INGI2347 - Securité

Projet de sécurité

Rapport n°2

par

François Beuvens
Grégoire de Hemptinne

Groupe 6

Table des matières

Section 1

Implémentation

1.1 Avant-propos

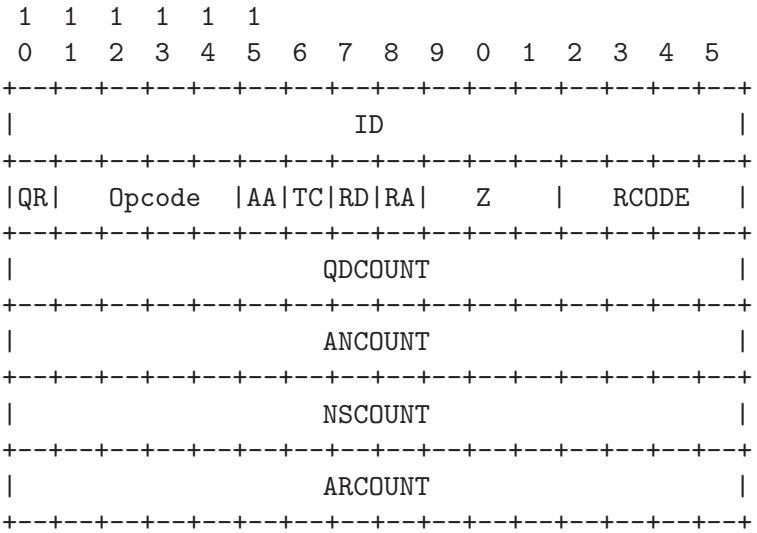
Afin de mieux comprendre l'implémentation de notre simulateur de serveur DNS, nous allons décrire dans cette section les formats des paquets échangés en détail.

1.1.1 Format général

```
+---------------------+
|        Header       |
+---------------------+
|       Question      | the question for the name server
+---------------------+
|        Answer       | RRs answering the question
+---------------------+
|      Authority      | RRs pointing toward an authority
+---------------------+
|      Additional     | RRs holding additional information
+---------------------+
```

La section `Header` est toujours présente. Une requête faite par un client possède en plus une section `Question` tandis qu'une réponse envoyée par le serveur comportera en plus du `Header` une section `Question` qui est la question envoyée par le client, et une ou plusieurs sections `Answer`, `Authority` et `Additional`.

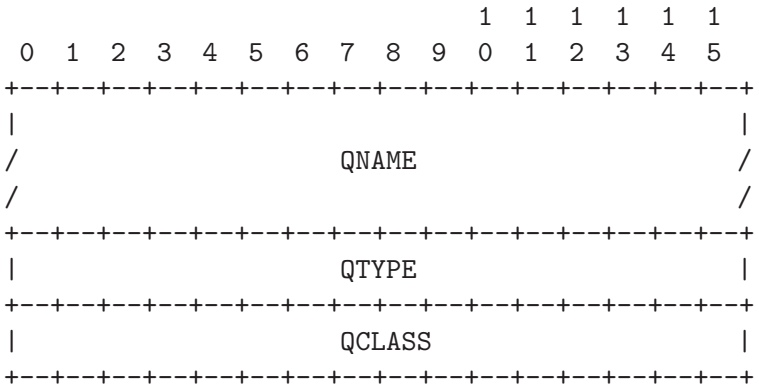
1.1.2 Header



– ID : identificateur de 16 bit généré par le programme.

- QR : bit spécifiant s'il s'agit d'une requête (0) ou d'une réponse (1).
- Opcode : spécification du genre de la requête (0 pour une requête standard ; 1 pour une requête inversée ; 2 pour une requête du status du serveur ; le reste étant réservé pour un usage futur).
- AA : *Authoritative Answer* - bit valide spécifiant dans la réponse que le serveur de noms répondant à l'autorité sur le nom de domaine se trouvant dans la question.
- TC : *TrunCation* - indique que le message a été tronqué suite à une longueur plus grande que permise par la chaîne de transmission.
- RD : *Recursion Desired* - bit à mettre à 1 si on veut que le serveur de noms poursuive la requête récursivement.
- RA : *Recursion Available* - permet à l'entité répondante d'indiquer s'il supporte ou non la récursion.
- Z : réservé pour un usage futur (pour l'instant, doit être mis à 0 dans chaque requête)
- RCODE : *Response Code* - 0 = pas d'erreur ; 1 = erreur de format ; 2 = erreur du serveur ; 3 = erreur de nom ; 4 = le serveur de noms ne supporte pas le type de requête effectuée ; 5 = le serveur de noms refuse d'effectuer la requête car sa politique l'en empêche.
- QDCOUNT : nombre d'entrées dans la section `Question`.
- ANCOUNT : nombre de RR's dans la section `Answer`.
- NSCOUNT : nombre de RR's dans la section `Authority`.
- ARCOUNT : nombre de RR's dans la section `Additional`.

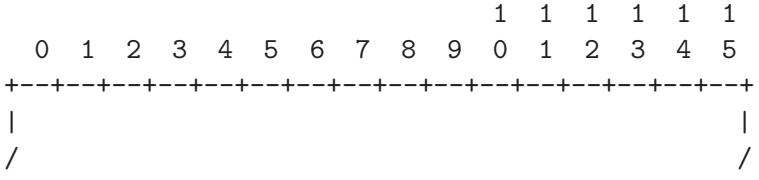
1.1.3 Question

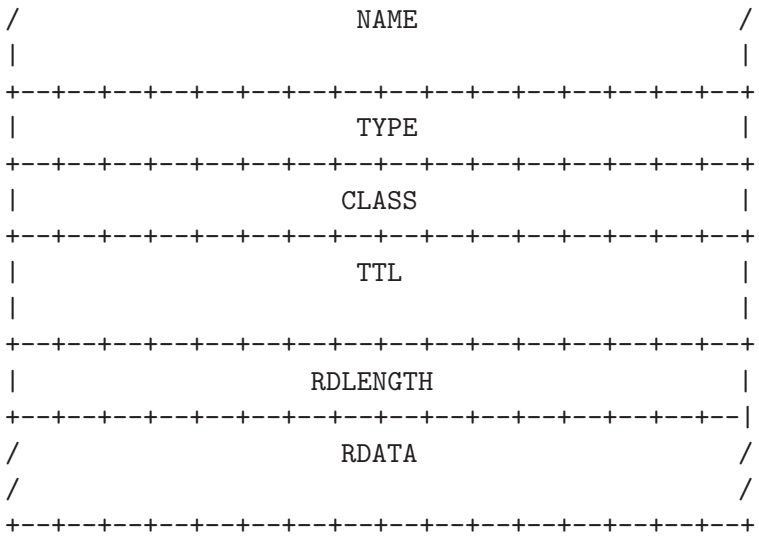


- QNAME : une séquence de byte représentant le nom de domaine, terminée par deux bytes valant 0.
- QTYPE : type de la requête (A, NS, MX, etc)
- QCLASS : classe de la requête (IN, etc)

1.1.4 Ressource Record

Les sections `Answer`, `Authority` et `Additional` répondent toutes trois au même format :





- NAME : nom de domaine auquel correspond le RR.
- TYPE : type du RR donnant la signification des données contenues dans le `RDATA`.
- CLASS : classe des donnes contenues dans le `RDATA`.
- TTL : temps pendant lequel le RR peut être contenu dans la cache.
- RDLENGTH : taille en octets des données.
- RDATA : données variant suivant le type et la classe du RR.

1.2 Réception d'une requête

```
rule dns_global_analyser;
```

Grâce à cette règle qui se réappelle à chaque fois, notre programme capte les paquets udp sur le port 53 et peut les analyser. L'analyse ne va concerner que les données du paquet udp qui contiennent le paquet dns qui nous intéresse.

Comme expliqué précédemment, celui-ci est tout d'abord composé d'une entête de 12 octets. Cette entête ne va dans un premier temps pas nous intéresser puisque seule la question - et plus précisément son type - fera l'objet d'une action dans un premier temps. Pour trouver le type, nous allons donc parcourir notre paquet DNS à partir du 13ème octet (début de la section `Question`) jusqu'à ce qu'on trouve une suite de deux bytes valant 0 (fin du champ `QNAME`). Nous sommes alors placés sur les deux bytes correspondant au type de la requête, que nous transformons en entier qui va permettre de décider quelle sera la prochaine règle à inférer. En effet, chaque type de requête possède une règle propre pour y répondre. La section suivante explicite plus en détail comment fonctionnent ces règles et de quoi se composent exactement les paquets renvoyés au client.

1.3 Réponse

Les règles permettant de répondre aux requêtes sont de la forme suivante :

```
rule reply_to_XXX(dns_query : string) ;
```

où XXX correspond au type de la requête à laquelle on veut répondre.

Chaque règle commence par construire l'entête du paquet qu'elle va envoyer, puis construit le ou les RR's `Answer`, `Authority` et `Additional` pour finalement envoyer au client un paquet udp dont les données sont la concaténation de l'entête, de la question et des RR's.

L'entête se construit de la façon suivante : l'`ID` est copié de l'entête de la requête ; le `QR` vaut 1, l'`Opcode` vaut 0, le `AA` vaut 0, le `TC` vaut 0, le `RD` vaut 1, le `RA` vaut 1, le `Z` vaut 0 et le `RCODE` vaut 0 ; le `QDCOUNT`, le `ANCOUNT`, le `NSCOUNT` et le `ARCOUNT` doivent être adaptés en fonction de chaque réponse.

La question est simplement la section `Question` de la requête qui est passée en paramètre de la règle.

Les sections `Answer`, `Authority` et `Additional` sont différentes pour chaque règle et son explicitées dans les sous-sections suivantes.

Dans le cas d'un serveur DNS classique, les réponses aux requêtes renvoient chaque fois vers un autre serveur DNS jusqu'à ce que la réponse à la requête soit satisfaisante. Sauf dans le cas des anciens serveurs DNS qui fonctionnent encore de manière récursive, mais ceux-ci ce font rare dans l'internet d'aujourd'hui pour des raisons de sécurité car ils sont facilement sujet à des attaques par dénis de service. Dès lors, partons du principe que les serveurs DNS sont itératifs, voici le fonctionnement classique.

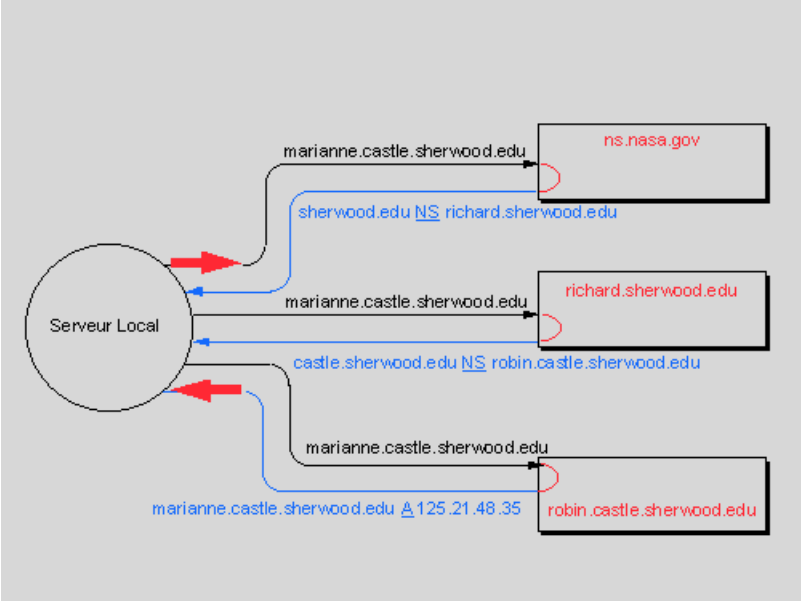


Fig. 1.1 – Serveur DNS classique

Cependant dans notre cas, notre but est bien d'attirer un maximum les robots pirates dans le Honey Pot de manière à pouvoir analyser leur comportement. Dès lors, plutôt que de renvoyer une réponse à une requête disant à l'interlocuteur d'aller voir ailleurs car nous ne sommes

pas responsable du domaine qu'il cherche, notre serveur va au contraire annoncer que le domaine recherché se trouve bien dans le Honey Pot et qu'il faut surtout continuer à chercher dans ce dernier. Comme toutes les adresses IP du Honey Pot réagissent avec le mécanisme de serveur DNS, on donne l'illusion à l'interlocuteur qu'il s'adresse à un autre serveur car on va lui proposer d'aller voir sur une autre adresse IP du Honey Pot.

1.3.1 A

Dans le cas d'une requête de type A, la réponse se compose de plusieurs éléments différents. D'abord le serveur répond de manière tout à fait complète à la question en fournissant le `canonical name` associé à l'adresse recherchée, et ensuite nous attribuons à ce `canonical name` une adresse IP appartenant au Honey Pot.

Pour trouver le `canonical name` associé à l'adresse de la requête, la technique utilisée est de parcourir l'adresse de la requête. On place un pointeur vers le caractère suivant le premier point de l'adresse. Si l'adresse contient plus de 1 point, c'est qu'on a un sous-domaine, donc on considère que le `canonical name` aura la même adresse, mais sans la partie avant le premier point.

Dans le cas où on nous enverrait une adresse IP plutôt qu'un nom de domaine, pour éviter de réagir de la même manière, on détecte si la requête comporte 3 chiffres avant le premier point. Dans ce cas on renvoie tout simplement l'adresse IP comme étant la valeur associée à la requête.

De cette manière, si le robot pirate se base sur des noms de sous-domaines pour chercher sur internet des potentielles cibles, il risque dans le futur de continuer à chercher sur le sous-domaine de la première requête et ainsi rester dans le Honey Pot tant qu'il fait des recherches sur le même sous-domaine. De cette manière on tente d'augmenter le nombre de requêtes sur le Honey Pot afin de pouvoir analyser les requêtes provenant des robots.

Ensuite, comme on a la possibilité d'utiliser les sections `authority` et `additional` on en profite pour fournir à l'interlocuteur les serveurs de noms associés au domaine de la requête. Et on associe ces serveurs de noms une fois de plus à des adresses IP appartenant au Honey Pot.

```
;; QUESTION SECTION:
;www.groupe6.be.                    IN      A

;; ANSWER SECTION:
www.groupe6.be.         12288   IN      CNAME   groupe6.be.
groupe6.be.             12288   IN      A       10.0.0.158

;; AUTHORITY SECTION:
groupe6.be.             12288   IN      NS      ns.asax1.asax2.be.
groupe6.be.             12288   IN      NS      ns.asax2.asax2.be.

;; ADDITIONAL SECTION:
ns.asax1.asax2.be.      12288   IN      A       10.0.0.2
```

```
ns.asax2.asax2.be.      12288   IN      A       10.0.0.157
```

1.3.2 MX

Dans le cas d'une requête de type MX, l'interlocuteur cherche à connaitre le serveur de mail associé à un nom de domaine. Donc le serveur lui répondra en lui donnant le `canonical name` du domaine recherché. Mais si on se contente de cette réponse l'interlocuteur n'a pas tous les éléments nécessaires car il doit aussi connaitre l'adresse IP associée à ce `canonical name`. C'est la raison pour laquelle les sections `authority` et `additional` sont utilisées. En effet, en plus de dire à l'interlocuteur qui est responsable de ce nom de domaine, on lui donne aussi les adresses IP auxquelles il peut joindre ces serveurs. Ces adresses IP faisant bien évidemment partie du Honey Pot.

```
;; QUESTION SECTION:
;www.groupe6.be.                        IN      MX

;; ANSWER SECTION:
www.groupe6.be.         12288   IN      CNAME   groupe6.be.

;; AUTHORITY SECTION:
groupe6.be.             12288   IN      NS      ns.asax1.asax2.be.
groupe6.be.             12288   IN      NS      ns.asax2.asax2.be.

;; ADDITIONAL SECTION:
ns.asax1.asax2.be.      12288   IN      A       10.0.0.2
ns.asax2.asax2.be.      12288   IN      A       10.0.0.157
```

1.3.3 CNAME

Dans le cas d'une requête de type CNAME, le raisonnement est le même que pour les requêtes de type MX.

```
;; QUESTION SECTION:
;www.groupe6.be.                        IN      CNAME

;; ANSWER SECTION:
www.groupe6.be.         12288   IN      CNAME   groupe6.be.

;; AUTHORITY SECTION:
groupe6.be.             12288   IN      NS      ns.asax1.asax2.be.
groupe6.be.             12288   IN      NS      ns.asax2.asax2.be.

;; ADDITIONAL SECTION:
ns.asax1.asax2.be.      12288   IN      A       10.0.0.2
ns.asax2.asax2.be.      12288   IN      A       10.0.0.157
```

1.3.4 ANY

Dans le cas d'une requête de type ANY sur un nom de domaine, une fois de plus on renvoie simplement le `canonical name` associé à l'adresse demandée et on associe le `canonical name` à des serveurs de noms appartenant au Honey Pot.

```
;; QUESTION SECTION:
;www.groupe6.be.                        IN      ANY

;; ANSWER SECTION:
www.groupe6.be.         12288   IN      CNAME   groupe6.be.

;; AUTHORITY SECTION:
groupe6.be.             12288   IN      NS      ns.asax1.asax2.be.
groupe6.be.             12288   IN      NS      ns.asax2.asax2.be.

;; ADDITIONAL SECTION:
ns.asax1.asax2.be.      12288   IN      A       10.0.0.2
ns.asax2.asax2.be.      12288   IN      A       10.0.0.157
```

1.3.5 SOA

Dans le cas d'une requête de type SOA, c'est une fois de plus le même raisonnement.

```
;; QUESTION SECTION:
;www.groupe6.be.                        IN      SOA

;; ANSWER SECTION:
www.groupe6.be.         12288   IN      CNAME   groupe6.be.

;; AUTHORITY SECTION:
groupe6.be.             12288   IN      NS      ns.asax1.asax2.be.
groupe6.be.             12288   IN      NS      ns.asax2.asax2.be.

;; ADDITIONAL SECTION:
ns.asax1.asax2.be.      12288   IN      A       10.0.0.2
ns.asax2.asax2.be.      12288   IN      A       10.0.0.157
```

1.3.6 NS

Dans le cas d'une requête de type NS, on demande au serveur de renvoyer l'adresse du serveur de nom du domaine. C'est la raison pour laquelle nous renvoyons d'abord la valeur de `canonical name` du domaine demandé, et ensuite on renvoie le nom du serveur de nom qui lui est attribué.

De manière à être complet dans notre réponse et éviter confusion, nous utilisons la section `additional` pour préciser les adresses IP correspondantes aux serveurs de noms. Ces adresses

IP sont évidemment des adresses du Honey Pot de manière à faire boucler l'interlocuteur dans le Honey Pot.

```
;; QUESTION SECTION:
;www.groupe6.be.                        IN      NS

;; ANSWER SECTION:
www.groupe6.be.         12288   IN      CNAME   groupe6.be.
groupe6.be.             12288   IN      NS      ns.asax1.asax2.be.
groupe6.be.             12288   IN      NS      ns.asax2.asax2.be.

;; ADDITIONAL SECTION:
ns.asax1.asax2.be.      12288   IN      A       10.0.0.2
ns.asax2.asax2.be.      12288   IN      A       10.0.0.157
```

1.3.7 not_implemented

Nous avons également prévu le cas où arriverait une requête d'un type que nous ne prenons pas en compte. Dans ce cas nous renvoyons un paquet formés des seules sections `header` et `question`, le `header` contenant un `RCODE` valant 4 (correpondant à une *not implemented error*).

1.4 Statistiques

Grâce à la règle `rule timeStat` nous imprimons toutes les 120 secondes les statistiques ayant trait au nombre de requête de chaque type effectuées. Ce choix a été fait étant donné que dans le cas ou notre programme va tourner en boucle pendant une longue période, il n'affichera jamais les statistiques à l'écran si nous utilisons une règle de type `trigger off at_completion` et affichera trop de statistiques si on affiche celles-ci à la réception de chaque paquet.

D'où le choix d'afficher les statistiques après un certain laps de temps. Mais le temps étant calculé en fonction du *timestemp* du paquet couramment reçu il se peut parfois que le temps entre deux affichages de statistique prenne plus de temps étant donné que tant qu'on ne reçoit pas de paquet, le temps n'est pas mis à jour.

Section 2

Test

Afin de bien comprendre l'utilité et l'effet de chaque byte des paquets échangés, nous nous sommes servis du rfc 1035 et du programme Wireshark (anciennement ethereal). Ce dernier nous a permis de capter des paquets correspondant à du trafic DNS pour les décomposer et bien intégrer les différentes sections de ces paquets.

Au niveau de la validation proprement dite, nous nous en sommes essentiellement tenus à faire des `dig` de la machine asax2 vers la machine asax1 pour divers noms de domaines et types de requêtes. L'affichage correct ou non de la réponse nous a alors guidé pour savoir si nous renvoyions des paquets bien formés ou pas. Pour tester la règle `not_implemented`, nous avons également envoyé un `dig` de la machine asax2 mais en hardcodant un numéro de type inconnu, causant par là le renvoi du paquet correspondant à un type non implémenté sur asax2 qui l'affichait de la bonne manière également.