Prise d'empreinte active des systèmes d'exploitation

Généralement appelé *OS fingerprinting* actif (OSFP dans cet article). Le but de l'OSFP est de détecter à distance quel système d'exploitation une machine utilise, par l'analyse des datagrammes IP qu'elle construit.

Le principe d'identification, ses limites

Puisque les RFCs décrivant l'implémentation d'une pile IP ne définissent pas de manière stricte comment les paquets IP doivent être construits, les programmeurs ont créé des différences dans les piles, sans vraiment le vouloir. L'OSFP actif est donc l'identification d'un système d'exploitation relié à un réseau par l'analyse de ces différences.

Le principe de fonctionnement est le suivant :

- envoi de datagrammes IP générant des différences significatives entre les implémentations ;
- construction d'une signature identifiant le système visé ;
- puis comparaison à une base de signatures connues pour trouver l'empreinte qui correspond au système testé.

Les datagrammes à envoyer qui génèrent ces différences sont à choisir judicieusement, par l'expérimentation. Certains paquets permettent de conclure directement sur la nature du système visé, tandis que d'autres écarteront simplement certaines possibilités quant au système visé.

Cette méthode d'OSFP à ses limites :

- certaines piles IP sont très utilisées (ex : 4.3BSD), et il sera impossible de faire la différence entre une imprimante utilisant cette pile, et un routeur (par exemple) ;
- dans les environnements fortement filtrés, certains tests ne pourront être joués, puisque bloqués par les dispositifs mis en place ;
- de même, les dispositifs de normalisation de paquets (ex : *Packet Filter* dans OpenBSD) bloqueront directement les paquets non-standards, qui apportent des précisions sur la nature du système testé ;
- et enfin, il peut se produire des collisions dans les signatures, même si bien souvent, cela signifie que nous avons affaire à une pile très commune.

Les utilisations possibles

Un réseau d'entreprise à souvent besoin de savoir quelles sont les machines non souhaitées, et l'identification des systèmes de ce réseau montrera par exemple que telle machine n'est pas sous un OS autorisé.

Lors d'audits de sécurité distants, ou de tests d'intrusion, l'identification des systèmes est une phase cruciale. Par exemple, pour exécuter un *exploit* (dans le cas de tests intrusifs), il est nécessaire de connaître précisément la cible pour choisir un *shellcode* (mais l'OSFP actif ne permet pas de détecter l'architecture, il faut trouver un autre moyen pour cela).

Une autre utilisation est celle à des fins de statistiques, de recensement. Certaines sociétés sont spécialisées dans ce domaine, comme NetCRAFT? , qui identifie les types de serveurs HTTP utilisés, ainsi que les systèmes d'exploitation de millions de machines sur Internet.

Les machines cibles

Les machines cibles de cet article sont des machines installées en standard (*out-of-the-box*). Aucunes modifications n'ont été faite quant aux paramètres de la pile IP.

- 192.168.0.1 : OpenBSD 3.0
- 192.168.0.10 : FreeBSD 4.7
- 192.168.0.20 : Solaris 8
- 192.168.0.30 : Solaris 7
- 192.168.0.51 : Windows 2000
- 192.168.0.55 : Linux 2.2
- 192.168.0.60 : Linux 2.4
- 192.168.1.10 : NetBSD 1.6

Figures

Pour bien comprendre comment des différences d'implémentations peuvent apparaître, voici les en-têtes des paquets qui seront analysés par la suite. On voit bien que l'en-tête TCP est pleine de possibilités.

Figure 1 : En-tête IP (RFC791)

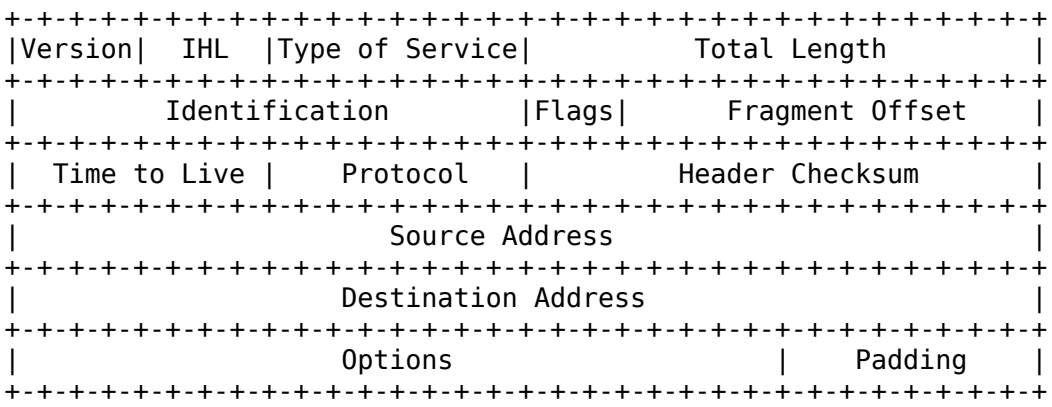


Figure 2 : En-tête TCP (RFC793)

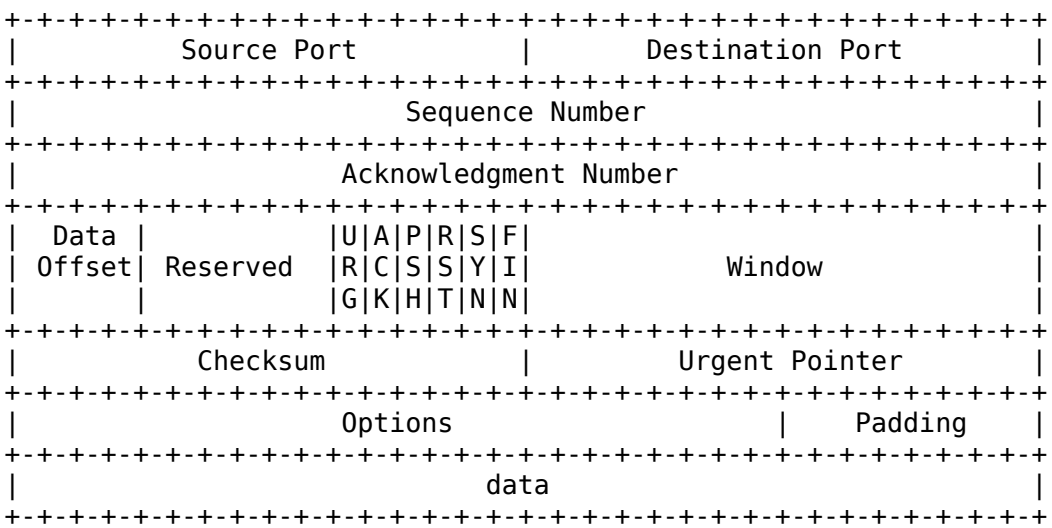
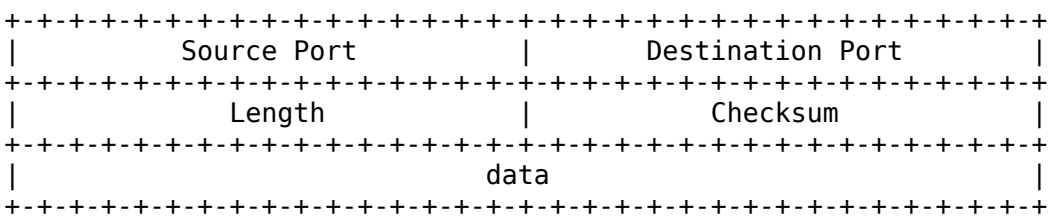


Figure 3 : En-tête UDP (RFC768)



Analyse des segments TCP

L'analyse des segments TCP est la méthode qui apporte le plus de facilité pour identifier un OS, puisque les paquets TCP comportent de nombreux champs, qui peuvent être construits de plusieurs manières.

Les tests décris ci-après donnent des résultats sur des systèmes installés par défaut, c'est-à-dire dont les paramètres de la pile IP n'ont pas été modifiés.

Taille de la fenêtre initiale

Lors de l'établissement d'une connection TCP, chaque partie TCP envoie un segment TCP avec la taille de la fenêtre qu'il aimerait utiliser. Cette valeur est laissée au choix du programmeur d'une pile IP, et varie grandement d'un OS à l'autre. Voici comment tester cette valeur :

```
# hping -c 1 -p 22 -S -w 512 192.168.0.1
HPING silenoz (vr0 192.168.0.1): S set, 40 headers + 0 data bytes
len=46 ip=192.168.0.1 flags=SA DF seq=0 ttl=128 id=5626 win=16384 rtt=7.7 ms
```

On voit ici que l'OS testé a une taille de fenêtre TCP initiale de 16384. Les systèmes qui ont cette taille de fenêtre sont ou bien de type BSD (FreeBSD, OpenBSD, NetBSD), ou bien de type AIX. D'autres valeurs identifient directement, et l'OS, et sa version.

Voici un tableau de quelques valeurs courantes :

AIX	**HP-UX**	**Linux 2.2**	**Solaris 7**	**OpenBSD 3.0**	**FreeBSD 4.7**	**Windows 2000**
16384	32768	32696	9112	16384	57344	16616

Options TCP supportées

Il existe un certain nombre d'options TCP. Trois sont obligatoires dans les implémentations TCP (d'après les RFCs) : MSS (*Maximum Segment Size*), nop (*no operation*), et end-of-options. Tous les systèmes que nous avons testés supportent effectivement ces options.

Voici presque toutes les options TCP implémentables (liste complète, voir RFC1700) :

- 0x00 : *End of options* (RFC793) ;
- 0x01 : *No operation* (RFC793) ;
- 0x02 : *MSS* (RFC793) ;
- 0x03 : *Window scale* (RFC1323) ;
- 0x04 : *Selective aknowlegment permitted* (RFC2018) ;
- 0x05 : *Selective aknowlegment option* (RFC2018) ;
- 0x06 : *echo-request* (RFC1072) ;
- 0x07 : *echo-reply* (RFC1072) ;
- 0x08 : *Timestamp* (RFC1323) ;
- 0x09 : *Partial Order Connection Permitted* (RFC1693) ;
- 0x0a : *Partial Order Service Profile* (RFC1693) ;
- 0x0b : *CC Connection Count* (RFC1644) ;
- 0x0c : *CC New* (RFC1644) ;
- 0x0d : *CC Echo* (RFC1644) ;
- 0x0e : *TCP Alternate Checksum Request* (RFC1146) ;
- 0x0f : *TCP Alternate Checksum Data* (RFC1146).

Certaines de ces options sont de nos jours obsolètes. Nous allons tester, pour l'exemple, si le système cible connaît l'option 0x03.

```
  # sendip -p ipv4 -is 192.168.0.55 -p tcp -td 22 -towscale 0 -tonop
192.168.0.20
```

Le tcpdump montre en retour :

```
  21:38:36.872657 192.168.0.20.22 > 192.168.0.55.0: S [tcp sum ok]
2564039607:2564039607(0) \
    ack 116380527 win 24656 <nop,wscale 0,mss 1460> (DF) (ttl 64, id 60679, len
48)
```

Ici, l'option est bien supportée.

	Linux 2.0	**FreeBSD 2.2.6**	**FreeBSD 4.7**	**Windows 2000**	**Solaris 8**
Support	Non	Non	Oui	Oui	Oui

Il est plus ou moins facile de faire des tests pour chacune de ces options. Le test des options TCP supportées est vraiment celui qui apporte le plus de précision quant à l'OS ciblé. On peut même imaginer un outil d'OSFP fondé uniquement sur le test du support de ces options.

Valeur du MSS

En envoyant un paquet SYN sans options TCP à un port ouvert, tous les systèmes cibles testés retournent un paquet SYN+ACK avec une option MSS, et une valeur pour celui-ci qui varie d'un système à l'autre.

```
  # hping -c 1 -p 23 -S 192.168.0.30
```

Sortie tcpdump :

```
  21:50:01.222976 192.168.0.30.23 > 192.168.0.10.2974: S [tcp sum ok]
386544370:386544370(0) \
    ack 1762951200 win 9112 <mss 536> (DF) (ttl 255, id 15767, len 44)
```

	Windows 2000	**Solaris 7**	**Solaris 8**	**AIX**
MSS	1460	536	1460	512

Analyse des messages ICMP

Nous n'allons pas ici refaire le papier de Ofir Arkin[1] portant sur l'usage du protocole ICMP à des fin de récupération d'informations. Nous allons seulement montrer qu'il est possible de différencier des systèmes en utilisant les messages ICMP (RFC792).

Messages ICMP request supportés

Il existe quatre types de messages ICMP de collecte d'informations diverses sur un système cible : echo-request, timestamp-request, netmask-request, et information-request.

Il suffit d'expédier ces quatre requêtes vers la cible, et de regarder si l'on obtient une réponse.

```
# sing -c 1 -tstamp 192.168.1.10
SINGing to agathon (192.168.1.10): 20 data bytes
20 bytes from 192.168.1.10: seq=0 ttl=254 TOS=0 diff=1185641
```

	Solaris 8	**Linux 2.4**	**Windows 2000**	**Windows NT 4.0**
echo-request supporté	Oui	Oui	Oui	Oui
timestamp-request supporté	Oui	Oui	Oui	Non
netmask-request supporté	Oui	Non	Non	Oui
information-request supporté	Non	Non	Non	Non

Données retournées dans les messages ICMP erreur

Lorsque qu'une erreur intervient sur un réseau IP, et que c'est une erreur qui est gérée par IP, un message ICMP est retourné à la source de la requête fautive. A des fins d'analyse par l'émetteur, une partie du contenu du datagramme ayant provoqué celle-ci est copié dans la section data du message ICMP signalant un problème. Il est écrit dans la RFC792 que seulement 28 octets du datagramme originel sont à recopier, mais les programmeurs en ont décidé autrement (pour une fois, ca vient pas des RFCs).

Le test suivant sert à récupérer quel est le nombre d'octet qu'un système copie dans ses messages ICMP d'erreur. On expédie un datagramme UDP de grosse taille à un port fermé pour avoir en retour un ICMP de type *destination unreachable* et de code *port unreachable*. La valeur 160 est arbitraire.

```
# hping -2 -c 1 -p 1234 -d 160 -E /dev/random 192.168.0.20
HPING caesar (vr0 192.168.0.20): udp mode set, 28 headers + 160 data bytes
ICMP Port Unreachable from ip=192.168.0.20 name=caesar.enslaved.lan
```

Tcpdump nous montre :

```
  18:09:32.185568 192.168.0.20 > 192.168.0.10: icmp: 192.168.0.20 udp port
1234 \
    unreachable (DF) (ttl 255, id 59094, len 112)
```

Soit 112 moins 20 pour IP, et moins 8 pour ICMP, pour un total de 84 octets de données dans la section data du message ICMP retourné.

OpenBSD 3.0	**Linux 2.4**	**Solaris 8**	**Windows 2000**
28	188	84	28

Analyse des datagrammes UDP

Les datagrammes UDP en eux-mêmes n'offrent pas suffisamment de souplesse dans l'implémentation pour introduire des différences (il y a très peu de champs, voir Figure 3). En revanche, certaines différences dans les implémentations IP ne sont visibles qu'en usant de datagrammes UDP (ou en tout cas donnent de meilleurs résultats en UDP).

Analyse des en-têtes IP

Le bit DF

En envoyant un datagramme a un port UDP ou aucune application ne tourne, l'OS destination génère un paquet ICMP de type *destination unreachable* et de code *port unreachable*. Certains systèmes ajoutent le bit DF au paquet ICMP généré en retour. Le test suivant envoie un datagramme sans le bit DF :

```
  # hping -2 -c 1 -p 1234 192.168.0.1
  HPING silenoz (vr0 192.168.0.1): udp mode set, 28 headers + 0 data bytes
  ICMP Port Unreachable from ip=192.168.0.1 name=silenoz.enslaved.lan
```

La réponse lue avec tcpdump nous montre que DF n'est pas mis en réponse. Mais certains systèmes le mette automatiquement.

	OpenBSD 3.0	**Windows 2000**	**Solaris 7**
Bit DF	0	0	1

L'IP ID

Certaines différences apparaissent dans l'en-tête IP selon que le paquet est UDP ou TCP (IP ID différent de 0 ou non). Par exemple, les systèmes Linux 2.4 ne mettent pas de valeur pour l'IP ID lorsqu'ils expédient un segment TCP avec le bit DF mis à 1. La raison est que l'IP ID ne sert que pour réassembler les datagrammes IP fragmentés.

Test en TCP, avec le bit DF mis à un :

```
  # hping -c 1 -y -p 80 192.168.0.55
  HPING 192.168.0.55 (vr0 192.168.0.55): S set, 40 headers + 0 data bytes
  len=44 ip=192.168.0.55 flags=SA DF seq=0 ttl=64 id=843 win=32696 rtt=22.9 ms
```

Test en UDP, le bit DF est laissé à zero :

```
# hping -2 -c 1 -p 22 192.168.0.1
```

Réponse à lire avec tcpdump.

	Linux 2.2	**Linux 2.4**	**OpenBSD 3.0**
TCP avec DF=1	IP_ID!=0	IP_ID =0	IP_ID!=0
UDP avec DF=0	IP_ID!=0	IP_ID!=0	IP_ID!=0

TTL par défaut

Un autre test est celui qui récupère la valeur par défaut du TTL d'un paquet IP. Ce test doit être accompli à l'aide de différents protocoles de niveau IP, puisque les TTL sont fonctions de ce facteur. On peut encore pousser ce test, puisque le TTL ne sera pas le même en TCP, si l'on met comme port cible un port ouvert, ou fermé (qui génère un RST). Dans les tests TCP suivant, nous envoyons un datagramme à un port ouvert.

Test avec UDP :

```
# hping -2 -c 1 -p 1234 192.168.0.51
HPING 192.168.0.51 (vr0 192.168.0.51): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from ip=192.168.0.51 name=win2k.enslaved.lan
```

Ici, le résultat est lu avec tcpdump, et on trouve un TTL de 128.

Test avec ICMP :

```
$ ping -c 1 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: icmp_seq=0 ttl=128 time=3.881 ms
```

Test avec TCP :

```
# hping -S -p 139 -c 1 192.168.0.51
HPING 192.168.0.51 (vr0 192.168.0.51): S set, 40 headers + 0 data bytes
len=46 ip=192.168.0.51 flags=SA DF seq=0 ttl=128 id=4020 win=16616 rtt=40.8 ms
```

	Windows 2000	**Solaris 2.5**	**OpenBSD 3.0**
UDP	128	255	255
TCP	128	64	128
ICMP	128	255	255

Analyse des protocoles IP supportés

Pour savoir quels protocoles un système implémente, il suffit d'envoyer un datagramme IP avec le numéro de protocole à tester. Si la cible retourne en réponse un message ICMP de type *destination unreachable* et de code *protocol unreachable*, c'est bien sûr que ce protocole n'est pas supporté. Si aucune réponse n'est donnée, le protocole est considéré comme supporté.

```
# sendip -p ipv4 -is 192.168.0.55 -ip 60 192.168.0.1
```

Le résultat avec tcpdump donne :

```
22:31:51.680033 192.168.0.1 > 192.168.0.55: icmp: 192.168.0.1 protocol 60 \
  unreachable (ttl 255, id 41033, len 48)
```

Les résultats suivant ont été obtenus avec l'option -sO de nmap.

	NetBSD 1.6	**OpenBSD 3.0**	**Linux 2.2**
ICMP	Oui	Oui	Oui
IGMP	Non	Oui	Oui
IP	Non	Oui	Non
TCP	Oui	Oui	Oui
UDP	Oui	Oui	Oui
IPv6	Non	Oui	Non
GRE	Non	Oui	Non
ESP	Non	Oui	Non
AH	Non	Oui	Non
Mobile	Non	Oui	Non
EtherIP	Non	Oui	Non
IPComp	Non	Oui	Non

Ces résultats sont à prendre avec un certain recule, puisque les deux systèmes BSD ont été bien modifiés (noyaux customisés).

Puisque le champ protocole de l'en-tête IP est codé sur 8 bits (voir Figure 1, il n'y a qu'un maximum de 255 protocoles supportés possible.

Analyse d'autres protocoles

Il est maintenant légitime d'imaginer l'identification des systèmes en analysant comment sont construits les paquets d'autres protocoles IP, tels que IGMP. Nous ne sommes pas au courant de recherches dans cette direction, c'est une idée qui pourrait être intéressante à explorer.

Quelques outils

Les outils suivants implémentent les méthodes décrites plus haut.

QueSO? , par Savage[2]

Le premier outil à faire de l'OSFP actif. Seulement 100 signatures. Il n'est pas très précis, et n'est plus maintenu depuis 1998.

Nmap, par Fyodor[3]

Le meilleur outil d'OSFP actuel. Il lance de très nombreux tests, a une base de plus de 400 signatures différentes. Régulièrement mis à jour, c'est l'outil incontournable.

Xprobe, par Fyodor Yarochkin et Ofir Arkin[4]

Xprobe est un *proof-of-concept* sur l'usage d'ICMP pour identifier les systèmes d'exploitation. Un reproche possible est qu'il n'a pas de fichier de signature, mais c'est normal, étant donné son mode de fonctionnement, qui consiste à suivre un arbre de tests, en n'en jouant que certains en fonction des résultats des précédents. Il est capable d'identifier certains OS avec uniquement un message ICMP. Il est meilleur que d'autres outils pour ce qui est de la détection des systèmes Microsoft.

RING, par Franck Veysset, Olivier Courtay, Olivier Heen[5]

Une nouvelle approche à l'OSFP actif. Nous n'avons pas parlé de cette méthode dans ce texte pour la simple raison qu'il sera le sujet d'un prochain article également sur l'OSFP actif.

Quelques mots pour finir

Le recoupement de tous ces tests, même ceux qui semblent ne pas apporter grand chose, permet de créer une signature pour un système.

Cet article a essayé d'expliquer comment fonctionne l'OSFP, et de donner des idées de recherches dans ce domaine qui, de l'avis de certains, n'a pas encore tout révélé. Toutes les méthodes n'ont donc pas été abordées ici, et celle abordées n'ont pas été approfondies complètement.

Des tests[9] qui apportent de l'information intéressante n'ont pas été abordés dans ce texte, puisque ces tests seraient bloqués par des dispositifs de normalisation de paquets (ou des dispositifs de filtrage bien configurés). Cela vient du fait que nous pensons que l'avenir de l'OSFP se trouve dans les tests utilisant uniquement des paquets standards.

Références

[1] ICMP Usage in Scanning - Ofir Arkin
    http://www.sys-security.com/html/projects/icmp.html
[2] QueSO - Savage
    http://www.apostols.org/projectz/queso/
[3] Nmap - Fyodor
    http://www.insecure.org/nmap/
[4] Xprobe - Ofir Arkin & Fyodor Yarochkin
    http://www.sys-security.com/html/projects/X.html
[5] RING/Cron-OS - Franck Veysset, Olivier Courtay, Olivier Heen
    http://www.gomor.org/bin/view/GomorOrg/RingCronOs
[6] SendIP - Mike Ricketts
    http://www.earth.li/projectpurple/progs/sendip.html
[7] SING - Alfredo Andres Omella (Slay)
    http://sourceforge.net/projects/sing/
[8] hping - Salvatore Sanfilippo (Antirez)
    http://www.hping.org/
[9] Remote OS detection via TCP/IP Stack FingerPrinting - Fyodor
    http://www.insecure.org/nmap/nmap-fingerprinting-article.html

---

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