé. Dans un second temps, les signatures sont effectuées avec la nouvelle clé, et l'ancienne clé est encore disponible pour vérifier les vieilles signatures. Finalement, l'ancienne clé est retirée de la zone.
2. la `double signature`. Tout d'abord, on ajoute la nouvelle ZSK et on signe la zone avec l'ancienne et la nouvelle clé. Dans un second temps, on retire la vieille clé et les signatures associées.

Des mécanismes similaires peuvent être utilisés pour renouveler les KSK mais une interaction avec la zone du dessus est nécessaire pour publier le nouvel enregistrement DS. Dans le cas d'une révocation de clé, on peut appliquer les mêmes techniques. On peut toutefois décider de renouveler les clés brutalement en une seule étape en cassant la chaine de confiance. Effectuer ces opérations manuellement étant un exercice délicat, des outils permettent de renouveler les clés automatiquement [43,42].

Exemple de validation de signatures par un serveur cache On cherche ici à valider la signature RRSIG associée à l'enregistrement A de `www.nic.se` de la même façon qu'un serveur DNS cache validant. On considère que le processus de résolution DNS a récupéré tous les enregistrements nécessaires à la validation et que la clé publique de la zone racine est connue (keytag #19036). La validation s'effectue de proche en proche à l'aide des étapes suivantes.

1. vérification de la signature des enregistrements DNSKEY (ZSK #21639 et #34525) de la zone racine avec la clé #19036 ;
2. vérification de la signature de la délégation DS de la zone `se` à l'aide de la clé #34525 puis vérification de la relation entre les enregistrements DS et DNSKEY (#7649) de la zone `se` ;
3. vérification de la signature des enregistrements DNSKEY (ZSK #12973, #39547, et #57240) de la zone `se` à l'aide de la clé #7649 ;
4. vérification de la signature de la délégation DS de la zone `nic.se` à l'aide de la clé #57240 puis vérification de la relation entre les enregistrements DS et DNSKEY (#16696) de la zone `nic.se` ;
5. vérification de la signature de l'enregistrement DNSKEY (ZSK #36149) de la zone `nic.se` à l'aide de la clé #16696 ;
6. vérification de la signature de l'enregistrement A du nom `www.nic.se` à l'aide de la clé #36149.

Problèmes connus du protocole DNSSEC Deux autres problèmes subsistent en plus de l'énumération de zones à l'aide des enregistrements NSEC ou NSEC3 que nous avons précédemment décrite.

En raison de la taille des clés et des signatures, les paquets DNSSEC sont nettement plus gros que les paquets DNS classiques (de 10 à 30 fois). Une attaque en déni de service pourrait bénéficier de cet effet d'amplification en usurpant l'adresse source d'une victime avant d'émettre des petites requêtes DNSSEC. Les réponses DNSSEC étant habituellement grosses, le lien de la victime serait ainsi plus facilement saturé. Une preuve de concept de cet effet d'amplification est disponible dans le paquet dnssecamp [8]. Un *draft* propose d'utiliser ECDSA [23] pour signer les zones. Il n'existe pas encore d'implantation mais les clés et signature ECDSA étant plus compactes, leur utilisation devrait réduire de façon très sensible l'effet d'amplification.

Les signatures demeurant valides jusqu'à leur expiration, il convient de faire attention aux données qu'elles protègent. Si un administrateur change l'adresse IP associée à une de ses machines, un attaquant pourra par exemple rejouer les anciennes données jusqu'à l'expiration des signatures associées. Un serveur validant acceptera ses signatures. Un utilisateur légitime ira donc se connecter à l'ancienne adresse IP.

Exemple sur un serveur DNS cache validant La configuration d'un serveur DNS cache validant se fait rapidement sur un serveur `BIND` récent. Il convient tout d'abord de récupérer la clé DNSKEY de la zone racine puis de vérifier son empreinte à l'aide des commandes présentes dans le listing 10. Une fois cette première étape

effectuée, on peut modifier le fichier named.conf avec les paramètres présentés dans le listing 10. On considère que le serveur fonctionne correctement en tant que cache. Trois paramètres sont importants : demander à `BIND` de faire du DNSSEC, de valider et finalement lui fournir la clé DNSKEY de la zone racine.

```
$ dig +dnssec . DNSKEY |grep 257 > trusted-key.key
$ wget https://data.iana.org/root-anchors/root-anchors.xml

$ DIGEST=`cat root-anchors.xml |awk -F '[<>]' '{if ($2 == "Digest") print $3
    ;}'`

$ for DS in $(dnssec-dsfromkey -f trusted-key.key . |cut -d " " -f 7- |sed 's/
    //g')
do
  if [ $DS == $DIGEST ]; then echo "Empreinte OK"; fi;
done;
```

Listing 1.6. DNSSEC & BIND: récupération et vérification de la clé de la zone racine

```
options {
        // DNSSEC
    dnssec-enable yes; // le serveur repond aux requetes DNSSEC
    dnssec-validation yes; // le serveur verifie les signatures
};

trusted-keys {
  "." 257 3 8 "cle DNSKEY contenue dans trusted-key.key sans espace";
};
```

Listing 1.7. DNSSEC & BIND: serveur cache validant et named.conf

Une fois le serveur redémarré, il est possible de faire une requête DNSSEC à l'aide de la commande `dig` et du paramètre `+dnssec`. Lorsqu'un enregistrement a été vérifié par le serveur cache validant, le drapeau AD (pour *Authentic Data*) est présent dans la réponse comme on peut le voir dans l'exemple 12. Il également possible de vérifier les signatures localement à l'aide de l'option *+sigchase* de la commande `dig`. Le fichier trusted-key.key contenant la clé publique de la zone racine doit être présent dans le répertoire courant pour que la commande 15 fonctionne.

```
                    # La sortie de dig a ete tronquee. #

$ dig +dnssec www.pir.org. A
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25219
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 5, ADDITIONAL: 1

;; QUESTION SECTION:
;www.pir.org.           IN   A
```

```
;; ANSWER SECTION:
www.pir.org.        252 IN   A    173.201.238.128
```

Listing 1.8. Interrogation du serveur cache validant

```
                    # La sortie de dig a ete tronquee. #

$ dig +dnssec +sigchase www.pir.org. A

; RRset to chase:
www.pir.org.            28      IN      A       173.201.238.128

;; WE HAVE MATERIAL, WE NOW DO VALIDATION
;; VERIFYING DS RRset for org. with DNSKEY:21639: success
;; OK We found DNSKEY (or more) to validate the RRset
;; Ok, find a Trusted Key in the DNSKEY RRset: 19036
;; VERIFYING DNSKEY RRset for . with DNSKEY:19036: success

;; Ok this DNSKEY is a Trusted Key, DNSSEC validation is ok: SUCCESS
```

Listing 1.9. Vérification locale de signature

Exemple de déploiement sur un serveur DNS avec BIND Les exemples ci-dessous visent à fournir aux lecteurs un point de départ simple et efficace pour tester DNSSEC. Ils ne sont pas exhaustifs et ne reflètent pas les bonnes pratiques : ils n'utilisent notamment pas de clé KSK.

De nouvelles commandes présentes dans BIND permettent de mettre en oeuvre DNSSEC. Afin de signer une zone, il convient tout d'abord de générer une clé ZSK (exemple 5). Le paramètre `-3` permet de créer une clé typée pour utiliser NSEC3. Dans l'exemple, la commande génère deux fichiers : `Kexample.net.+007+04889.key`, la clé publique, et `Kexample.net.+007+04889.private`, la clé privée. Étant donné le fichier de zone `example.net`, la clé générée permettra de signer la zone à l'aide de la commande `dnssec-signzone` (exemple 4). Les paramètres `-3` et `-H` permettent de fournir respectivement le sel et le nombre d'itérations de la fonction de hachage pour NSEC3. La zone signée est présente dans le fichier `example.net.signed`. Pour finir la mise en oeuvre de DNSSEC, il suffit d'ajouter cette zone dans le fichier `named.conf` comme on le fait habituellement pour une zone DNS classique (exemple 2). Il est désormais possible d'interroger le serveur et de vérifier la signature à l'aide de la commande `dig` (exemple 15). Lors d'un déploiement réel, il convient de passer un enregistrement DS à son bureau d'enregistrement. Le contenu de cet enregistrement se trouve dans le fichier `dsset-example.net` qui a été crée lors de la signature de la zone.

```
$ dnssec-keygen -3 -r /dev/urandom example.net.
Generating key pair.........++++++
.....................................................++++++
Kexample.net.+007+04889
```

Listing 1.10. DNSSEC & BIND: génération de la clé ZSK

```
$ dnssec-signzone -S -P -3 5353544943 -H 10 example.net
Fetching ZSK 4889/NSEC3RSASHA1 from key repository.
example.net.signed
```

Listing 1.11. DNSSEC & BIND: signature de la zone

```
zone "example.net." { type master; file "/etc/bind/example.net.signed"; };
```

Listing 1.12. DNSSEC & BIND: prise en compte de la zone signée

```
                    # La sortie de dig a ete tronquee. #

$ dig @127.0.0.1 +dnssec +norecurse +trusted-key=Kexample.net.+007+04889.key
+sigchase a.example.net.
;; RRset to chase:
a.example.net.      86400 IN A 192.168.0.1

;; WE HAVE MATERIAL, WE NOW DO VALIDATION
;; VERIFYING A RRset for a.example.net. with DNSKEY:4889: success
;; OK We found DNSKEY (or more) to validate the RRset
;; Ok, find a Trusted Key in the DNSKEY RRset: 4889
;; VERIFYING DNSKEY RRset for example.net. with DNSKEY:4889: success

;; Ok this DNSKEY is a Trusted Key, DNSSEC validation is ok: SUCCESS
```

Listing 1.13. DNSSEC & BIND: validation du déploiement avec dig

Exemple de déploiement sur un serveur DNS avec PhreeBird Le proxy `PhreeBird` [31] est actuellement la solution la plus simple pour mettre en oeuvre DNSSEC rapidement. Il se place entre un serveur DNS autoritaire classique et le reste de l'Internet. Un administrateur peut ainsi protéger sa zone sans changer son serveur DNS. Après avoir téléchargé et compilé l'archive, il faut tout d'abord générer la clé privée de la zone :

```
$ bin/phreebird -g
Generated key: dns.key.  Restart without -g.
```

Listing 1.14. DNSSEC & PhreeBird: génération de la clé

Dans un second temps, il faut faire écouter le serveur autoritaire sur un port différent du 53, par exemple 5353. Pour `BIND`, cela peut être fait à l'aide de la directive *port* du fichier named.conf. Pour protéger la zone, il suffit de lancer `PhreeBird` en lui donnant l'adresse IP et le port du vrai serveur autoritaire :

```
# phreebird -b 127.0.0.1:5353
```

Listing 1.15. DNSSEC & PhreeBird: génération de la clé

La zone est désormais protégée comme on peut le voir sur l'exemple 15.

```
                    # La sortie de dig a ete tronquee. #

$ dig @127.0.0.1 +dnssec www.example.net A
;; QUESTION SECTION:
;www.example.net.        IN   A

;; ANSWER SECTION:
www.example.net.         A    192.168.0.2
www.example.net.         RRSIG    A 7 3 86400 20110427140919 \
                 20110330140919 7217 example.net. e8VDuHmMK \
         ShAxwYjPT963UDjjltsgaYNIaE7TTDuVnJBmfk5JbG \
         dE1uyskwE1AeLHN4vAJWJRpf6uyr5F8j7YEa12hIvz \
         pJmdKlq41q4a5iYaRwWA2ItzyaTddKtwN5jKT15eaF \
         3J1sX1n1iAJRI9cCiSMO47H3k4x1KyOjJ bBw=
```

Listing 1.16. DNSSEC & PhreeBird: zone protégée

`PhreeBird` met en oeuvre le mécanisme de NSEC3 narrow pour limiter les effets de l'énumération de zone. L'exemple 17 montre le résultat de ce mécanisme lorsque l'on recherche l'adresse IP de `b.example.net` dont le haché est `fh9thtjh6bkevho0fssfca4ob1oued8d` compte tenu des paramètres NSEC3 de la zone.

```
                    # La sortie de dig a ete tronquee. #

$ dig @127.0.0.1 +dnssec b.example.net A
;; QUESTION SECTION:
;b.example.net.          IN   A

;; AUTHORITY SECTION:
fh9thtjh6bkevho0fssfca4ob1oued8c.example.net. 0 NSEC3 1 0 1 \
                 1290 FH9THTJH6BKEVHO0FSSFCA4OB1OUED8E A RRSIG
fh9thtjh6bkevho0fssfca4ob1oued8c.example.net. 0 RRSIG NSEC3 \
                 7 3 0 20110427140935 20110330140935 7217 \
         example.net. NV60LuomLCL2gKo6sGgL60SNqb8qiZ \
         /c4F6MJ/jtBkTBxJw1XcSnbx4onX7PFfjIOui8y5S3n \
         alhJ42s1uSdXJQvrX7bs+gbgpI7wNE3n4rhmw65xH7S \
         ftd0UGB5tz4YFPI6e4e1s3W/QAMgwYjFI8oA1DsWYMv \
         y55bVcP64 8u0=
```

Listing 1.17. DNSSEC & PhreeBird: NSEC3 narrow

L'enregistrement DS à fournir au bureau d'enregistrement peut être récupéré à l'aide de la commande suivante.

```
$ dig @127.0.0.1 example.net. DS
example.net.        3600    IN  DS  7217 7 1
FE4E5812432735EE87377EF6AD5217EFADF00A37
```

Listing 1.18. DNSSEC & PhreeBird: récupération du DS

3.2 Protection avec DNSCurve

Développé par Daniel J. Bernstein, DNSCurve [8] est souvent comparé à tort à DNSSEC. Même si dans les esprits, les objectifs de ces deux protocoles se rejoignent, ils sont plus complémentaires que concurrents. Comme l'indiquent ses spécifications [12], DNSCurve fournit uniquement une protection au niveau du lien entre un client et un serveur DNS, ou entre deux serveurs et ne vise pas à fournir une protection des données échangées. Un serveur cache peut toujours mentir à un client. La société OpenDNS a apporté son soutien à DNSCurve en 2010 [13] mais rien ne semble avoir été fait en un an. Il semble aujourd'hui que l'utilisation de DNSCurve soit très marginale.

Ce protocole est donc comparable aux protections fournies par les enregistrements TSIG ou TKEY. Il repose sur une unique courbe elliptique définie par son créateur ce qui lui permet d'échanger des messages plus petits que DNSSEC et donc d'éviter les problèmes d'amplification de DNSSEC.

Plus spécifiquement, DNSCurve permet d'assurer la confidentialité et l'intégrité des échanges entre clients et serveurs caches, et entre serveurs cache et serveurs autoritaires. Un attaquant ne peut plus espionner et forger des messages DNS. Pour se faire, les requêtes DNS échangées sont chiffrées avec la clé publique du serveur. Les réponses le sont avec celle de son homologue. La clé publique du client est transportée dans les requêtes. Celle du serveur est quant à elle encodée dans l'enregistrement NS associée à la zone à protéger. Il faut toutefois noter que le niveau de confiance des clés n'est pas pris en compte par DNSCurve.

Un administrateur souhaitant utiliser DNSCurve pour protéger les communications avec son serveur autoritaire doit placer la clé publique de sa zone dans la zone de niveau supérieure par le biais de son bureau d'enregistrement. Contrairement à DNSSSEC, qui utilise l'enregistrement DS, DNSCurve utilise un enregistrement NS dont la valeur est un nom DNS comportant la clé publique encodée en BASE32

préfixée par les caractères `uz5`. L'exemple 3 montre les enregistrements à ajouter via le bureau d'enregistrement pour protéger la zone `example.net`.

```
example.net. NS \
uz5t9z4dy1lu2jy8xjws61jvmvl34c40my8j79d5qh55r6x4jszn6l.example.net
```

Listing 1.19. DNSCurve: enregistrements dans la zone supérieure

Les communications sont effectuées dans deux formats. Le premier, appelé, *streamline*, est incompatible avec le DNS. Il s'agit de la concaténation d'un motif spécial, de nonces[3], du message DNS chiffré avec une clé publique, et le cas échéant de la clé publique du client. Dans le cas d'une requête, le motif est `Q6fnvWj8`. Pour une réponse, c'est `R6fnvWJ8`.

Le second format, appelé TXT, est un paquet DNS standard comprenant une seule question sur un enregistrement TXT. Son nom est similaire au contenu du format *streamline* mais est encodé au format BASE32 sous forme de nom DNS. Il est indistinguable des messages DNS classiques, et permet de passer les équipements vérifiant le format des messages UDP transmis sur le port 53. Pour éviter que la réponse soit mise en cache, son TTL est positionné à 0.

Actuellement, DNSCurve revient à faire du chiffrement opportuniste lorsqu'une clé publique est disponible dans un enregistrement NS. Contrairement à DNSSEC, il n'y a pas de chaîne de confiance. Un attaquant pourrait empêcher la protection par DNSCurve en masquant l'existence de l'enregistrement NS correspondant à la clé publique. La révocation et le changement de la clé protégeant la zone se faisant uniquement par le biais du bureau d'enregistrement, il n'est pas possible de la changer rapidement en cas de compromission ou de renouvellement opérationnel. L'utilisation de serveurs autoritaires maitres et esclaves n'est pas abordé dans les documentations de DNSCurve mais soulève de nombreuses questions : est-il préférable d'installer la même clé privée partout ? Faut-il une clé privée par serveur ?

Exemples d'utilisation de DNSCurve En mars 2011, les implantations de DNSCurve fonctionnent correctement mais elles sont quelque peu rudimentaires. Certaines ne sont pas supportées et sont amenées à évoluer. Nous ne décrirons donc pas leur installation dans ce document, et laissons le lecteur suivre les indications détaillées fournies par les implantations de ces outils.

Trois implantations disponibles sont basées sur `djbdns` [6], deux proxy DNSCurve à placer devant un serveur autoritaire ou cache (CurveDNS [37] et `forward` [15]), et

3. abbréviation anglaise de *number used once*.

un patch pour `dnscache` [14]. Une quatrième est présente dans le serveur autoritaire `gdnsd` [11] mais n'a pas été testée lors de la rédaction de cet article.

Du côté des outils, le dépôt github de Matthew Dempsky [15] fournit le programme dnsq.py écrit en Python. Elle permet d'émettre et recevoir facilement des paquets protégés par DNSCurve en fournissant la clé publique du serveur avec lequel on cherche à communiquer. L'exemple 14 montre les paquets échangés lors de l'utilisation de cette commande face au serveur forward [15]. Le format *streamline* est ici utilisé : on peut reconnaître les motifs spécifiques aux questions et réponses DNSCurve dans ce format.

```
$ ./dnsq.py A www.sstic.org 192.168.21.1
    uz5l1k6sntvj4htqfnrdhm5jl699453vhmth7bd5t2lfvcbvnpjdzs

# tcpdump port 53
[..]
 0.000000 192.168.21.8 -> 192.168.21.1 DNS Unknown operation (12)
[..]
0020  15 01 97 7f 00 35 00 6b 6b f5 51 36 66 6e 76 57   .....5.kk.Q6fnvW
0030  6a 38 da 73 3e 67 df d0 b9 d0 de 1e e5 2b 60 eb   j8.s>g.......+‘.

 0.001162 192.168.21.1 -> 192.168.21.8 DNS Unknown operation (12)
[..]
0020  15 08 00 35 97 7f 01 05 ac 70 52 36 66 6e 76 57   ...5.....pR6fnvW
0030  4a 38 88 e8 d5 96 68 2a b1 4d 74 66 16 70 15 00   J8....h*.Mtf.p..
```

Listing 1.20. DNSCurve: exemples de paquets échangés

3.3 Protection de la synchronisation maître/esclave via TSIG

TSIG est une extension au protocole DNS qui permet de rajouter une signature aux messages échangés entre les différents intervenants. La RFC2845 [47] ajoute un nouvel enregistrement aux messages DNS contenant la signature de ceux-ci.

Cette extension a été développée afin de protéger en intégrité les messages de mises à jour dynamiques des serveurs DNS. En effet, il n'est habituellement pas nécessaire de protéger les requêtes DNS des clients, mais il est plus important de protéger les mises à jour des zones par un autre serveur DNS, ou par un serveur DHCP.

L'extension contient le nom de la clé partagée, l'algorithme utilisé pour signer les données (HMAC-MD5 par défaut), l'horodatage de la signature et bien évidemment la signature elle-même.

La RFC2845 [47] ne définit comme algorithme que HMAC-MD5 mais précise que d'autres algorithmes pourront être ajoutés, et on trouve en effet les algorithmes HMAC classiques jusqu'à HMAC-SHA512.

Configuration On commence par générer une clé symétrique pour un couple {maître, esclave} :

```
$ /usr/sbin/dnssec-keygen -a HMAC-SHA512 \
    -b 192 -n HOST master-slave
Kmaster-slave.+165+54391
$ grep '^Key:' Kmaster-slave.+165+54391.private
Key: 9HrPMpbo/bSnFu2FgJVlzIKYz67Z589Z
```

Listing 1.21. Génération de la clé master-slave

On positionne ensuite la clé partagée sur les deux serveurs dans la configuration de BIND. Côté maître on autorise les serveurs identifiés par la clé correspondante à réaliser des demandes de transfert de zone. On configure aussi le serveur pour protéger les transferts eux-mêmes par TSIG.

```
include "/etc/bind/master-slave.key";

server 192.0.2.2 { keys master-slave; };

zone "example.net" {
    type master;
    allow-transfer { key master-slave; };
};
```

Listing 1.22. /etc/bind/named.conf (maître)

Côté esclave on précise la clé à utiliser lors des échanges avec le serveur maître :

```
include "/etc/bind/master-slave.key";

server 192.0.2.1 { keys "master-slave"; };

zone "example.net" {
    type slave;
    masters { 192.0.2.1; }
};
```

Listing 1.23. /etc/bind/named.conf (esclave)

On peut utiliser plusieurs identifiants de clé dans la directive `keys` sous `server` et donc configurer plusieurs clés valides pour le même serveur maître. C'est de l'autre côté qu'il sera possible d'accepter un nom de clé différent pour différentes zones (via la directive `allow-transfer`).

Le fichier de clé contient la clé symétrique elle-même avec des droits restreints.

```
key master-slave {
    algorithm hmac-sha512;
    secret 9HrPMpbo/bSnFu2FgJVlzIKYz67Z589Z;
};
```

Listing 1.24. /etc/bind/master-slave.key

4 Les moyens de protection du dernier lien

Dans cette section, on se concentre sur les moyens de protection des échanges entre un client et son serveur cache (en général celui de son fournisseur d'accès ou le serveur cache de l'entreprise). Les données présentes dans le cache sont supposées intègres et on cherche à protéger le trafic sur le lien réseau entre le client et le serveur.

4.1 Utilisation de TCP

Une première idée de protection du protocole DNS au niveau local est l'utilisation du protocole TCP pour toutes les requêtes. En effet, le mode connecté permet de se protéger de certaines attaques de type homme du milieu. Le surcoût lié au passage au TCP implique l'utilisation d'un cache sur le client, mais la simplicité de la solution la rend intéressante.

Les tests de comparaison des temps de résolution en utilisant UDP et TCP montrent que l'utilisation de ce dernier pour une ressource située en cache augmente effectivement le temps de réponse (lorsque la ressource n'est pas en cache, le temps de recherche récursive rend négligeable le surcoût).

Cette augmentation dépend de plusieurs facteurs, en particulier la taille des paquets et le support éventuel de EDNS0 (extension DNS permettant le support de paquets UDP au delà de 512 octets, décrit dans la RFC2671 [46]). Le nombre de réponses contenues dans l'enregistrement influe aussi puisqu'il peut multiplier les segments TCP et donc les allez-retours entre les correspondants.

Un facteur également très influent est le nombre de requêtes effectuées. En effet, selon le cas, `getaddrinfo()` peut demander des enregistrements A et AAAA. Par défaut sur un système utilisant une double pile il fera deux demandes et recevra donc deux réponses. Si celles-ci se retrouvent dans deux segment TCP différents cela augmente encore le nombre d'allez-retour nécessaires.

Enfin, la possibilité de garder la session TCP ouverte est indispensable pour éviter de remonter une session (incluant le *three-way handshake*) à chaque requête.

Les mesures effectuées sur différents déploiements de type Intranet avec différentes machines (en terme de puissance, de charge et de connectivité) montrent que le rapport entre les temps de réponse en TCP et UDP est très variable.

L'autre surcoût, non négligeable a priori, est celui sur la charge du serveur DNS mais là encore il assez difficile de reproduire les conditions supportées par un serveur DNS comme celui d'un fournisseur d'accès.

Enfin il est difficile de forcer une implantation à utiliser le mode TCP plutôt qu'UDP puisque ce dernier est le mode par défaut du protocole.

4.2 Utilisation de TSIG

La protection du dernier lien via TSIG ressemble beaucoup à celle de la synchronisation maître/esclave. Comme TSIG est prévu pour protéger n'importe quel type de message DNS, ce mécanisme peut être utilisé entre n'importe quels intervenants échangeant via le protocole DNS.

Lorsqu'on cherche à protéger les communications entre les clients et les serveurs, les problèmes de distribution de clés et d'intégration de celles-ci dans la configuration deviennent prépondérant (comme souvent dans les protocoles utilisant des clés symétriques). Il est bien évidemment possible d'utiliser une clé pour un serveur mais cette clé se trouve donc sur chacun des clients ce qui augmente fortement les risques de compromission de celle-ci.

On peut améliorer ceci en utilisant le protocole TKEY (section 4.3) qui permet de générer des clés symétriques (pour TSIG) selon différentes méthodes.

En dehors de TKEY, il n'existe pas de méthode standardisée de génération ou de distribution clés pour utilisation dans TSIG. Un administrateur peut donc envisager d'utiliser des solutions non basées sur DNS pour pouvoir diffuser ces clés (intégration dans un paquet de la distribution, récupération dans un annuaire de type LDAP etc.), l'important étant bien évidemment d'assurer la protection de cette diffusion.

Configuration Les résolveurs standards tels que la `glibc` ne supportent pas la protection des messages DNS via TSIG. Il faut donc utiliser un résolveur indépendant ou un serveur DNS local utilisé en mode *forward* qui permet l'utilisation de fonctions avancées. Dans notre cas l'exemple utilise `lwresd`[29] basé sur BIND.

Le serveur BIND répond automatiquement à des requêtes protégées par TSIG en utilisant la clé correspondante s'il l'a a disposition, il suffit donc de lui faire connaître la clé et de configurer les clients pour que les requêtes à destination de ce serveur utilisent bien TSIG.

On commence tout d'abord par générer une clé symétrique. Son mode de distribution n'est pas abordé ici mais c'est évidemment une question à se poser lors de tout déploiement.

```
$ /usr/sbin/dnssec-keygen -a HMAC-SHA512 \
    -b 192 -n HOST tsig-key
Ktsig-key.+165+04566
$ grep '^Key:' Ktsig-key.+165+04566.private
Key: aDclENhm/lgfvFpqFe5UZukkG07gPt78
```

Listing 1.25. Génération de la clé

On positionne la clé partagée sur le serveur dans la configuration du serveur BIND :

```
    allow-recursion { key tsig-key; }

    include "/etc/bind/tsig.key";
```

Listing 1.26. /etc/bind/named.conf

L'utilisation d'un `include` permet de mettre des droits renforcés sur le fichier contenant la clé symétrique.

```
    key tsig-key {
        algorithm hmac-sha512;
        secret aDclENhm/lgfvFpqFe5UZukkG07gPt78;
    };
```

Listing 1.27. /etc/bind/tsig.key

Côté client on installe `lwresd` et on le configure de la même façon.

```
    lwres {
        listen-on { 127.0.0.1; };
    };
    server 192.0.2.1 { keys stub-key; };

    include "/etc/bind/tsig.key";
```

Listing 1.28. /etc/bind/lwresd.conf

```
    key stub-key {
        algorithm hmac-sha512;
        secret aDclENhm/lgfvFpqFe5UZukkG07gPt78;
    };
```

Listing 1.29. /etc/bind/tsig.key

Il faut aussi penser à changer le type de résolution de la `libc` :

```
        hosts:          files lwres
```

Listing 1.30. `/etc/nsswitch.conf`

4.3 Protocole d'échange de clés TKEY

La RFC2930 [18] définit un processus de négociation/génération de clés TSIG afin d'éviter l'utilisation de clés symétriques statiques et potentiellement communes à tout un déploiement. Ce processus (TKEY) prévoit quatre mécanismes de mise en place des clés : *server assignement*, *resolver assignement*, *Diffie-Hellman* et GSS-API [33].

Les deux premières méthodes correspondent à une création de la clé TSIG par l'un des participants et à l'envoi de celle-ci (chiffrée par la clé publique de l'homologue, que le créateur doit donc posséder). Le mécanisme *client assignement* ressemble beaucoup à du TLS utilisant une clé publique RSA pour la partie *key agreement*. Sans plus de protection, on obtient une authentification d'un des deux participants (celui qui reçoit la clé symétrique) mais pas de l'autre. Selon le cas cela peut ne pas être dérangeant suivant que l'on cherche plutôt à protéger les clients ou le serveur.

La méthode *Diffie-Hellman*, comme son nom l'indique, utilise simplement un échange *Diffie-Hellman* pour générer une clé partagée à partir de valeurs fournies par les deux participants.

La RFC précise que ces trois méthodes doivent obligatoirement être authentifiées, les paquets échangés (contenant la clé symétrique chiffrée dans un cas et les paramètres publics *Diffie-Hellman* dans l'autre) doivent donc être protégés en intégrités (en utilisant TSIG par exemple), ce qui ramène potentiellement au problème initial.

La méthode `GSS-API` [35] s'affranchit complètement des besoins d'authentification puisque celle-ci est gérée au niveau de l'API elle même. La `GSS-API` (*Generic Security Service Application Programming Interface*) est un standard définissant une interface pour la fourniture de services de sécurité dont l'implantation la plus connue est *Kerberos*. Elle est utilisée par exemple pour protéger des requêtes de mises à jour entre serveurs DNS BIND et DNS Microsoft.

TKEY, malgré ses aspects intéressants, est malheureusement très peu implémenté en pratique. BIND ne supporte que les mécanismes `GSS-API` et *Diffie-Hellman* et encore, le support de ce dernier est très sommaire (en particulier il n'est pas possible de déclencher une négociation TKEY en tant que resolveur dans BIND). Les autres serveurs DNS et resolveurs ne disposent pas du tout de cette fonctionnalité.

4.4 Protection du dernier lien avec un proxy DNS local

Le nombre d'implantations des différents moyens de sécurisation étant relativement faible, il est intéressant d'envisager une protection via l'utilisation d'un relais qui ferait le lien entre clients et serveurs caches.

Ceci nécessite, suivant le cas, la présence côté client, côté serveur ou les deux, d'un démon qui encapsulerait les messages DNS en ajoutant la protection.

Il est par exemple difficile de forcer un résolveur DNS en mode TCP, il serait donc possible d'utiliser `socat` [3] ou `netcat` [22] pour encapsuler le trafic entre client et serveur dans du TCP.

Une façon de protéger le trafic est d'utiliser un tunnel TLS pour chiffrer et authentifier celui-ci. `stunnel` [44] permet de protéger un flux TCP arbitraire dans un tunnel[21]. Cette méthode peut même servir à protéger les flux d'un client Windows puisque `socat` et `stunnel` sont disponibles sur ce système.

Une autre idée serait d'utiliser DTLS (Datagram TLS) [38]. Le support de ce protocole est là encore spartiate mais il est envisageable d'utiliser un proxy léger pour encapsuler le trafic DNS UDP. OpenSSL [1] supporte (depuis la version 0.9.8) le protocole DTLS.

L'idée du proxy permet aussi d'envisager des migrations en douceur (côté serveur comme côté client) lorsque les logiciels ne supportent pas encore des mécanismes tels que DNSCurve avec des outils comme CurveDNS [37] et mdempsky [15].

L'empilement des couches et le changement éventuel de protocole induit des délais qui peuvent rapidement augmenter. Il est donc fortement recommandé d'utiliser un cache tel que `nscd` (fourni par la `glibc`) sur le client. Ceci a forcément des conséquences sur la fraîcheur des informations remontées par le système mais permet d'économiser énormément de temps pour des enregistrements fréquemment demandés.

5 Architecture de résolution dans un environnement contrôlé

Dans le cadre d'un déploiement type Intranet, l'architecture du système est beaucoup moins pyramidale que l'infrastructure DNS d'Internet, on a typiquement un à deux types de serveur : les serveurs autoritaires, gérant la ou les zones internes, et les serveurs caches, interrogés par les clients (ces deux types sont parfois même fusionnés).

Dans un système d'information interne, les contraintes ne sont pas les mêmes que sur un réseau ouvert tel qu'Internet. En particulier, les impératifs de résilience ont un lien direct avec l'activité de l'entreprise ou du service. Une façon de maintenir le service disponible est de multiplier les serveurs, mais la mise en œuvre est parfois complexe, tant au niveau de l'implantation que de l'exploitation (procédures de bascules, maintenance du système en mode dégradé etc.).

Il est possible de déployer des serveurs DNS "multi-maîtres" dont la réplication ne fait pas intervenir le protocole DNS (qui implique une relation maître/esclave) en diffusant les données des zones par différents moyens tel que `rsync` ou `scp` ou en les stockant dans des dépôts `subversion` ou `git` et en mettant à jour régulièrement les dépôts locaux (de façon manuelle ou automatique). Certains serveurs peuvent aussi stocker leurs données de zones dans un annuaire LDAP ou *Active Directory* ou bien dans une base SQL, auquel cas la gestion de la réplication se fait via le mécanisme de réplication de l'annuaire ou de la base de données.

Ce type de déploiement est faisable mais la gestion des différents services ou de la réplication peut complexifier fortement le système.

Déploiement La solution que nous nous proposons d'examiner repose sur l'utilisation d'annuaires LDAP pour faire la résolution de nom. L'annuaire ne sert pas simplement au stockage des données mais répond directement aux requêtes des clients. Ceci nécessite plusieurs choses :

- un ou plusieurs serveurs LDAP contenant les données des zones ;
- un moyen pour les clients d'interroger directement les serveurs LDAP pour demander une résolution.

Pour la partie annuaire, le schéma RFC2307 [24] permet de stocker des enregistrements de machines, et donc des résolutions de type A ou AAAA.

Pour les clients de type UNIX utilisant NSS [4], il existe des bibliothèques (`nss-ldap` et `nss-ldapd`) permettant la résolution de nom via LDAP. Celles-ci sont plus connues (et plus utilisées) pour la résolution d'identifiants et de groupes (`passwd` et `group` dans `/etc/nsswitch.conf`) mais elles peuvent aussi servir pour la résolution de noms d'hôtes. Concrètement, lorsque la machine cliente a besoin de résoudre un nom, la `libc` (au travers de NSS) interroge un démon qui contacte le serveur LDAP et garde les réponses en cache. Ceci ne fonctionne que pour la résolution de nom et non pour les enregistrements particuliers (de type MX ou TXT, qui doivent être interrogés directement et nécessitent donc un support dans l'application cliente).

Cette solution présente un certain nombre d'avantages pour un système d'information avec un fort besoin de maîtrise :

- l'utilisation d'un annuaire LDAP permet de s'insérer de façon cohérente dans un système d'information déjà existant ;
- la réplication multi-maître est bien supportée par les annuaires LDAP ce qui permet de continuer à alimenter ceux-ci en cas de crise (les solutions de repli et de redondance reposent souvent sur un changement de l'enregistrement DNS, ce qui implique que le système soit disponible et qu'il soit accessible en écriture) ;

4. *Name Service Switch.*

– l'utilisation de TCP au lieu d'UDP facilite la protection contre les attaques *man in the middle* comme vu à la section 4.1
– l'utilisation des requêtes LDAP permet d'authentifier de façon aisée les serveurs comme les clients : utilisation de LDAPS, *bind* des clients etc. ;
– l'utilisation de LDAPS permet d'assurer l'intégrité des messages (et donc des enregistrements) ainsi que la confidentialité des données en chiffrant tout le trafic entre clients et serveurs si nécessaire ;

Cette solution présente ceci dit quelques inconvénients, souvent reliés aux avantages correspondant :

– s'il n'existe pas déjà d'annuaire technique, sa mise en place n'est pas triviale ;
– l'utilisation de TCP est plus coûteuse en ressources et en trafic réseau ;
– de façon générale, un serveur LDAP est plus coûteux et plus complexe qu'un serveur DNS.

Le dernier inconvénient est à relativiser : la sécurisation du protocole DNS (telle que montrée dans cet article) est complexe et non dépourvue d'écueils. Le fait de se passer complètement du protocole DNS pour utiliser le protocole LDAP et les logiciels associés n'est pas dénué d'intérêt. La jeunesse des implantations côté client est par contre aussi à prendre en compte.

Configuration L'utilisation de LDAP pour la résolution de nom nécessite l'installation et la configuration d'un certain nombre d'outils sur les serveurs (serveur LDAP) et les clients (bibliothèque `nss-ldapd` et démon cache `nslcd`).

Côté serveur, le serveur LDAP doit incorporer le schéma de la RFC2307 [24]. On ajoute à l'annuaire les entrées correspondant aux machines :

```
dn: cn=ldap-srv.example.net,ou=Hosts,dc=example,dc=net
objectClass:top
objectClass:ipHost
objectClass:device
ipHostNumber: 192.0.2.1
ipHostNumber: 192.0.2.3
cn: ldap-srv.example.net
cn: ldap-srv

dn: cn=ldap-client.example.net,ou=Hosts,dc=example,dc=net
objectClass:top
objectClass:ipHost
objectClass:device
ipHostNumber: 192.0.2.2
cn: ldap-client.example.net
cn: ldap-client
```

Listing 1.31. `LDIF` pour l'ajout des machines dans l'annuaire

La configuration du client se passe dans le fichier `/etc/nsswitch.conf` (pour indiquer à la `libc` que la résolution passe par `ldap`), le fichier `/etc/hosts` (pour configurer en dur l'adresse du ou des serveurs LDAP) et dans le `/etc/nslcd.conf` (pour configurer l'accès à l'annuaire).

Des exemples de configuration sont donnés :

```
hosts:          files ldap dns
```

Listing 1.32. Extrait du `/etc/nsswitch.conf`

```
192.0.2.1   ldap-srv
```

Listing 1.33. Extrait du `/etc/hosts`

```
# The user and group nslcd should run as.
uid nslcd
gid nslcd

# The location at which the LDAP server(s) should be reachable.
uri ldaps://ldap-srv

# The search base that will be used for all queries.
base dc=example,dc=net

# TLS
ssl on
tls_reqcert allow
tls_cacertfile /etc/ldap/LDAP-cacert.pem
```

Listing 1.34. `/etc/nslcd.conf`

Et on peut demander une résolution sur le client :

```
$ getent ahosts ldap-srv.exemple.net
192.0.2.1 STREAM ldap-srv.example.net
192.0.2.1 DGRAM
192.0.2.1 RAW
192.0.2.3 STREAM
192.0.2.3 DGRAM
192.0.2.3 RAW
```

Listing 1.35. Résolution de nom via LDAP

Le support sur les autres systèmes d'exploitation : `nss-ldapd` est restreint aux systèmes d'exploitation de type Unix puisque NSS est une spécificité de Solaris portée ensuite sur les autres Unix mais non présente ailleurs.

Côté serveur, il est possible d'utiliser n'importe quel serveur LDAP à condition d'accorder les schémas du client et du serveur (et donc d'interroger par exemple un serveur Active Directory incluant le schéma RFC2307 [24].

Côté client, il n'existe malheureusement pas de moyen simple d'interfacer un client Windows ou MacOS X avec un serveur LDAP, ce qui limite forcément la solution à des déploiements particuliers.

Les solutions pour les environnements Windows et Mac OS X (le bien connu Active Directory pour Windows, et le moins connu Open Directory pour Mac OS X), même s'ils sont censés fournir des services d'annuaire génériques, utilisent en fait lourdement le DNS pour la résolution de nom et ne permettent pas d'utiliser directement les annuaires LDAP pour ce type d'architecture.

6 Conclusion

Le protocole DNS est un élément nécessaire au bon fonctionnement de l'Internet. Il est donc primordial de protéger l'architecture sur lequel il s'appuie ainsi que les enregistrements qu'il transporte. La pollution de cache et les DNS menteurs sont parmi les vulnérabilités les plus importantes que le protocole DNS a récemment rencontrées. Différentes solutions permettant d'assurer, à différents niveaux, l'intégrité des enregistrements ont été présentées dans cet article comme le protocole DNSSEC et TSIG.

Si certaines sont plus faciles à déployer que d'autres, il est important de souligner qu'elles changent toutes la donne quant à l'administration des serveurs DNS que l'on connaît. Bien qu'elles apportent des protections, leur mauvaise utilisation peut rapidement avoir des conséquences néfastes. À la complexité de l'administration de serveurs DNS, s'ajoute désormais la gestion de clés. C'est un point particulièrement sensible car si un problème survient sur les clés, toute la résolution DNS s'effondre. Ainsi, il n'est plus possible de valider une zone protégée par DNSSEC dont les clés ont expiré. Certaines zones ayant déployé DNSSEC depuis Juillet 2010 ont déjà souffert de ce problème.

Enfin, comme nous l'avons vu, les standards et propositions de solution ne manquent pas mais les implantations sont encore incomplètes, même pour des standards assez vieux comme TSIG et TKEY, ce qui limite fortement des déploiements efficaces. L'avenir apportera très certainement son lot d'innovations quant à la protection du dernier lien entre les clients et les serveurs DNS cache.

Références

1. OpenSSL. http://www.openssl.org/.

2. Root Server Technical Operations Assn. http://www.root-servers.org/.
3. socat - Multipurpose relay. http://www.dest-unreach.org/socat/.
4. R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034 (Proposed Standard), March 2005. Updated by RFCs 4470, 6014.
5. David Barr. The dnswalk dns database debugger. http://sourceforge.net/projects/dnswalk/.
6. D. J. Bernstein. djbdns. http://http://cr.yp.to/djbdns.html.
7. D. J. Bernstein. The nsec3walker tool. http://dnscurve.org/nsec3walker.html.
8. Daniel J. Bernstein. Dnscurve : Usable security for dns. http://www.dnscurve.org/.
9. Daniel J. Bernstein. The qmail home page. http://www.qmail.org/.
10. Philippe Biondi. Scapy : DNS Answering Machine. http://trac.secdev.org/scapy/browser/scapy/layers/dns.py.
11. Brandon L Black. gdnsd - an authoritative dns server. http://code.google.com/p/gdnsd/.
12. M. Dempsky. Dnscurve : Link-level security for the domain name system. Internet-Draft, February 2010.
13. M. Dempsky. OpenDNS adopts DNSCurve. http://blog.opendns.com/2010/02/23/opendns-dnscurve/, February 2010.
14. Matthew Dempsky. DNSCurve support to dnscache. http://shinobi.dempsky.org/~matthew/patches/djbdns-dnscurve-20090602.patch.
15. Matthew Dempsky. Tools for DNS curve implementation. http://github.com/mdempsky/dnscurve/.
16. V. Dolmatov, A. Chuprina, and I. Ustinov. Use of GOST Signature Algorithms in DNSKEY and RRSIG Resource Records for DNSSEC. RFC 5933 (Proposed Standard), July 2010.
17. D. Eastlake 3rd. DSA KEYs and SIGs in the Domain Name System (DNS). RFC 2536 (Proposed Standard), March 1999.
18. D. Eastlake 3rd. Secret Key Establishment for DNS (TKEY RR). RFC 2930 (Proposed Standard), September 2000.
19. D. Eastlake 3rd. RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS). RFC 3110 (Proposed Standard), May 2001.
20. Steve Fried. An Illustrated Guide to the Kaminsky DNS Vulnerability. http://www.unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html, August 2008.
21. German Privacy Foundation. TLS-DNS. https://www.privacyfoundation.de/wiki/SSL-DNS.
22. GNU Netcat project. netcat, network swiss army knife. http://netcat.sourceforge.net.
23. P. Hoffman and W. Wijngaards. Elliptic Curve DSA for DNSSEC. Internet-Draft, December 2010.
24. L. Howard. An Approach for Using LDAP as a Network Information Service. RFC 2307 (Experimental), March 1998.
25. A. Hubert and R. van Mook. Measures for Making DNS More Resilient against Forged Answers. RFC 5452 (Proposed Standard), January 2009.
26. Bert Hubert. PowerDNS. http://www.powerdns.com/.
27. ICANN. Verisign's Wildcard Service Deployment. http://www.icann.org/en/topics/wildcard-history.html, 2003.
28. ICANN. Root server attack on 6 February 2007. Technical report, ICANN, 2007.
29. Internet Systems Consortium. Lightweight resolver daemon. http://www.isc.org/software/bind/.
30. J. Jansen. Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC. RFC 5702 (Proposed Standard), October 2009.

31. Dan Kaminsky. PhreeBird. http://dankaminsky.com/phreebird/.
32. O. Kolkman and R. Gieben. DNSSEC Operational Practices. RFC 4641 (Informational), September 2006.
33. S. Kwan, P. Garg, J. Gilroy, L. Esibov, J. Westhead, and R. Hall. Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG). RFC 3645 (Proposed Standard), October 2003.
34. B. Laurie, G. Sisson, R. Arends, and D. Blacka. DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. RFC 5155 (Proposed Standard), March 2008.
35. J. Linn. Generic Security Service Application Program Interface Version 2, Update 1. RFC 2743 (Proposed Standard), January 2000. Updated by RFC 5554.
36. P.V. Mockapetris. Domain names : Implementation specification. RFC 883, November 1983. Obsoleted by RFCs 1034, 1035, updated by RFC 973.
37. CurveDNS Project. Curvedns. http://curvedns.on2it.net/.
38. E. Rescorla and N. Modadugu. Datagram Transport Layer Security. RFC 4347 (Proposed Standard), April 2006. Updated by RFC 5746.
39. J. Schlyter. DNS Security (DNSSEC) NextSECure (NSEC) RDATA Format. RFC 3845 (Proposed Standard), August 2004. Obsoleted by RFCs 4033, 4034, 4035.
40. SecSpider. SecSpider the DNSSEC Monitoring Project. http://secspider.cs.ucla.edu/.
41. Dug Song. dsniff. http://monkey.org/~dugsong/dsniff/.
42. The Dnssec-Tools Project. DNSSEC-TOOLS. https://www.dnssec-tools.org/.
43. The OpenDNSSEC project. OpenDNSSEC. http://www.opendnssec.org/.
44. Michal Trojnara. stunnel, multiplatform SSL proxy. http://www.stunnel.org/.
45. Iljitsch van Beijnum. China censorship leaks outside great firewall via root server. http://arstechnica.com/tech-policy/news/2010/03/china-censorship-leaks-outside-great-firewall-via-root-server.ars, March 2010.
46. P. Vixie. Extension Mechanisms for DNS (EDNS0). RFC 2671 (Proposed Standard), August 1999.
47. P. Vixie, O. Gudmundsson, D. Eastlake 3rd, and B. Wellington. Secret Key Transaction Authentication for DNS (TSIG). RFC 2845 (Proposed Standard), May 2000. Updated by RFC 3645.
48. S. Weiler and J. Ihren. Minimally Covering NSEC Records and DNSSEC On-line Signing. RFC 4470 (Proposed Standard), April 2006.