Se protéger contre l'identification par prise d'empreinte TCP/IP

Nous étudions dans cet article deux types de mécanismes ayant pour but de compliquer la détection du système d'exploitation par analyse de la pile TCP/IP [1]. Nous ne discutons que des méthodes de protection au niveau des couches 3 et 4, il n'adresse aucun des problèmes de détection par analyse des bannières, ou analyse de protocoles applicatifs.

Procédure de test utilisée au cours de cet article

Pour chaque test, *nmap* [2] et *Xprobe* [3] sont utilisés pour valider la qualité de chaque mécanisme de protection. *ring* [4] pourrait être utilisé, mais ses résultats ne seront pas impactés, étant donné qu'il s'appuie uniquement sur l'algorithme de retransmission des segments TCP lors d'une tentative de connexion, et que ce paramètre n'est pas modifié dans nos démonstrations.

Notons que même si *nmap* est utilisé dans une écrasante majorité des tentatives d'OS *fingerprinting*, il existe certainement des outils non publics, donc construits sur des tests non connus, et que l'on ne peut alors pas contrer en utilisant des méthodes spécifiques écrites contre un outil en particulier. C'est pour cela que nous nous focalisons ici sur les méthodes génériques de protection uniquement.

L'utilité de se protéger

Le but de cette protection est d'empêcher la détection de l'OS par l'analyse de la pile TCP/IP. Pourquoi faire cela ? Tout simplement parce dans le cadre d'intrusion réseau, c'est une étape primordiale qui servira lors du lancement d'exploits (entre autres lors du choix du shellcode à utiliser).

On peut penser que cela ressemble fortement à de la sécurité par l'obscurité, mais en fait il n'en est rien ; c'est une couche supplémentaire de protection, puisque limiter la fuite d'information est un principe de sécurité.

Le principe de cette protection

Le principe est de réduire le nombre possible de réponses a des probes (donc de limiter au maximum la fuite d'information), afin de diminuer le nombre de signatures différentes possibles, rendant ainsi difficile l'identification sûre. Il est également utile de modifier des valeurs bien connues pour être déterminantes pour tel ou tel OS (exemple : taille de la fenêtre TCP initiale).

Deux grandes méthodes de protection efficaces

Nous abordons dans cet article deux grandes méthodes de protection. Une autre (très populaire si l'on en juge par le nombre d'outils disponibles) qui n'est pas abordée ici consiste à simplement leurrer *nmap* [5][6][7], ou spécifiquement d'autres outils d'OS *fingerprinting*. Puisqu'elle n'est pas générique, c'est-à-dire qu'elle ne traite pas le problème dans son ensemble, indépendamment d'une implémentation de détection distante d'OS, elle est facilement contournable, et ne présente donc pas d'intérêt (c'est dans ce cas de la sécurité par l'obscurité).

Approche de personnalisation de la pile IP

Cette méthode à pour but de modifier le comportement de la pile IP de manière à ce que la signature de l'OS ne corresponde plus à celle bien connue. On peut aussi coupler cela avec un dispositif de filtrage en coupure, et un dispositif de normalisation de paquet pour réduire encore la fuite d'information.

On l'applique localement à chaque machine à protéger (sauf dans le cas de l'ajout d'un dispositif de filtrage en coupure, nous y reviendrons). Aucune modification n'est faite au noyau de l'OS, seuls les paramètres configurables par le système lui-même sont ajustés. Elle n'est pas intrusive, et est donc plus sûr quant à la stabilité du système.

Approche de transformation de la pile IP

Cette deuxième approche consiste en l'application de *patch*s dans le noyau permettant de modifier complètement le comportement de la pile. Cette méthode donne une plus grande latittude dans le changement de son comportement, et nous comparons dans cet article l'efficacité des deux méthodes. En revanche, étant plus intrusive, il peut y avoir des impacts importants quant aux performances réseau et au comportement RFC de la pile (cet article n'aborde pas ces problèmes).

Elle fonctionne également localement à la machine, mais est surtout aussi appliquable à l'entrée d'un réseau, et protège donc toutes les machines internes à celui-ci. C'est une approche beaucoup plus globale contre la détection de l'OS sur un réseau à protéger, donc dans son principe bien plus proche d'une bonne politique de sécurité.

Personnalisation de la pile IP et filtrage

FreeBSD blackhole(4) [8] *sysctl*

La fonctionalité *blackhole(4)* change le comportement de la pile lors de l'émission de paquets en réponse aux scans de ports TCP et UDP. Cela fonctionne comme un dispositif de filtrage, mais est utilisable directement par un appel à *sysctl(8)* :

```
fbsd49# sysctl -a | grep blackhole
net.inet.tcp.blackhole: 0
net.inet.udp.blackhole: 0
```

L'utilisation de cette fonctionalité permet à une machine n'ayant aucun port ouvert d'être invisible si seulement un scan de port TCP ou UDP est utilisé pour la trouver.

Le but premier de *blackhole(4)* n'est pas de masquer l'empreinte de sa pile IP, mais puisqu'il permet de ne pas répondre a certains probes, il limite la fuite d'information (nécessaire à une bonne identification d'OS).

En effet, lors d'un scan TCP, un port fermé emettra un RST+ACK par défaut, mais si le *sysctl* est placé à 1, il n'est pas émis. Si l'option est placée à 2, aucun paquet TCP expédié à ce port ne recoit de réponse. La différence entre 1 et 2 en TCP est que à 1, si le paquet TCP à un *flag* SYN, il n'y a pas de réponse, mais si il n'y a pas de SYN, mais un FIN (par exemple), il y en a une. A 2, même le FIN n'obtient pas de réponse (pour une description des différents types de scan de port, voir [9]).

Les trois tests sur un FreeBSD 4.9, sans utilisation de *blackhole(4)* :

1. Un TCP SYN sur un port fermé

```
obsd32# hping -S -p 23 -c 1 fbsd49
HPING fbsd49 (ne3 192.168.0.52): S set, 40 headers + 0 data bytes
len=46 ip=192.168.0.52 ttl=64 id=549 sport=23 flags=RA seq=0 win=0 rtt=1.0
ms
```

2. Un TCP FIN sur un port fermé

```
obsd32# hping -F -p 23 -c 1 fbsd49
HPING fbsd49 (ne3 192.168.0.52): F set, 40 headers + 0 data bytes
len=46 ip=192.168.0.52 ttl=64 id=702 sport=23 flags=RA seq=0 win=0 rtt=0.7
ms
```

3. Un UDP sur un port fermé

```
obsd32# hping --udp -p 23 -c 1 fbsd49
HPING fbsd49 (ne3 192.168.0.52): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from ip=192.168.0.52 name=fbsd49.enslaved.lan
```

Tableau des resultats avec les trois valeurs pour les *sysctl*s TCP et UDP :

Test	blackhole=0	blackhole=1	blackhole=2
1 : TCP	RST+ACK	Pas de réponse	Pas de réponse
2 : TCP	RST+ACK	RST+ACK	Pas de réponse
3 : UDP	ICMP Port Unreachable	Pas de réponse	N.A.

Une détection d'OS avec *nmap*, et les *sysctl*s par défaut (0) (pour une description plus complète de l'interprétation des résultats de *nmap*, se reporter à [10]) :

```
obsd32# nmap -p 22,23 -O -vv --osscan_guess fbsd49
[..]
PORT   STATE  SERVICE
22/tcp open   ssh
23/tcp closed telnet
Device type: general purpose
Running: FreeBSD 4.X
OS details: FreeBSD 4.6.2-RELEASE - 4.8-RELEASE
OS Fingerprint:
TSeq(Class=TR%IPID=I%TS=100HZ)
T1(Resp=Y%DF=Y%W=E000%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
```

```
T3(Resp=Y%DF=Y%W=E000%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=0%ULEN=134%DAT=E)
[..]
```

Puis un test avec *Xprobe* (pour une description des tests de *Xprobe*, voir [11]) :

```
obsd32# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p udp:514:open
fbsd49
[..]
[+] Primary guess:
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.8" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.7" (Guess probability: 100%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.4" (Guess probability: 97%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.6" (Guess probability: 97%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.6.2" (Guess probability: 97%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 5.1" (Guess probability: 94%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 5.0" (Guess probability: 94%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.5" (Guess probability: 91%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.4.3" (Guess probability: 79%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.5" (Guess probability: 79%)
[..]
```

Maintenant, en placant les *sysctl*s comme ceci :

```
fbsd49# sysctl -w net.inet.tcp.blackhole=2
fbsd49# sysctl -w net.inet.udp.blackhole=1

obsd32# nmap -p 22,23 -O -vv --osscan_guess fbsd49
[..]
PORT   STATE    SERVICE
22/tcp open     ssh
23/tcp filtered telnet
Device type: general purpose
Running: FreeBSD 4.X
OS details: FreeBSD 4.6.2-RELEASE - 4.8-RELEASE, FreeBSD-4.7-RELEASE-p3, FreeBSD
4.8-STABLE
OS Fingerprint:
TSeq(Class=TR%IPID=I%TS=100HZ)
T1(Resp=Y%DF=Y%W=E000%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=E000%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=N)
T6(Resp=N)
T7(Resp=N)
PU(Resp=N)
[..]

obsd32# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p udp:514:open
fbsd49
[..]
[+] Primary guess:
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.7" (Guess probability: 70%)
[+] Other guesses:
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.8" (Guess probability: 70%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.4" (Guess probability: 67%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.6" (Guess probability: 67%)
```

```
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.6.2" (Guess probability: 67%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 5.1" (Guess probability: 64%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 5.0" (Guess probability: 64%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 2.2.8" (Guess probability: 61%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.5" (Guess probability: 61%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 2.2.7" (Guess probability: 61%)
[..]
```

On distingue clairement que *nmap* obtient très peu de réponses à ses probes, mais que cela ne l'empêche pas de trouver que la machine tourne sous FreeBSD 4.x. Ceci dit, il est trivial d'ajouter la signature dans le fichier de signatures, avec comme nom `*FreeBSD 4.x with blackhole*'. *Xprobe*, quant à lui, parvient aussi toujours à trouver l'OS, mais il est moins sûr de ses résultats. C'est normal, étant donné que lui aussi à moins de réponses à ses tests.

En résumé, cela n'est pas suffisant pour se protéger. La raison en est que un port TCP ouvert fournit suffisamment d'information pour identifier un système. Donc, dans le paragraphe suivant, nous allons étudier la personnalisation de cette pile TCP.

FreeBSD *net.inet sysctl*s

Il y a quatres branches de la *MIB* (*Management Information Base*) sur lesquelles nous pouvons jouer sous FreeBSD pour modifier significativement le comportement de la pile IP. Les voici :

- net.inet.ip
- net.inet.tcp
- net.inet.udp
- net.inet.icmp

Nous ne rentrons pas dans le détail de chaque variable (88 au total sous FreeBSD 5.2.1). En revanche, déroulons un exemple d'utilisation, en modifiant la taille de la fenêtre TCP, ainsi que le comportement TCP au regard de la RFC1323 (qui discute de l'implémentation des options TCP *Timestamp* et *Window scale* à des fins d'amélioration des performances).

Valeurs par défaut	Valeurs modifiées pour déjouer l'OS fingerprinting
net.inet.tcp.blackhole=0	net.inet.tcp.blackhole=2
net.inet.udp.blackhole=0	net.inet.udp.blackhole=1
net.inet.tcp.sendspace=32768	net.inet.tcp.sendspace=16384
net.inet.tcp.recvspace=57344	net.inet.tcp.recvspace=16384
net.inet.tcp.rfc1323=1	net.inet.tcp.rfc1323=0

```
obsd32# nmap -p 22,23 -O -vv --osscan_guess fbsd49
[..]
PORT   STATE    SERVICE
22/tcp open     ssh
23/tcp filtered telnet
Device type: general purpose
Running (JUST GUESSING) : Amiga AmigaOS (94%), FreeBSD 4.X|5.X (90%), IBM AIX
4.X|5.X (85%)
Aggressive OS guesses: AmigaOS Miami 3.0 (94%), AmigaOS Miami 3.1-3.2 (94%), \
  FreeBSD 4.3 - 4.4-RELEASE (90%), FreeBSD 4.7-RELEASE (X86) (90%), \
  FreeBSD 5.0-RELEASE (90%), IBM AIX 4.3.2.0-4.3.3.0 on an IBM RS/* (85%), \
  IBM AIX 5.1 (85%), FreeBSD 4.1.1 - 4.3 (X86) (85%), FreeBSD 4.9 - 5.1 (85%)
No exact OS matches for host (test conditions non-ideal).
```

```
TCP/IP fingerprint:
SInfo(V=3.50%P=i386-unknown-openbsd3.2%D=5/21%Time=40ADE5B5%O=22%C=-1)
TSeq(Class=TR%IPID=I%TS=U)
T1(Resp=Y%DF=Y%W=4000%ACK=S++%Flags=AS%Ops=M)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=4000%ACK=S++%Flags=AS%Ops=M)
T4(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=N)
T6(Resp=N)
T7(Resp=N)
PU(Resp=N)
[..]

obsd32# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p udp:514:open
fbsd49
[..]
[+] Primary guess:
[+] Host 192.168.0.52 Running OS: "Linux Kernel 2.0.36" (Guess probability: 61%)
[+] Other guesses:
[+] Host 192.168.0.52 Running OS: "Linux Kernel 2.0.34" (Guess probability: 61%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 3.1" (Guess probability: 61%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 3.2" (Guess probability: 61%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 3.3" (Guess probability: 61%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 3.4" (Guess probability: 61%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 3.5.1" (Guess probability: 61%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.0" (Guess probability: 61%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.1.1" (Guess probability: 61%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.2" (Guess probability: 61%)
[..]
```

Voilà, on distingue maintenant que *nmap* n'a plus suffisamment de réponses déterminantes pour identifier l'OS de manière sûre. Mais ce n'est pas encore parfait, FreeBSD 4.x est en deuxième choix probable, et on peut toujours rajouter une signature *nmap* (même si avec si peu de réponses, c'est un peu risqué). De même, *Xprobe* hésite entre un Linux 2.0.x et quelques versions de FreeBSD. On peut bien sûr parfaire ce masquage en jouant sur d'autres éléments de la *MIB*.

Linux 2.4 *net.ipv4 sysctl*s

Le document [12] décrit une partie des éléments de la *MIB* concernant *net.ipv4*. Il y a 160 paramètres sur lesquels jouer, dont 29 uniquement pour TCP, mais malheureusement, peu d'entre eux modifient effectivement le format des paquets. Nous allons donner un exemple de déjouement d'OS *fingerprinting* en manipulant les paramètres TCP concernant la RFC1323 (donc les même que pour FreeBSD testés précédemment).

Valeurs par défaut	Valeurs modifiées pour déjouer l'OS fingerprinting
net.ipv4.tcp_window_scaling=1	net.ipv4.tcp_window_scaling=0
net.ipv4.tcp_timestamps=1	net.ipv4.tcp_timestamps=0

Un scan sur une machine Linux 2.4.18 par défaut nous donne :

```
obsd32# nmap -p 22,23 -O -vv --osscan_guess rh72
[..]
PORT   STATE  SERVICE
22/tcp open   ssh
23/tcp closed telnet
Device type: general purpose
Running: Linux 2.4.X|2.5.X
```

```
OS details: Linux Kernel 2.4.0 - 2.5.20
OS Fingerprint:
TSeq(Class=RI%gcd=1%SI=2CE144%IPID=Z%TS=100HZ)
T1(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
[..]

obsd32# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p udp:111:open
rh72
[..]
[+] Primary guess:
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.5" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.6" (Guess probability: 100%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.7" (Guess probability: 100%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.8" (Guess probability: 100%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.9" (Guess probability: 100%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.10" (Guess probability:
100%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.11" (Guess probability:
100%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.12" (Guess probability:
100%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.13" (Guess probability:
100%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.14" (Guess probability:
100%)
[..]
```

Maintenant, appliquons les modifications décrites dans le tableau précédent :

```
obsd32# nmap -p 22,23 -O -vv --osscan_guess rh72
[..]
PORT   STATE  SERVICE
22/tcp open   ssh
23/tcp closed telnet
Device type: general purpose
Running: Linux 2.4.X
OS details: Linux 2.4.7 (X86)
OS Fingerprint:
TSeq(Class=RI%gcd=2%SI=1B643D%IPID=Z%TS=U)
T1(Resp=Y%DF=Y%W=16D0%ACK=S++%Flags=AS%Ops=M)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=16D0%ACK=S++%Flags=AS%Ops=M)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
[..]

obsd32# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p udp:111:open
rh72
[..]
[+] Primary guess:
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.5" (Guess probability: 85%)
```

```
[+] Other guesses:
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.6" (Guess probability: 85%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.7" (Guess probability: 85%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.8" (Guess probability: 85%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.9" (Guess probability: 85%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.10" (Guess probability: 85%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.18" (Guess probability: 85%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.17" (Guess probability: 85%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.16" (Guess probability: 85%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.15" (Guess probability: 85%)
[..]
```

Cela n'impacte pas la détection d'OS. La raison en est que la taille de la fenêtre TCP initiale ne varie pas, et que c'est vraiment un paramètre très significatif pour l'identification. Chaque système, ou presque, utilise une valeur différente, et se baser uniquement sur celle-ci s'avère très efficace (nous ferons un exemple un peu plus loin). Malheureusement, sous Linux 2.4, il n'existe pas de moyen pour la manipuler sans modifier le noyau. En effet, Linux utilise un algorithme d'*autotuning* pour écrire certains champs des paquets.

Nous allons tricher un peu juste pour vérifier cela. Nous modifions la taille de la fenêtre par la méthode décrite dans la section 5.2., et relancer nos tests :

```
obsd32# nmap -p 22,23 -O -vv --osscan_guess rh72
[..]
PORT   STATE  SERVICE
22/tcp open   ssh
23/tcp closed telnet
Device type: general purpose|printer
Running (JUST GUESSING) : Linux 2.4.X|2.6.X|2.5.X|2.3.X (94%), Maxim-IC TiniOS
(94%), \
  Novell Netware 5.X|4.X (87%), Lexmark embedded (86%), Sun Solaris 2.X|7 (86%)
Aggressive OS guesses: Linux 2.4.18 - 2.4.19 w/o tcp_timestamps (94%), Linux
2.4.7 (X86) (94%), \
  Linux 2.6.0-test5-love3 (X86) (94%), Maxim-IC TiniOS DS80c400 (94%), Linux
Kernel 2.4.0 - 2.5.20 (89%), \
  Linux Kernel 2.4.0 - 2.5.20 w/o tcp_timestamps (89%), Linux 2.4.23-grsec w/o
timestamps (88%), \
  Linux 2.4.18 (88%), Linux 2.4.22 (X86) w/grsecurity patch and with timestamps
disabled (88%), \
  Linux 2.3.49 x86 (88%)
No exact OS matches for host (If you know what OS is running on it, \
  see http://www.insecure.org/cgi-bin/nmap-submit.cgi).
TCP/IP fingerprint:
SInfo(V=3.50%P=i386-unknown-openbsd3.2%D=6/5%Time=40C1AB5B%O=22%C=23)
TSeq(Class=RI%gcd=1%SI=3473CE%IPID=Z%TS=U)
TSeq(Class=RI%gcd=1%SI=347313%IPID=Z%TS=U)
TSeq(Class=RI%gcd=3%SI=117BDE%IPID=Z%TS=U)
T1(Resp=Y%DF=Y%W=1000%ACK=S++%Flags=AS%Ops=M)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=1000%ACK=S++%Flags=AS%Ops=M)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=N)
[..]

obsd32# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p udp:111:open
rh72
[..]
[+] Primary guess:
```

```
[+] Host 192.168.0.60 Running OS: "FreeBSD 3.5.1" (Guess probability: 58%)
[+] Other guesses:
[+] Host 192.168.0.60 Running OS: "FreeBSD 4.0" (Guess probability: 58%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 4.1.1" (Guess probability: 58%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 4.2" (Guess probability: 58%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 4.3" (Guess probability: 58%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 3.4" (Guess probability: 58%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 3.3" (Guess probability: 58%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 3.2" (Guess probability: 58%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 3.1" (Guess probability: 58%)
[+] Host 192.168.0.60 Running OS: "Linux Kernel 2.4.5" (Guess probability: 55%)
[..]
```

Nous voyons bien que l'impact sur la détection d'OS est grand, mais Linux 2.4 apparait toujours dans les possibilités d'OS identifié.

Inspection de l'état (*stateful inspection*)

Lors de dialogues IP, certains paquets servent à établir une connexion (TCP), ou pseudo-connexion (UDP, ICMP). L'inspection de l'état est alors utilisée pour créer un contexte sur ces connexions, servant ainsi à éliminer les paquets qui sont hors de celles-ci. Par exemple, si aucune connexion TCP n'a été établie, il est inutile de traiter un segment TCP ayant un *flag* ACK.

L'inspection de l'état prend alors son sens contre l'OS *fingerprinting* lorsque certains probes se basent sur des paquets hors connexion. En général, elle est associée à une politique de filtrage telle que : tout ce qui n'est pas permit explicitement est détruit. Exemple, une machine fournit un service SMTP, il est alors inutile d'accepter des connexions sur un autre port que le 25. Donc, seuls les segments TCP ayant un *flag* SYN permettant de créer un contexte de connexion seront acceptés, et ensuite seuls les segments TCP en rapport avec cette connexion seront acceptés.

Une autre fonctionalité apportée grace à l'inspection de l'état est celle de réécriture de l'*ISN* (*Initial Sequence Number*) utilisé lors de l'établissement d'une connexion TCP. Certains probes d'OS *fingerprinting* se basent sur l'algorithme de génération de l'*ISN* pour acquérir un peu plus d'information. Dans *Packet Filter* sous OpenBSD, c'est la directive `modulate state` qui joue sur ce paramètre.

Dans les tests suivants portant sur la normalisation de paquets, nous utilisons un filtrage complet (seul un port TCP accessible), couplé à du *stateful inspection* et de la réécriture d'*ISN*.

Filtrage et normalisation de paquets via *Packet Filter*

Le but initial de la normalisation n'est pas de contrer les détection d'OS [13], mais cela peut servir à bloquer des scans non-standard, donc limiter cette vilaine fuite d'information nécessaire à une bonne détection.

Au contraire de *fingerprint scrubber* (nous y viendrons), la directive *scrub* de *Packet Filter* n'uniformise pas les paquets pour tous les réseaux protégés, elle se contente de vérifier la validité de ceux-ci par rapport à certains tests de normalisation, en détruisant (c'est à dire que le datagramme IP est simplement détruit, et que aucune réponse n'est expédiée à la source émettrice de celui-ci) ou en modifiant les datagrammes si non conformes.

Examinons les principales fonctionnalités de normalisation de paquets implémentées dans *Packet Filter* de OpenBSD 3.5. Dans */usr/src/sys/net/pf_norm.c*, les fonctions *pf_normalize_ip()*, *pf_normalize_tcp()*, et *pf_normalize_tcpopt()* (nous nous intéressons dans cet article a IPv4, mais *Packet Filter* implémente aussi la normalisation pour IPv6).

- *pf_normalize_ip()*
    1. Si la taille de l'en-tête IP est invalide, le datagramme est détruit.

2. Les datagrammes IP fragmentés sont reconstruits directement par le *firewall* : de cette manière les machines protégées ne recoivent que des datagrammes complets, et un outil basé par exemple sur l'identification de l'algorithme de réassemblage IP ne fonctionnera pas (NDLA : je n'ai pas connaissance d'un outil tel que celui-ci, mais cela est possible, au minimum pour identifier la famille du système d'exploitation [14]). Ce mécanisme est un bon mécanisme d'uniformisation de paquets. Voir l'article de Mr Hartmeier concernant le réassemblage IP [15].

- *pf_normalize_tcp()*
  1. Si la taille de l'en-tête TCP est invalide, le segment TCP sera détruit.
  2. Certaines combinaisons de *flag*s TCP sont invalides. Ces segments TCP seront détruits. Un exemple intéressant : si il n'y a pas de *flag* ACK, mais qu'il y a un *flag* FIN, URG ou PSH, le segment TCP est détruit. Ainsi, certains probes de *nmap* sont bloqués (ils n'obtiennent pas de réponse).
  3. D'autres combinaisons de *flag*s TCP sont aussi invalides, mais sont ajustés plutot que détruits. Exemple : SYN+FIN, le FIN est enlevé.
  4. Le *flag* URG est traité d'une certaine manière : si il y a un *urgent pointer* dans le segment TCP, mais pas de *flag* URG, le segment est réécrit en enlevant le *urgent pointer* (le champ est mis à 0 dans l'en-tête TCP.

- *pf_normalize_tcpopt()*
  1. Seule l'option TCP *MSS* (*Maximum Segment Size*) est gérée. Si la directive `max-mss` est utilisée au coté de `scrub` dans le fichier de règles *Packet Filter*, la valeur de cette option est ajustée à la valeur configurée.

Bien sûr, lors de la réécriture des paquets, le *checksum* est recalculé, et il y a donc un impact en terme de performance, même si assez faible.

Il manque de nombreuses fonctionalités de normalisation, mais cela réduit quand même le nombre possible de probes servant à obtenir la fuite d'information nécessaire aux outils de *fingerprinting*. Pour connaître d'autres normalisation a appliquer, se réferrer à [13]. Probablement que celles-ci seront implémentées prochainement dans *Packet Filter*.

On peut également noter que *nmap* détecte l'utilisation de la directive `scrub` dans OpenBSD *Packet Filter*. Signalons que le code de *pf_norm.c* a évolué au cours des versions de *Packet Filter*, induisant très probablement des différences détectables pour une identification plus précise (NDLA : si un lecteur un tant soit peu motivé se penchait là-dessus, merci de me tenir informé).

Les tests contre FreeBSD

Reprenons notre FreeBSD personnalisé via les *sysctl*s décrit précédemment (en remettant à 1 la variable définissant le comportement RFC1323), et faisons passer le scan *nmap* par une machine sous OpenBSD 3.5 avec *Packet Filter*. Le dispositif de filtrage ne laisse passer que les connexions vers le port 22, en faisant du *stateful inspection*, de la normalisation de paquets, ainsi que de la réécriture d'*ISN*, et tout le reste est détruit.

```
agathon2# nmap -p 22,23 -O -vv --osscan_guess -P0 fbsd49
[..]
PORT   STATE    SERVICE
22/tcp open     ssh
23/tcp filtered telnet
Device type: general purpose
Running (JUST GUESSING) : FreeBSD 4.X|5.X (90%), Microsoft Windows 2003/.NET|NT/
2K/XP (90%), IBM AIX 4.X (88%)
Aggressive OS guesses: FreeBSD 4.7 - 4.8-RELEASE (90%), FreeBSD 4.9 - 5.1 (90%),
\
  Microsoft Windows Server 2003 (90%), Microsoft Windows 2000 SP3 (90%), IBM AIX
```

```
4.3.2.0-4.3.3.0 on an IBM RS/* (88%)
No exact OS matches for host (test conditions non-ideal).
TCP/IP fingerprint:
SInfo(V=3.50%P=i386-unknown-freebsd4.9%D=4/20%Time=3EA2786A%O=22%C=-1)
TSeq(Class=TR%IPID=I%TS=100HZ)
T1(Resp=Y%DF=Y%W=4000%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=N)
T4(Resp=N)
T5(Resp=N)
T6(Resp=N)
T7(Resp=N)
PU(Resp=N)
[..]

agathon2# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p
udp:514:open fbsd49
[..]
[+] Primary guess:
[+] Host 192.168.0.52 Running OS: "NetBSD 1.5.3" (Guess probability: 50%)
[+] Other guesses:
[+] Host 192.168.0.52 Running OS: "NetBSD 1.5.2" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.5.1" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.5" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.4.3" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.4.2" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.4.1" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.4" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.3.3" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.4" (Guess probability: 50%)
[..]
```

Maintenant, en désactivant la normalisation de paquets :

```
agathon2# nmap -p 22,23 -O -vv --osscan_guess -P0 fbsd49
[..]
PORT   STATE    SERVICE
22/tcp open     ssh
23/tcp filtered telnet
Device type: general purpose
Running: FreeBSD 4.X|5.X
OS details: FreeBSD 4.9 - 5.1
OS Fingerprint:
TSeq(Class=TR%IPID=I%TS=100HZ)
T1(Resp=Y%DF=Y%W=4000%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=4000%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(Resp=N)
T5(Resp=N)
T6(Resp=N)
T7(Resp=N)
PU(Resp=N)
[..]

agathon2# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p
udp:514:open fbsd49
[..]
[+] Primary guess:
[+] Host 192.168.0.52 Running OS: "NetBSD 1.5.3" (Guess probability: 50%)
[+] Other guesses:
[+] Host 192.168.0.52 Running OS: "NetBSD 1.5.2" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.5.1" (Guess probability: 50%)
```

```
[+] Host 192.168.0.52 Running OS: "NetBSD 1.5" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.4.3" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.4.2" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.4.1" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.4" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "NetBSD 1.3.3" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.4" (Guess probability: 50%)
[..]
```

On voit bien ici que le normalisation de paquets est utile : *nmap* obtient une réponse à un test de plus sans celle-ci (donc 2 tests réussis au lieu d'1), et cela est suffisent pour avoir FreeBSD dans la liste des OS possibles.

Un autre test en mettant *net.inet.tcp.rfc1323* à 0, et normalisation de paquets activée nous donne :

```
agathon2# nmap -p 22,23 -O -vv --osscan_guess -P0 fbsd49
[..]
PORT   STATE    SERVICE
22/tcp open     ssh
23/tcp filtered telnet
Device type: general purpose
Running (JUST GUESSING) : IBM AIX 4.X (90%), Amiga AmigaOS (87%)
Aggressive OS guesses: IBM AIX 4.2.X-4.3.3.0 (90%), IBM AIX 4.3.2.0-4.3.3.0 on
an IBM RS/* (88%), \
  IBM AIX 4.3 (88%), AmigaOS Miami 3.0 (87%), AmigaOS Miami 3.1-3.2 (87%)
No exact OS matches for host (test conditions non-ideal).
TCP/IP fingerprint:
SInfo(V=3.50%P=i386-unknown-freebsd4.9%D=4/20%Time=3EA279CA%O=22%C=-1)
TSeq(Class=TR%IPID=I%TS=U)
T1(Resp=Y%DF=Y%W=4000%ACK=S++%Flags=AS%Ops=M)
T2(Resp=N)
T3(Resp=N)
T4(Resp=N)
T5(Resp=N)
T6(Resp=N)
T7(Resp=N)
PU(Resp=N)
[..]

agathon2# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p
udp:514:open fbsd49
[..]
[+] Primary guess:
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.3" (Guess probability: 50%)
[+] Other guesses:
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.2" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.1.1" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.0" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "Microsoft Windows NT 4 Workstation Service
Pack 6a" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "Microsoft Windows NT 4 Workstation Service
Pack 5" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "Microsoft Windows NT 4 Workstation Service
Pack 4" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 3.5.1" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "Foundry Networks Device (Big/Net/Fast Iron)
Software Version 07.6.01BT51" \
  (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "Foundry Networks Device (Big/Net/Fast Iron)
Software Version 07.5.05KT53" \
  (Guess probability: 50%)
[..]
```

Puis sans normalisation :

```
agathon2# nmap -p 22,23 -O -vv --osscan_guess -P0 fbsd49
[..]
PORT   STATE    SERVICE
22/tcp open     ssh
23/tcp filtered telnet
Device type: general purpose
Running: Amiga AmigaOS
OS details: AmigaOS Miami 3.0, AmigaOS Miami 3.1-3.2
OS Fingerprint:
TSeq(Class=TR%IPID=I%TS=U)
T1(Resp=Y%DF=Y%W=4000%ACK=S++%Flags=AS%Ops=M)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=4000%ACK=S++%Flags=AS%Ops=M)
T4(Resp=N)
T5(Resp=N)
T6(Resp=N)
T7(Resp=N)
PU(Resp=N)
[..]

agathon2# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p
udp:514:open fbsd49
[..]
[+] Primary guess:
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.3" (Guess probability: 50%)
[+] Other guesses:
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.2" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.1.1" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 4.0" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "Microsoft Windows NT 4 Workstation Service
Pack 6a" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "Microsoft Windows NT 4 Workstation Service
Pack 5" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "Microsoft Windows NT 4 Workstation Service
Pack 4" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "FreeBSD 3.5.1" (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "Foundry Networks Device (Big/Net/Fast Iron)
Software Version 07.6.01BT51" \
  (Guess probability: 50%)
[+] Host 192.168.0.52 Running OS: "Foundry Networks Device (Big/Net/Fast Iron)
Software Version 07.5.05KT53" \
  (Guess probability: 50%)
[..]
```

Ces résultats sont intéressants, *nmap* n'a pas dans sa liste FreeBSD du tout, et *Xprobe* hésite entre plusieurs systèmes bien différents.

Les tests contre Linux

Maintenant, les même tests que dans la section 4.3. avec les modifications des éléments de la *MIB* Linux (RFC1323=0, taille de la fenêtre non modifiée), avec le même dispositif de filtrage (normalisation de paquets, réécriture de l'*ISN*, et *stateful inspection*) :

```
agathon2# nmap -p 22,23 -O -vv --osscan_guess -P0 rh72
[..]
PORT   STATE    SERVICE
22/tcp open     ssh
23/tcp filtered telnet
Device type: general purpose
```

```
Running (JUST GUESSING) : IBM AIX 4.X (88%)
Aggressive OS guesses: IBM AIX 4.3.2.0-4.3.3.0 on an IBM RS/* (88%), IBM AIX 4.3
(88%)
No exact OS matches for host (test conditions non-ideal).
TCP/IP fingerprint:
SInfo(V=3.50%P=i386-unknown-freebsd4.9%D=4/20%Time=3EA29EA4%O=22%C=-1)
TSeq(Class=TR%IPID=Z%TS=U)
T1(Resp=Y%DF=Y%W=16D0%ACK=S++%Flags=AS%Ops=M)
T2(Resp=N)
T3(Resp=N)
T4(Resp=N)
T5(Resp=N)
T6(Resp=N)
T7(Resp=N)
PU(Resp=N)
[..]

agathon2# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p
udp:514:open rh72
[..]
[+] Primary guess:
[+] Host 192.168.0.60 Running OS: "Microsoft Windows NT 4 Workstation Service
Pack 6a" (Guess probability: 44%)
[+] Other guesses:
[+] Host 192.168.0.60 Running OS: "FreeBSD 3.3" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "Microsoft Windows NT 4 Workstation Service
Pack 4" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 3.5.1" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 4.3" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 4.1.1" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 4.1.1" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 4.3" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 3.5.1" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "Microsoft Windows NT 4 Workstation Service
Pack 4" (Guess probability: 44%)
[..]
```

Puis sans normalisation :

```
agathon2# nmap -p 22,23 -O -vv --osscan_guess -P0 rh72
[..]
PORT   STATE    SERVICE
22/tcp open     ssh
23/tcp filtered telnet
Device type: general purpose
Running (JUST GUESSING) : Linux 2.4.X|2.6.X (93%), Microsoft Windows 95/98/ME|
NT/2K/XP (85%)
Aggressive OS guesses: Linux 2.4.18 - 2.4.19 w/o tcp_timestamps (93%), Linux
2.4.7 (X86) (93%), \
  Linux 2.6.0-test5-love3 (X86) (93%), Linux 2.4.22 (X86) w/grsecurity patch and
with timestamps disabled (86%), \
  Microsoft Windows NT 3.51 SP5, NT 4.0 or 95/98/98SE (85%)
No exact OS matches for host (test conditions non-ideal).
TCP/IP fingerprint:
SInfo(V=3.50%P=i386-unknown-freebsd4.9%D=4/20%Time=3EA29E42%O=22%C=-1)
TSeq(Class=TR%IPID=Z%TS=U)
T1(Resp=Y%DF=Y%W=16D0%ACK=S++%Flags=AS%Ops=M)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=16D0%ACK=S++%Flags=AS%Ops=M)
T4(Resp=N)
T5(Resp=N)
T6(Resp=N)
```

```
T7(Resp=N)
PU(Resp=N)
[..]

agathon2# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p
udp:514:open rh72
[..]
[+] Primary guess:
[+] Host 192.168.0.60 Running OS: "Microsoft Windows NT 4 Workstation Service
Pack 6a" (Guess probability: 44%)
[+] Other guesses:
[+] Host 192.168.0.60 Running OS: "FreeBSD 3.3" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "Microsoft Windows NT 4 Workstation Service
Pack 4" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 3.5.1" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 4.3" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 4.1.1" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 4.1.1" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 4.3" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "FreeBSD 3.5.1" (Guess probability: 44%)
[+] Host 192.168.0.60 Running OS: "Microsoft Windows NT 4 Workstation Service
Pack 4" (Guess probability: 44%)
[..]
```

Tableau récapitulatif des résultats précédents, et de ceux non mentionnés (résultats des outils simplifiés pour raison de clarté) : Légende :

- 0 : aucune protection.
- B : FreeBSD *blackhole* activé.
- R : options TCP RFC1323 désactivées.
- W : taille de la fenêtre modifiée.
- F : filtrage (seul un port TCP est accessible, le reste est détruit).
- N : normalisation de paquets.
- S : *stateful inspection*.
- I : réécriture de l'*ISN*.

>

FreeBSD 4.9	0	B	B+R+W	B+W+F+S+I	B+W+F+N+S+I	B+W+F+S+I+R	B+W+F+S+I+R+N
nmap	FreeBSD 4.x	FreeBSD 4.x	AmigaOS, FreeBSD 4.x-5.x, AIX 4.x-5.x	FreeBSD 4.x-5.x	FreeBSD 4.x-5.x, Windows 2003-.NET-NT-2K-XP, AIX 4.x	AmigaOS Miami	AIX 4.x, AmigaOS?
Xprobe	FreeBSD 4.x	FreeBSD 4.x	Linux 2.0.x, FreeBSD 3.x-4.x	NetBSD 1.3.x-1.4.x-1.5.x, FreeBSD 4.x	NetBSD 1.3.x-1.4.x-1.5.x, FreeBSD 4.x	FreeBSD 3.x-4.x, Windows NT 4, Foundry Networks Device	FreeBSD 3.x-4.x, Windows NT 4, Foundry Networks Device

Linux 2.4.18	0	R	R+W	R+F+S+I	R+F+S+I+N
nmap	Linux 2.4.x-2.5.x	Linux 2.4.x	Linux 2.3.x à 2.6.x, TiniOS, Netware 4.x-5.x, Lexmark embedded, Solaris 2.6-7	Linux 2.4.x-2.6.x, Windows 95-98-ME-NT-2K-XP	AIX 4.x
Xprobe	Linux 2.4.x	Linux 2.4.x	FreeBSD 3.x-4.x, Linux 2.4.x	Windows NT 4, FreeBSD 3.x-4.x	Windows NT 4, FreeBSD 3.x-4.x

Note : les différences étranges dans les résultats de ces outils en fonction des mécanismes de protection appliqués peuvent s'expliquer par le fait que, pour certains systèmes d'exploitation, l'absence de réponse à un probe est un comportement connu, donc auquel on associe une signature.

Conclusion, on voit bien que la normalisation de paquets, associée au filtrage et à la personnalisation IP permet de bien masquer son OS, en limitant fortement la fuite d'information. Mais il y a des impacts quant aux performances, puisqu'il faut recalculer le *checksum* des en-têtes IP et TCP.

Il faut noter ici que de toute facon, utiliser la normalisation de paquets, un filtrage complet (ne laisser passer que ce qui sert à la fourniture du service voulu) associé à l'inspection de l'état est une bonne pratique de sécurité [16], il faut donc l'activer tout le temps pour protéger les réseaux.

Application de *patch*s à la pile IP

FreeBSD *fingerprint scrubber* [17]

Le *fingerprint scrubber* a pour but d'anonymiser une pile IP (plutôt un réseau de piles IP), non de l'usurper, ou pire de contrer *nmap* uniquement. De cette manière il est impossible de trouver un mécanisme de contournement, puisque les paquets sont réécrits de manière uniforme, indépendemment du type de probe. Il est à placer à l'entrée d'un réseau, pour uniformiser les signatures des machines à protéger. Ainsi, un réseau hétérogène se transformera en un réseau homogène, du point de vue IP. En fait, cela fonctionne comme un *reverse proxy* IP.

- IP *scrubbing*
    1. Réassemblage des fragments, l'algorithme utilisé étant celui de FreeBSD 2.2.8, avec quelques petites modifications.
    2. Changement des *flag*s IP, comme le champ *TOS* (*Type Of Service*) et des bits concernant la fragmentation. Pour ce qui est du *TOS*, certaines combinaisons sont invalides et sont donc enlevées. Le bit *don't fragment* est autommatiquement enlevé, mais comme cela casse le *PMTUD* (*Path MTU Discovery* [18]), une option existe pour le désactiver.
- ICMP *scrubbing*
    1. La copie des en-têtes IP contenues dans les messages ICMP d'erreur en retour (exemple : ICMP *port unreachable*) sont tronqués à 8 octets, puisque de nombreuses différences dans les implémentations IP font que l'OS peut être identifié par ce paramètre.
    2. Limitation du nombre de messages ICMP générés en réponse à des probes.
- TCP *scrubbing*
    1. Un mécanisme simplifié de *stateful inspection* est utilisé afin de ne laisser passer que les segments TCP en rapport avec une connexion établie. Le but étant ici de bloquer les scans de ports *stealth* souvent utilisés dans les préliminaires à l'OS *fingerprinting*.
    2. Changement de l'ordre des options TCP dans un format unique, les options connues de la pile de FreeBSD placées en premier, les inconnues ensuite. Ce paramètre est configurable par l'utilisateur. En effet, l'ordre des options TCP est un paramètre

important dans l'identification d'OS.
    3. Grace au mécanisme de *stateful inspection*, l'*ISN* peut également etre réécrit.
- Changement des algorithmes de retransmission, et du nombre de paquets émis
    1. Le principe est de changer l'algorithme de retransmission des paquets (dépendant bien sûr d'une implémentation TCP/IP). Cette option n'a pas été implémentée dans *fingerprint scrubber*, mais est évoquée dans l'article.
    2. Limitation du nombre de paquets générés, seule cette protection est implémentée, et concerne les messages ICMP de retour consécutifs à des scans.

Les résultats décrits dans le papier sont les suivants (une tentative de joindre par courriel les auteurs s'étant révélée infructueuse, nous n'avons pas pu tester par nous-même) : Machines du réseau cible : FreeBSD 2.2.8, Solaris 2.7 x86, Windows NT 4.0 SP3, et Linux 2.2.12. Sans le dispositif de *scrubbing*, *nmap* identifie tous les systèmes, et avec, il n'arrive même pas a avoir dans sa liste d'OS possibles le système scanné.

Mais le *fingerprint scrubber* est sûrement identifiable, puisque son comportement est statique. Ce n'est pas un problème, puisque son but est de bloquer les différences dans les implémentations IP, et cela marche (enfin, au moins sur le papier, étant donné qu'il ne fonctionne que sur FreeBSD 2.2.8). Donc, dans le pire des cas, on saura qu'il y a un FreeBSD 2.2.8 en coupure sur le réseau cible.

Linux *IP Personality* [19]

IP Personality est un *patch* pour les noyaux Linux 2.4.18 (et d'autres versions, voir le site). Il permet de paramétrer le comportement IP, c'est-à-dire de réécrire complètement les datagrammes via la table *mangle* de *netfilter* [20]. Une *target PERS* est ajoutée, et les fichiers de configuration écrivent leurs directives à cet endroit. C'est donc une syntaxe basée sur un langage spécifique dans un fichier de directives qui servira à modifier le comportement de la pile via cette table *mangle* de *netfilter*.

Il fonctionne de deux façons, l'une en coupure (comme *fingerprint scrubber*), l'autre en local sur la machine. Suivant l'utilisation, la configuration de *netfilter* diffère (via un *iptables* modifié). C'est-à-dire que les modifications appliquées aux datagrammes se font soit dans la chaine *PREROUTING*, soit dans la chaine *OUTPUT* (ou les deux). Mais il y a une limitation quant à son utilisation par la chaine *PREROUTING* [21]. En gros, il n'est capable de réécrire que les paquets TCP. C'est assez dommageable, mais si votre réseau ne fournit que des services sur TCP, et qu'une bonne politique de filtrage est appliquée (uniquement les services offerts sont accessibles), cela n'a aucun impact.

Il est livré avec quelques fichiers de configuration qui sont des règles écrites dans l'optique de contrer *nmap*. Ils sont écrits en fonctions de ses probes, de manière à lui répondre pour que la signature construite soit celle que l'on souhaite donner à sa pile. Mais nous verrons que cela se montre quand même efficace contre d'autres outils.

Mais deux gros problèmes surviennent avec cette approche d'usurpation d'identité :

1. Les règles sont écrites en fonction des probes de *nmap* : donc en changeant judicieusement ses probes, on peut contourner les modifications apportées aux réponses passant par la table *mangle*.
2. Le *spoofing* de piles IP faibles insère des vulnérabilités directement dans la pile, comme par exemple rendre le *blind TCP connection hijacking* [22] possible (voir `dreamcast.conf`). Le problème vient simplement du fait que l'on peut facilement connaitre le numéro *ISN* généré suivant.

En prenant le fichier *freebsd.conf* comme exemple, on voit clairement que le choix du placement (ou de leur présence) des options TCP est fondé sur celles demandées par *nmap* lors de ses probes. Bien sûr, on peut améliorer le fichier de configuration, mais l'approche d'usurpation d'identité est à déconseiller. En clair, surtout ne pas modifier les réponses en fonction de paramètres trop précis de

la requête IP, il faut être le plus générique possible (tout en respectant les RFCs).

```
[..]
    if (option(mss)) { /* nmap has mss on all of its pkts */
      if (listen) {
        if (flags(syn&ece) || flags(syn&fin&urg&push)) { /* nmap test 1 or 3 */
          set(df, 1);
          set(win, 0x403D);
          set(ack, this + 1);
          set(flags, ack|syn);
          insert(mss, this+1);
          insert(timestamp);
          copy(wscale);
          reply;
        }
[..]
```

Configurons maintenant la cible à tester pour ressembler à un FreeBSD :

```
rh72# iptables -t mangle -A PREROUTING -j PERS --local --tweak dst --conf
freebsd.conf
rh72# iptables -t mangle -A OUTPUT     -j PERS --local --tweak src --conf
freebsd.conf
```

Lancons nos tests *nmap* et *Xprobe* (il n'y a ici aucun filtrage) :

```
obsd32# nmap -P0 -vv -p 22,23 -O --osscan_guess rh72
[..]
PORT   STATE  SERVICE
22/tcp open   ssh
23/tcp closed telnet
Device type: general purpose
Running: FreeBSD 2.X|3.X|4.X
OS details: FreeBSD 2.2.1 - 4.1
OS Fingerprint:
TSeq(Class=RI%gcd=1%SI=2602%IPID=I%TS=100HZ)
T1(Resp=Y%DF=Y%W=403D%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=403D%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(Resp=Y%DF=N%W=4000%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=F%RIPCK=F%UCK=0%ULEN=134%DAT=E)

obsd32# xprobe2 -p tcp:22:open -p tcp:23:closed -p udp:50:closed -p udp:111:open
rh72
[..]
[+] Primary guess:
[+] Host 192.168.0.60 Running OS: "FreeBSD 2.2.7" (Guess probability: 82%)
[+] Other guesses:
[+] Host 192.168.0.60 Running OS: "FreeBSD 2.2.8" (Guess probability: 82%)
[+] Host 192.168.0.60 Running OS: "OpenBSD 3.0" (Guess probability: 82%)
[+] Host 192.168.0.60 Running OS: "OpenBSD 3.1" (Guess probability: 82%)
[+] Host 192.168.0.60 Running OS: "OpenBSD 3.2" (Guess probability: 82%)
[+] Host 192.168.0.60 Running OS: "OpenBSD 3.3" (Guess probability: 82%)
[+] Host 192.168.0.60 Running OS: "NetBSD 1.5.2" (Guess probability: 79%)
[+] Host 192.168.0.60 Running OS: "NetBSD 1.4" (Guess probability: 79%)
[+] Host 192.168.0.60 Running OS: "NetBSD 1.4.1" (Guess probability: 79%)
[+] Host 192.168.0.60 Running OS: "NetBSD 1.5.1" (Guess probability: 79%)
[..]
```

Le dispositif se montre donc très efficace, même si l'outil utilisé n'est pas *nmap*. Mais *hping* peut identifier le système cible :

```
obsd32# hping -c 1 -S -p 22 rh72
HPING rh72 (ne3 192.168.0.60): S set, 40 headers + 0 data bytes
len=46 ip=192.168.0.60 ttl=64 DF id=35789 sport=22 flags=SA seq=0 win=5840
rtt=1.6 ms
```

C'est la taille de la fenêtre TCP qui révèle la vrai nature de l'OS, une taille de 5840 est caractéristique d'un Linux 2.4. Parfois, les tests les plus simples se trouvent être les plus efficaces 
.

Regardons un exemple de transformation de la pile. Nous changeons simplement la taille de la fenêtre TCP. Voici à quoi correspond ce fichier :

```
rh72# cat modif-win.conf
id "ModifWin";

tcp {
  incoming yes;
  outgoing yes;
  max-window 4096;
}
```

Nous appliquons maintenant ce *mangling* dans *netfilter* (vous noterez donc que c'est une modification pour les accès à la machine locale uniquement) :

```
rh72# iptables -t mangle -A PREROUTING -j PERS --local --tweak dst --conf modif-
win.conf
rh72# iptables -t mangle -A OUTPUT     -j PERS --local --tweak src --conf modif-
win.conf
```

Un probe hping :

```
# hping -S -p 22 -c 1 rh72
HPING rh72 (ne3 192.168.0.60): S set, 40 headers + 0 data bytes
len=46 ip=192.168.0.60 ttl=64 DF id=0 sport=22 flags=SA seq=0 win=4096 rtt=1.7
ms
```

La taille de la fênetre fait donc bien 4096. Si l'on avait appliqué cette modification en *src* dans la chaîne *PREROUTING* également (en enlevant aussi le paramètre `--local`), et que ce Linux faisait office de routeur filtrant, toutes les machines derrière auraient également bénéficié de ce changement. Pour une explication détaillée des possibilités de l'outil, consulter la documentation qui est très bien faite [23].

Grâce à la puissance du moteur de règles, on peut écrire des fichiers de configuration très génériques pour contrer tout type de probes de détection d'OS. Ainsi, on peut écrire un fichier de règles pour faire fonctionner *IP Personality* presque comme le *fingerprint scrubber* de FreeBSD 2.2.8 (*IP Personality* ayant les limitations évoquées plus haut).

Nous n'allons pas fournir de fichier de configuration parfait, puisqu'il n'y en a pas. C'est a chacun d'en créer un, de manière a ce qu'il soit unique. En effet, l'utilisation d'un fichier de configuration connu permettrait d'identifier le routeur filtrant comme utilisant *IP Personality*, et donc de savoir qu'il tourne sous Linux 2.4. Même si ce n'est pas forcément très grave, il vaut mieux limiter la fuite d'information quant à la topologie de son réseau, c'est une couche supplémentaire de sécurité (voir section 1.).

Puisque l'utilité maximale pour *IP Personality* est d'être sur un routeur filtrant, on peut le coupler avec une vraie bonne politique de filtrage, voire un dispositif de normalisation de paquets. Ainsi, on obtient des piles IP pas mal protégées contre l'identification distante.

Conclusion

La personnalisation IP donne un niveau acceptable de protection contre la détection d'OS, mais a le problème de nécessiter une configuration par machine, et par OS. Mais nous n'avons testé ici que deux systèmes ouverts, il est possible également sous Windows de modifier le comportement IP, voici quelques références [24][25], pour les lecteurs intéressés par l'expérimentation.

La solution *IP Personality* est intéressante, mais n'est pas des plus aisée à mettre en oeuvre, puisqu'il est conseillé d'écrire son propre fichier de règles (avec tous les tests de performances que cela implique) pour avoir une signature vraiment anonyme.

La solution définitive (même si non testée dans cet article) est donc le *fingerprint scrubber*, mais hélas, il n'y a pas de code publique disponible. Je lance donc personnellement un appel aux lecteurs, à savoir réaliser une implémentation sur les systèmes d'exploitations ouverts suivant : Linux 2.6.x, FreeBSD 5.x, OpenBSD 3.x (enfin, si vous avez le temps, hein).

Il reste donc du travail dans le domaine du masquage de signature IP, même si la théorie existe, et semble fonctionner. Concernant une implémentation commerciale, nous n'avons pas connaissance à l'heure actuelle d'un produit faisant du *fingerprint scrubbing*.

Références

[1] Prise d'empreinte active des systèmes d'exploitation
http://www.gomor.org/article/misc7
[2] nmap
http://www.insecure.org/nmap/
[3] Xprobe
http://www.sys-security.com/html/projects/X.html
[4] ring
http://www.intranode.com/fr/pg/tech/resource_fr.htm
[5] iplog
http://ojnk.sourceforge.net/
[6] antinmap
http://www.kofein.com.ua/hack/s/162.html
[7] Fingerprint Fucker
http://www.s0ftpj.org/en/site.html
[8] FreeBSD blackhole(4) man page
http://www.freebsd.org/cgi/man.cgi?query=blackhole&apropos=0&sektion=0&manpath=FreeBSD+4.9-RELEASE&format=html
[9] The Art of Port Scanning
http://www.insecure.org/nmap/nmap_doc.html
[10] Remote OS detection via TCP/IP Stack FingerPrinting
http://www.insecure.org/nmap/nmap-fingerprinting-article.html
[11] ICMP Usage In Scanning
http://www.sys-security.com/html/projects/icmp.html
[12] /usr/src/linux-2.4/Documentation/networking/ip-sysctl.txt
[13] Network Intrusion Detection Evasion, Traffic Normalization, and End-to-End Protocol Semantics
http://www.icir.org/vern/papers/norm-usenix-sec-01-html/index.html
[14] Insertion, Evasion, and Denial of Service : Eluding Network Intrusion Detection
http://www.netsys.com/library/papers/ptacek-newsham-evasion98.pdf
[15] Design and Performance of the OpenBSD Stateful Packet Filter (pf)
http://www.usenix.org/events/usenix02/tech/freenix/full_papers/hartmeier/hartmeier_html/
[16] Building Internet Firewalls - O'Reilly and Associates
[17] Defeating TCP/IP Stack Fingerprinting
http://www.usenix.org/publications/library/proceedings/sec2000/smart.html
[18] Path MTU Discovery
http://www.rfc-archive.org/getrfc.php?rfc=1191

[19] IP Personality
http://ippersonality.sourceforge.net/
[20] Le filtrage de paquets sous Linux
http://www.miscmag.com/articles/index.php3?page=107
[21] IP Personality limitations
http://ippersonality.sourceforge.net/doc/ippersonality-fr-2.html
[22] Failles IP
http://www.laurentconstantin.com/common/utilordi/b2introreseau/failles_ip.doc
[23] IP Personality documentation
http://ippersonality.sourceforge.net/doc/ippersonality-fr.html
[24] TCP/IP and NBT Configuration Parameters for Windows 2000 or Windows NT
http://support.microsoft.com/support/kb/articles/Q120/6/42.asp
[25] Windows 2000/XP TCP/IP tuning
http://www.voodoobox.nl/tweaks/wintweak.html

Patrice "GomoR" Auffret - <gomor[at]gomor.org>