Tester la robustesse des serveurs DNS avec Scapy

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 4 mars 2010

https://www.bortzmeyer.org/paquets-invalides-scapy.html

Comme aime le dire Paul Vixie, l'Internet est, pour les serveurs réseau, plutôt l'équivalent de New York que celui du petit village tranquille où tout le monde se connait. Si on s'évanouit de peur à chaque truc bizarre qui passe, on ne va pas loin. Un serveur réseau connecté à l'Internet doit être **robuste** : il va voir des tas de paquets mal formés et ne devra pas planter (ou, pire, ouvrir une voie de pénétration) lorsqu'un de ces paquets arrive. Un des outils pour tester facilement la robustesse d'un serveur est Scapy. Voyons son application à des serveurs DNS.

Un serveur DNS reçoit des paquets UDP et TCP qui sont normalement conformes à la description qu'en fait le RFC 1035 [1] et quelques RFC ultérieurs (notamment le RFC 2671). Suivre à la lettre ces RFC devrait mener à du code parfait. Ainsi, pour analyser un nom de domaine figurant dans la section Question d'un paquet, on lit la section 4.1.2 du RFC 1035, qui dit que le nom est représenté par une suite de composantes, chaque composante étant faite d'un octet (qui indique la longueur de la composante) et d'une suite d'octets pour le texte. On arrive à du code qui ressemble à (en Go) :

```
for {
        labelsize, error := buf.ReadByte()
        if labelsize == 0 {
                break
        }
        label := make([]byte, labelsize)
        n, error := buf.Read(label)
        nlabels += 1
        labels = labels[0:nlabels]
        labels[nlabels-1] = string(label)
}
```

ou bien (en C) :

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc1035.txt

```
for (sectionptr = qsection; !end_of_name;) {
  labelsize = (uint8_t) * sectionptr;
  if (labelsize == 0) {
      sectionptr++;
      end_of_name = true;
  } else {
      strncat(fqdn, ".", 1);
      strncat(fqdn, (char *) sectionptr + 1, labelsize);
      fqdn_length += (labelsize + 1);
      fqdn[fqdn_length] = '\0';
      sectionptr = sectionptr + labelsize + 1;
  }
}
```

Ces codes (simplifiés) marchent avec des paquets normaux. Mais l'expérience montre clairement que, dans le vrai Internet, il existe une minorité non négligeable de paquets anormaux. On les voit indirectement dans les alertes de sécurité de Wireshark <http://www.wireshark.org/security/>. Pour le cas du DNS, l'usage d'un outil comme DNSmezzo <http://www.dnsmezzo.net> sur un serveur de noms public montre rapidement des milliers de paquets qui violent le RFC. D'où viennent ces paquets ? Certains sont issus d'erreurs de programmation (il y a des tas d'amateurs incompétents qui programment), d'autres sont des essais par des étudiants en plein TP d'écriture d'un client DNS, d'autres enfin sont d'authentiques tentatives de piratage, par exemple en cherchant à provoquer un débordement de tampon chez le serveur.

Tout code réel (i.e. pas les deux exemples plus haut) va donc chercher à se protéger contre de tels paquets. On vérifie donc, on regarde si un pointeur de compression ne sort pas du paquet, on teste si on est arrivé au bout du paquet avant de continuer aveuglément, etc. C'est le B. A. BA de la programmation réseau.

Mais comment tester que ces protections sont efficaces ? Il faudrait pouvoir fabriquer facilement des paquets anormaux. Et la plupart des bibliothèques de programmation réseau sont au contraire conçues pour faciliter la production de paquets normaux. C'est là que Scapy devient utile.

Scapy est un outil d'analyse de paquets existants et de fabrication de paquets, offrant une très grande souplesse de manipulation. Scapy fait tout tout seul... mais laisse le programmeur intervenir où il veut. C'est l'outil rêvé des programmeurs qui veulent tester un logiciel (ou des pirates qui veulent l'attaquer). S'appuyant sur Python, il nécessite de connaitre ce langage. Lisez la documentation <http://www.secdev.org/projects/scapy/doc/usage.html>. Puis utilisons-le en interactif pour commencer :

```
# scapy
>>> p = IP(dst="203.0.113.162")/UDP(sport=RandShort(),dport=53)/\
...       DNS(rd=1,qd=DNSQR(qname="www.slashdot.org", qtype="AAAA"))
```

On a fabriqué une requête DNS (demande de l'adresse IPv6 - `AAAA` - de `www.slashdot.org`). Notez qu'un certain nombre de paramètre n'ont pas été fournis (par exemple l'adresse IP source), Scapy les fournira. Vérifions :

```
>>> p.show()
###[ IP ]###
  version= 4
```

```
...
  chksum= 0x0
  src= 203.0.113.69
  dst= 203.0.113.162
  options= ''
###[ UDP ]###
     sport= <RandShort>
     dport= domain
     len= None
     chksum= 0x0
###[ DNS ]###
        id= 0
        qr= 0
        opcode= QUERY
        aa= 0
        tc= 0
        rd= 1
        ra= 0
        z= 0
        rcode= ok
        qdcount= 1
        ancount= 0
        nscount= 0
        arcount= 0
        \qd\
         |###[ DNS Question Record ]###
         |  qname= 'www.slashdot.org'
         |  qtype= AAAA
         |  qclass= IN
        an= None
        ns= None
        ar= None
```

Le champ `src` (adresse IP source) a bien été rempli. Des champs comme la longueur du paquet UDP (`len`, actuellement `None`, la valeur indéfinie en Python) seront remplis automatiquement lors de l'émission du paquet. De même, l'appel à `RandShort()` ne sera évalué qu'au moment de l'envoi du paquet (et produira un nombre aléatoire).

D'ores et déjà, je peux interactivement modifier tous les champs, par exemple :

```
>>> p.rd = 0
```

et `p.show()` (ou bien tcpdump si on envoie le paquet) montrera bien que la requête n'est plus récursive.

Ah, justement, envoyons le paquet :

```
>>> sr1(p)
Begin emission:
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
<IP  version=4L ihl=5L tos=0x0 len=62 id=0 flags=DF frag=0L ttl=63 proto=udp chksum=0xb1bb src=203.0.113.162 dst=
```

Le paquet a été transmis et la réponse (« il n'y a pas d'adresse IPv6 pour `www.slashdot.org` ») reçue et analysée.

Bien, tout cela, c'étaient des paquets DNS normaux. Maintenant, essayons de fabriquer des paquets anormaux, pour voir si le serveur DNS réagit bien. Je vais tester avec un serveur DNS très sommaire, qui a peu de protections, GRONG <https://www.bortzmeyer.org/dnsserver-en-go.html>. Si je lance GRONG et que je lui envoie le paquet ci-dessus, le serveur affiche :

```
% ./server -debug=4
34 bytes packet from 203.0.113.69:24007
Query is true, Opcode is 0, Recursion is true, Rcode is 0
FQDN is www.slashdot.org, type is 28, class is 1
Replying with ID 0...
```

Place maintenant à un paquet DNS malformé. Changeons `qdcount` (dans la section DNS) qui indique la taille de la section Question (normalement, 1) :

```
>>> p[DNS].qdcount=0
```

et envoyons ce paquet incohérent (`qdcount` ne correspond plus à la taille de la section Question), le résultat ne se fait pas attendre, le serveur, programmé de manière insuffisamment défensive, plante :

```
34 bytes packet from 203.0.113.69:23287
throw: index out of range

panic PC=0x40078b90
throw+0x46 /usr/local/go/src/pkg/runtime/runtime.c:74
        throw(0x80bcd88, 0x0)
runtime.throwindex+0x24 /usr/local/go/src/pkg/runtime/runtime.c:47
        runtime.throwindex()
main.parse+0x407 /home/stephane/src/Go/dns/grong/server.go:137
        main.parse(0x40046c00, 0x0, 0x0)
main.generichandle+0x5d /home/stephane/src/Go/dns/grong/server.go:147
        main.generichandle(0x40046c00, 0x400475c0, 0x40042270, 0x0, 0x0, ...)
main.udphandle+0x136 /home/stephane/src/Go/dns/grong/server.go:191
        main.udphandle(0x400414c8, 0x400475c0, 0x40042270, 0x40046c00, 0x0, ...)
goexit /usr/local/go/src/pkg/runtime/proc.c:140
        goexit()
0x400414c8 unknown pc
```

Heureusement, Go nous fournit une belle pile d'appels, qui nous permettra d'améliorer le serveur (ce sera pour un autre article).

Changer uniquement un champ est amusant mais on peut aussi modifier plus drastiquement le paquet. Par exemple, si je veux le tronquer ? Il faut pour cela sérialiser le paquet en une chaîne de caractères (avec `str()`), le modifier puis le retransformer en paquet :

```
>>> s = str(p)
>>> p2 = IP(s[:-4])
>>> sr1(p2)
```

Ce code transforme `p` dans la chaîne `s` puis retransforme en paquet la chaîne privée de ses quatre derniers octets. Mais le paquet n'arrive pas au serveur de noms, qui semble ne rien recevoir. C'est parce que le gros problème de cette approche est que la sérialisation « gèle » les valeurs « flottantes » comme la longueur indiquée dans l'en-tête, qui est désormais invalide. Remettons tout cela à zéro, ainsi que les sommes de contrôle et Scapy le recalculera proprement :

```
>>> del p2[UDP].len
>>> del p2[IP].len
>>> p2[UDP].sport=RandShort()
>>> del p2[UDP].chksum
>>> del p2[IP].chksum
>>> sr1(p2)
```

Désormais, le paquet tronqué est bien reçu... et plante le serveur :

```
30 bytes packet from 203.0.113.69:12662
Error in Read of an int16: EOF (0 bytes read)
```

Et si je veux modifier une valeur quelconque, non accessible depuis un champ nommé, par exemple la longueur d'une des composantes du FQDN ? Là encore, on sérialise, on fabrique une nouvelle chaîne en changeant un des octets (ici, le n° 44) et on refait le paquet.

```
>>> s=str(p)
>>> s
'E\x00\x00>\x00\x01\x00\x00@\x11\xf0\xfb\xc0\x86\x04E\xc0\x86\x04au*\x1fu\x00*\xce\x18\x00\x00\x01\x00\x00\x01\x
>>> s[44]
'\x08'
>>> s2 = s[:44] + '\x030' + s[45:]
>>> s2
'E\x00\x00>\x00\x01\x00\x00@\x11\xf0\xfb\xc0\x86\x04E\xc0\x86\x04au*\x1fu\x00*\xce\x18\x00\x00\x01\x00\x00\x01\x
>>> p2 = IP(s2)
>>> del p2[UDP].chksum
>>> del p2[IP].chksum
>>> sr1(p2)
```

À noter que cette erreur particulière ne pose pas de problème à GRONG qui ignore ce paquet, reconnu comme invalide.

Bien sûr, il y a des tas de tests que l'on souhaite ainsi faire. Scapy fournit d'ailleurs des moyens de faire varier certains paramètres ("*fuzzy testing*"). On a donc intérêt à mettre ces tests dans un programme Python normal, qui utilisera Scapy pour faire tourner tous les tests... et vérifier ainsi le serveur. Pour un exemple d'un tel programme, voir (en ligne sur `https://www.bortzmeyer.org/files/test-dns-scapy.py`). Je n'ai pas de scrupule à livrer ce programme de test de robustesse car BIND et nsd, bien écrits, y survivent sans problème.

Ah, un petit gag avec Scapy, qui est dans la FAQ mais qui m'avait coûté quelque temps de recherche. Au début, on commence souvent par tester un serveur qui figure sur la même machine et on teste donc l'adresse IP locale, `127.0.0.1`. Mais l'interface locale a quelques particularités et Scapy exige donc qu'on indique, dans ce cas :

```
>>> conf.L3socket=L3RawSocket
```

Cette instruction ne marche pas, à l'heure actuelle, depuis un programme Python, uniquement en interactif (bogue Scapy #193 <http://trac.secdev.org/scapy/ticket/193>). Depuis un programme, on obtient un `NameError: global name 'Ether' is not defined`. La correction est d'ajouter `from scapy.layers.l2 import Ether` dans les importations de `scapy/supersocket.py` (merci à rmkml).

Autre piège où j'ai passé du temps en apprenant Scapy : = ne fait pas une copie des paquets, le nouveau paquet, si j'écris `p2 = p` est juste un pointeur. Pour copier réellement un paquet :

```
>>> p2 = p.copy()
```

Merci à Pierre-François Bonnefoi pour son aide. Il est par ailleurs l'auteur d'un excellent support de cours Scapy en français <http://ishtar.msi.unilim.fr/uploads/media/Scapy_handout_2009_2010_TOC.pdf>. L'exposé « "*Network packet forgery with Scapy*" <http://www.secdev.org/conf/scapy_pacsec05.pdf> », de Philippe Biondi, contient plein de techniques utiles.