L'art du Port Scanning

 ${\bf Nicolas~Bareil~(nbareil@madchat.org)}$

17 décembre 2002

Table des matières

Introduction

Le piratage d'un serveur n'est possible que dans le cas où l'on possède suffisament d'informations sur cette machine; en effet, nous devons être en mesure de connaître les services (daemon) qui tournent, afin de déterminer les failles potentielles.

Cette phase est d'importance capitale pour un pirate, nous tenterons donc dans ce papier d'expliquer les différentes possibilités offertes à notre jeune *cracker*.

Au fur et à mesure, les techniques se sont portées sur plusieurs plans : on commence simplement par se connecter bêtement, puis, on joue plus finement en ouvrant à moitié les connexions. Enfin, on manipule les paquets en leur appliquant des états incohérents. Ces différentes méthodes sont plus ou moins efficaces niveau furtivité.

Il n'existe pas énormément d'outil de scan, personnellement, je n'en connais que deux qui soient aboutis : nmap¹ et Nessus². nmap est constamment en developpement, son auteur principal, Fyodor, est toujours à l'écoute des propositions de fonctionnalités ou des rapports de bug. Si on est un administrateur parano, il est vivement conseillé de lire la liste de développement³, car par ce biais, on est au courant des dernières trouvailles en matière de scanning.

La question de la légalité des scans est un sujet brulant pour certains, avec des analogies plus ou moins justes. Si cet aspect du sujet vous intéresse, je vous invite à lire la discussion qui a eu lieu sur fr.comp.securite

¹nmap : http://www.insecure.org/nmap/

 $^{^2{}m Nessus}$: http://www.nessus.org/

Chapitre 1

Méthode directe

1.1 Appel système connect()

1.1.1 Description rapide

Cette technique est sûrement la plus ancienne, en effet, elle consiste simplement à utiliser l'appel système connect() sur chaque port de la machine distante.

1.1.2 Discussion technique

Ici, il n'y a aucune ruse de sioux : aucune dissimulation n'est effectuée. Toutes les étapes de l'établissement d'une connexion sont réalisées. C'est la méthode classique pour se connecter à un serveur.

1.1.3 Avantages et inconvénients

Son gros avantage réside dans sa simplicité : un *scanner* de ce type est très facilement réalisable par un programmeur débutant. Ce *scan* possède également l'avantage d'être employé par un utilisateur quelconque : les privilèges root sont inutiles.

Par contre, il y a beaucoup plus d'inconvénients : Étant donné que ce scan est connu de tout le monde, les IDS le reconnaissent sans grande difficulté. Autre inconvénient : sa lourdeur d'utilisation, effectuer un scan de ce type est très consommateur en bande passante car pour chaque port (il y en a quand même $2^{16}-1:65\;535$ ports), il y a un échange d'au moins 6 paquets (voir à ce sujet le 3-way-handshake).

nmap (ainsi que tout scanner un tant soit peu évolué) propose une option pour alléger votre besogne, c'est l'option -F (Fast Scan mode) qui, au lieu de tenter de se connecter aux 65 535 ports, va juste essayer les ports classiques listés dans le fichier /etc/services (326 services sous Debian GNU/Linux).

1.1.4 nmap

Utilisation

C'est le *scan* par défaut, Fyodor¹ l'appelle également *Vanilla Scan*. Mais pour les gens compliqués, c'est l'option -sT.

nmap -sT babasse.matoufou.org

May the source be with you...

Si vous voulez mettre vos mains dans le code de *nmap*, qui en passant, est trés bien programmé, cela se passe dans "src-nmap/scan_engine.c à partir de la ligne 836, c'est la fonction pos_scan().

1.2 Utilisons identd

1.2.1 Introduction

Ce protocole, normé par l'IETF² et basé sur TCP, ³, semble obsolète de nos jours. En effet, qui, à part les vieux baroudeurs, s'en rappelle? C'est pour cette raison qu'à l'installation, on ne se pose même plus la question : on supprime le service et on a bien raison de le faire. La fonction principale de ce serveur est d'authentifier l'identité de l'utilisateur qui a lancé le programme à l'écoute d'un port TCP spécifique.

L'utilisation d'*identd* ne nous permet pas de scanner un serveur, c'est pour cela qu'on est obligé de le coupler avec un *Vanilla Scan*.

1.2.2 Et en quoi cela nous intéresse t-il?

Et bien en substance, cette option de *scan* n'est pas vraiment vitale, en effet, la seule information que l'on peut en tirer est le propriétaire. Par exemple, on peut savoir à distance si le serveur HTTP tourne sous l'identité root ou non.

1.2.3 Comment ça marche?

Le serveur *identd* peut nous renvoyer 2 types de réponse globale : en cas de réussite, il renvoit une chaîne de caractères USERID, ou dans le cas contraire, il renvoit la chaîne de caractères ERROR suivi du type d'erreur.

 $^{^{1}}$ Fyodor - L'auteur de nmap

²IETF: Internet Engineering Task Force

 $^{^3}$ rfc 1413 (disponible sur ftp ://ftp.ietf.org/rfc/rfc1413.txt)

1.2.4 Avantages et inconvénients

Les avantages ne sont pas légions; il n'y en a qu'un seul en fait : c'est simplement la faculté de pouvoir connaître l'utilisateur qui fait tourner un service. On peut donc parler des inconvénients qui sont trés nombreux.

Indisponibilité : Je ne connais aucun administrateur qui ait laissé ce service disponible sur son serveur. Et pour cause, ce protocole est clairement défini comme *dangereux* dans la RFC vis-à-vis de la vie privée.

Fiabilité : Les nouveaux serveurs *identd* possèdent des limitations dans le nombre de questions/réponses. Il y a également deux approches envisagées pour lutter contre le *scan* :

- 1. Interdire les interrogations à propos des ports inférieurs à 1024,
- 2. Interdire les interrogations de ports possèdants des *sockets* en état TCP_LISTEN

Furtivité nulle: Avec cette option, vous êtes pratiquement sûr d'être repéré, en effet, pour l'utiliser, vous devez forçément employer le *Vanilla Scan*, et ce scan étant déjà facile à repérer, les différentes connexions au port 113 vous interdisent toute tentative de furtivité.

1.2.5 nmap

Utilisation

Comme tout le monde le sait, nmap sait presque tout faire (sauf le café malheureusement), donc pour effectuer un scan avec ident, c'est l'option -I. Mais n'oubliez pas que l'option -sT ($Vanilla\ Scan$) est implicite.

nmap -I babasse.matoufou.org

May the source be with you...

Cela se passe dans "src-nmap/nmap.c à la ligne 1729, c'est la fonction getidentinfoz().

Chapitre 2

Jouons avec l'établissement et la fermeture de la connexion...

2.1 Syn Scanning

2.1.1 Handshake, nouveau type de Milk-shake?

Avant propos

Lorsque vous contactez une machine, par exemple pour relever vos mails, vous établissez une connection TCP. Cet ensemble de protocoles, contrairement à ICMP ou UDP, permet de faire transporter vos données en mode "connecté". C'est à dire que pour initier la liaison, vous devez vous présenter, parler, puis dire au revoir. Autre caractéristique de TCP, c'est la gestion de ports et c'est cette dernière partie qui nous intéresse.

Handshake

Littéralement, handshake signifie "poignée de main", c'est donc ce que va faire un ordinateur lorsqu'il veut en contacter un autre. Cette étape est obligatoire et doit être réalisée avec succès, il n'y a pas d'autre possibilité pour initier une connection.

Pourquoi est-ce que ce processus est obligatoire? Car il garanti que les deux ordinateurs sont prêt à recevoir des données et il permet également de s'accorder sur les numéros de séquences.

Explication précise

Lorsque l'ordinateur A veut contacter l'ordinateur B, A doit tout d'abord envoyer un segment avec le bit de contrôle SYN activé en direction de B. Ce segment, encapsulé dans un paquet IP, contient alors toutes les informations nécessaires pour se présenter à B: l'adresse source et destination, le port source et le port visé. C'est à partir de ces données que B va alors choisir d'ouvrir ou non une connexion. Si B autorise la liaison, il va renvoyer un segment TCP avec le bit SYN activé et le champ ACK contenant le numéro de séquence du segment initial. L'ordinateur A, à la réception de ce segment, n'aura plus qu'à confirmer l'arrivée du segment par l'émission d'un segment avec le bit ACK rempli.

A partir de ce moment, les deux ordinateurs sont synchronisés au niveau de leur numéro de séquence ¹ et sont prêt à recevoir des données. Le premier ayant envie de parler pouvant commencer.

Dans le cas où B refuserait d'établir la connexion, celui-ci émettrait un segment RST en direction de A. La poignée de main ayant échoué, la connexion ne peut avoir lieu. On peut alors en déduire l'état de B : port ouvert, ou fermé. Bien sûr, le refus de B peut être dû à un autre type de problème, mais ce n'est pas notre soucis pour un scan.

2.1.2 Description

Cette technique va ouvrir une connexion qu'à moitié (On parle alors de Half Open Scan). Comme nous l'avons vu précédement, l'initialisation d'une connection n'est possible que si le handshake est réalisé avec succès, c'est-à-dire si les cinq segments TCP ont été échangés. Notre but est de savoir si le port distant est ouvert ou non, or nous savons que pour connaître son état, il suffit d'envoyer un seul segment SYN (le premier du handshake). A la réception de ce paquet, le serveur va alors renvoyer soit un segment RST nous laissant penser que le port est fermé, soit un segment SYN + ACK autorisant la connexion (le port est donc ouvert). Dans ce dernier cas, l'ordinateur attaquant n'émettra qu'un segment RST pour couper court à la liaison (ce n'est pas le scanner qui va se charger de cette tâche en fait, mais le noyau ne reconnaissant pas cette connexion).

2.1.3 Un peu de technique :

En pratique

L'utilitaire de Salvatore Sanfilippo, hping², permet de créer des segments/paquets suivant notre envie. C'est grâce à lui que nous allons mettre en évidence les techniques décrites.

On essaie tout d'abord sur un port ouvert (sshd):

nbareil@mouarf % sudo hping2 --syn -p 22 -c 1 babasse.matoufou.org
HPING babasse.matoufou.org (eth0 192.168.0.42): S set, 40 headers + 0 data bytes
len=46 ip=192.168.0.42 flags=SA DF seq=0 ttl=64 id=57085 win=5840 rtt=2.0 ms

 $^{^{1}\}mathrm{SYN}: \mathrm{Synchronisation}$

²hping: http://www.hping.org/

Puis sur un port fermé:

nbareil@mouarf % sudo hping2 -p 42 -c 1 --syn babasse.matoufou.org HPING babasse.matoufou.org (eth0 192.168.0.42): S set, 40 headers + 0 data bytes len=46 ip=192.168.0.42 flags=RA DF seq=0 ttl=255 id=64331 win=0 rtt=0.3 ms

Nous voyons alors que pour le port fermé, la réponse a été un RST + ACK (RA), alors que pour le port ouvert, nous avons eu le droit à SYN + ACK (SA).

2.1.4 Programmation

Les systèmes d'exploitation permettent aux programmeurs d'accéder aux sockets de bas-niveau : On peut pratiquement manipuler directement la carte réseaux ; en effet, on peut construire nos propres paquets IP donc créér nos segments TCP... On appelle cette fonctionnalité les "Raw sockets", elle est disponible sous tous les Unix, mais également sous GNU/Linux et les nouveaux systèmes Microsoft (Windows XP et 2000).

Sous Unix, il suffit de créér une socket avec le mode SOCK_RAW pour la positionner en mode raw. L'ennui de cette méthode est sa faible capacité à être porté sous d'autres plate-formes. Chaque système à sa facon de voir les choses, il est alors nécessaire d'employer une couche de haut-niveau qui se chargera de ces problèmes. En C, il existe la très populaire bibliothèque Pcap qui tourne sur pratiquement tous les ordinateurs. Si le C vous effraie, une implémentation Perl existe, c'est le module Net: RawIP disponible sur le CPAN³.

Avantages et inconvénients

Le gros avantage de ce scan est sa furtivité relative : peu d'administrateurs enregistrent ce type de log. C'est sûrement le scan le plus discret possible : seulement deux paquets sont émis au maximum pour chaque port, alors que pour le Vanilla Scan, il en faut le double. Au niveau de la programmation, que ce soit en C ou en Perl, la technique n'est pas très compliquée (triviale en Perl comme d'habitude).

Néanmoins, une contrainte existe, c'est que l'utilisation des "raw sockets" est limitée à l'administrateur (le méchant root). C'est le seul ennui de ce scan, ce qui n'est pas grand chose. Malheureusement, avec l'arrivée des IDS⁴ modernes, cette technique n'est plus aussi furtive qu'auparavant, en particulier à cause des nombreux DoS⁵ par "SYN flooding" qui ont poussé les IDS à intégrer la détection de SYN suspect.

³CPAN : http ://www.cpan.org ⁴IDS : Intrusion Detection System

⁵DoS : Denial of Service

2.1.5 nmap

Utilisation

nmap permet ce type de scan avec l'emploi de l'option -sS:

nbareil@mouarf % sudo nmap -sS babasse.matoufou.org

2.2 FIN Scanning

2.2.1 Ça y est, j'ai fini

Contrairement à ce que pense beaucoup de programmeurs, TCP/IP ne définit pas une méthode pour dire : "Je coupe la communication maintenant", par contre, il faut dire : "Je n'ai plus rien à émettre". Ici encore, il y a un échange (de la même facon que le 3 handshake) entre les deux parties pour clore une connexion existante.

Lorsq'un programme (A) utilise l'appel système close(), un segment TCP avec le champ de contrôle FIN est émit vers l'ordinateur (B). Dans le cas normal (si la connexion existe réellement), le correspondant répond alors par un segment FIN + ACK.

Mais si l'ordinateur B recoit un FIN alors qu'aucune connexion avec A n'est établie, que se passe t'il? D'après la RFC 793, un port fermé est obligé de retourner un segment RST à la réception d'un segment n'ayant pas le bit RST d'activé. Dans notre cas, et pour passer inaperçu, nous allons profiter de cette obligation pour forger un segment FIN (sans le bit RST) en destination de notre port cible. Si nous recevons un segment RST, c'est que le port est fermé, autrement, c'est qu'il est ouvert.

2.2.2 Démonstration

nbareil@mouarf % sudo hping2 -p 42 -c 1 --fin babasse.matoufou.org HPING babasse.matoufou.org (eth0 192.168.0.42): F set, 40 headers + 0 data bytes len=46 ip=192.168.0.42 flags=RA DF seq=0 ttl=255 id=59792 win=0 rtt=0.4 ms

```
--- babasse.matoufou.org hping statistic ---
1 packets tramitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.4/0.4 ms
```

Ici, un segment RST + ACK a été renvoyé (RA), ce qui confirme l'état du port 42: il est fermé.

Sur le serveur sshd:

nbareil@mouarf % sudo hping2-p 22 -c 1 --fin babasse.matoufou.org
HPING babasse.matoufou.org (eth0 192.168.0.42): F set, 40 headers + 0 data bytes

```
--- babasse.matoufou.org hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Rien de plus décevant que de ne rien recevoir : Malgré l'emission d'un segment FIN, nous n'avons reçu aucune réponse. Pas de nouvelle, bonne nouvelle : le port 22 est ouvert, sous réserve du bon état du réseau , mais nous le verrons plus tard.

2.2.3 Programmation

La technique utilisée est très proche de celle du SYN-scan, à la différence que nous devons gérer un timeout (dépassement de temps). Si nous ne recevons aucun paquet après l'émission de notre segment FIN (en utilisant les raw-sockets) dans les trente millisecondes qui suivent, c'est qu'il y a deux possibilitées :

- Soit le réseau a perdu la paquet RST + ACK (temps de routage, perte...),
- Soit le port est bel et bien ouvert, et dans ce cas là, c'est tout à fait normal qu'on ne recoive aucune nouvelle.

Il faut donc gérer ces paramètres. A vos clavier pour imaginer un algorithme simple qui s'adapterait à l'état du réseau : sachant qu'un réseau en mauvais état demandera un gros *timeout*.

2.2.4 Avantages et inconvénients

Bien que très discret, ce scan n'est pas la solution miracle. En effet, tout *IDS* moderne conservant en mémoire les différents états de connexion pourra facilement voir qu'un segment FIN anormal a été envoyé. Et là, remontée d'alarme, police, prison, DST, balance.

L'algorithme utilisé doit être irréprochable : le *timeout* doit être ajusté en fonction de plusieurs paramètres : capacité, débit, état, et stabilité du réseau. Si 80% des paquets IP sont égarés, le *scan* sera soit faux, soit très lent.

2.3 ACK Scanning et Window Scanning

2.3.1 Description

L'ACK scanning ne permet pas de connaître l'état réel des ports sur une machine, mais de savoir si ce port est filtré par un firewall.

Sur les firewalls non *stateful*, la seule façon d'autoriser l'établissement d'une connexion est de consulter le drapeau SYN d'un segment : si celuici est activé, c'est que c'est le premier paquet (peut-être même le dernier) d'une connexion. L'administrateur doit donc filtrer ce type de paquet : si aucun paquet SYN ne peut être envoyé, aucune connexion n'aura lieu, il n'y

a donc pas de raison d'interdire les segments ACK, ou FIN, puisqu'ils n'ont pas de sens sans contexte de connexion.

Comme nous avons vu précédemment, un ordinateur qui reçoit un segment "orphelin" doit renvoyer un paquet RST. Mais cela n'est possible que si cette machine l'a recu, c'est-à-dire si aucun firewall n'a bloqué le paquet. C'est grâce à ce raisonnement que nous pouvons déterminer le jeu de règle de l'administrateur.

2.3.2 Démonstration

Dans un premier temps, nous allons supprimer les règles de firewall sur la machine cible :

```
root@babasse # iptables -F
```

Et maintenant, envoyons un segment avec le bit ACK activé sur un port ouvert (sshd):

nbareil@mouarf % sudo hping --ack -c 1 -p 22 babasse.matoufou.org HPING babasse.matoufou.org (eth0 192.168.0.42): A set, 40 headers + 0 data bytes len=46 ip=192.168.0.42 flags=R DF seq=0 ttl=255 id=52398 win=0 rtt=0.4 ms

```
--- babasse.matoufou.org hping statistic ---
1 packets tramitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.4/0.4 ms
```

Puis sur un port fermé:

nbareil@mouarf % sudo hping --ack -c 1 -p 42 babasse.matoufou.org HPING babasse.matoufou.org (eth0 192.168.0.42): A set, 40 headers + 0 data bytes len=46 ip=192.168.0.42 flags=R DF seq=0 ttl=255 id=60265 win=0 rtt=0.5 ms

```
--- babasse.matoufou.org hping statistic ---
1 packets tramitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.5/0.5/0.5 ms
```

Comme vous pouvez le constater, il n'y a aucune différence. En effet, comme le veut la RFC, un segment RST a été envoyé puisque ce paquet venait de nulle part.

Mais maintenant, réactivons le firewall (Netfilter sous GNU/Linux 2.4.18), en ne laissant passer que le flux ssh :

```
root@babasse # iptables -A INPUT -p tcp --destination-port ssh \
    -m state --state NEW -j ACCEPT

root@babasse # iptables -A INPUT -m state --state ESTABLISHED,RELATED \
    -j ACCEPT

Et maintenant, recommençons nos tests.
Sur un port que l'on sait ouvert :

nbareil@mouarf % sudo hping --ack -c 1 -p 22 babasse.matoufou.org

HPING babasse.matoufou.org (eth0 192.168.0.42): A set, 40 headers + 0 data bytes len=46 ip=192.168.0.42 flags=R DF seq=0 ttl=255 id=2485 win=0 rtt=0.7 ms

--- babasse.matoufou.org hping statistic ---
1 packets tramitted, 1 packets received, 0% packet loss round-trip min/avg/max = 0.7/0.7/0.7 ms

Ici, rien de nouveau. Mais qu'en est il sur le port fermé (42)?
```

nbareil@mouarf % sudo hping --ack -c 1 -p 42 babasse.matoufou.org
HPING babasse.matoufou.org (eth0 192.168.0.42): A set, 40 headers + 0 data bytes

```
--- babasse.matoufou.org hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

root@babasse # iptables -P INPUT DROP

Et voila : le firewall a $drop-\acute{e}$ notre segment RST silencieusement. Nous ne recevrons aucune réponse.

Grâce à cette méthode, nous pouvons connaître les lignes générales des règles de l'administrateur.

Auparavant, nous pouvions connaître l'état d'un port en étudiant les segments RST, de fait, un "bug" était présent dans la pile TCP/IP des vieux BSD, or, c'est de ce code que la majorité des systèmes se sont inspirés. Le scanner analysait le champ TTL ⁶ des paquets RST : si celui-ci était supérieur à 64, cela signifiait que le port était ouvert, autrement, il était fermé. Vu comme cela, ça parait magique n'est-ce pas? Et bien désolé, mais cette époque est terminée : la grand majorité des systèmes ne sont plus vulnérables.

Par contre, on peut toujours utiliser un "bug" présent sur un grand nombre de système. Ce problème est encore exploitable en fonction de la réponse du serveur (notre fameux segment RST). Si l'on analyse ce paquet,

⁶TTL : Time To Live

on se rend compte que dans le cadre d'un port ouvert, le champ Window est non nul, alors qu'il est égal à zéro pour un port fermé. Magique non?

Comme nous pouvons le voir, les principes utilisés sont très simples et résultent souvent d'un *bug*. C'est pour cette raison qu'il ne faut pas avoir une confiance absolue dans les résultats de scan : on ne sait jamais si la machine victime est "vulnérable" à ce problème ou non.

2.3.3 Utilisation

ACK scanning

nbareil@mouarf % sudo nmap -sA babasse.matoufou.org

Window Scanning

nbareil@mouarf % sudo nmap -sW babasse.matoufou.org

2.4 Idle Host Scan

2.4.1 Description

Cette méthode, également appelée Dumb Host Scanning permet à l'attaquant une certaine forme d'anonymat. Il est possible de remonter vers l'attaquant à condition que l'on ait 42 ans à gaspiller. Comment cette discrétion est-elle possible? Tout simplement parce que la machine cible n'aura jamais affaire à nous : lors de notre envoi de segments SYN,ACK,FIN, nous n'allons pas mettre notre adresse IP dans les paquets, mais celle d'une troisième machine respectant une condition unique, et impérative : elle doit être inactive, on appelle ce type d'ordinateur un zombie. C'est-à-dire qu'il ne recevra ou n'émettra aucun paquet autre que les nôtres. Grâce à la popularisation des connexions haut débits, on est capable de trouver très facilement une machine sous Microsoft Windows qui corresponde à nos besoins.

2.4.2 Mais encore...

L'attaque se décompose en 3 étapes :

- 1. Prise de connaissance de l'état initial de la pile TCP/IP de la machine Zombie.
- 2. Envoi des paquets spoofés vers la machine cible.
- 3. Étude de l'état final de la pile TCP/IP de notre Zombie

Un paquet IP contient un identifiant appelé *id* qui est unique pour chaque connexion. Ce champ doit être incrémenté d'une façon particulière pour que le système puisse servir de Zombie. En effet, à chaque envoi de paquet, la pile TCP/IP doit ajouter un nombre quelconque qui reste fixe. Par exemple, sous

Microsoft Windows, il y a une différence de 256 entre 2 paquets envoyés. Mais n'essayez pas sous GNU/Linux car cela ne marche pas de la même façon.

C'est en regardant ce champ que nous pouvons savoir si entre deux paquets que nous avons envoyés, la machine *Zombie* a reçu des connexions.

ID initial

La première phase va alors consister à connaître l'id courant de la pile de la machine. Il suffit de recevoir un simple paquet, on se contentera souvent d'un segment RST après l'envoi d'un segment ACK (voir la partie précédente à ce sujet). On peut donc prévoir la valeur de l'ID prochain qui sera égale à :

next_id = current_id + decallage

Test du port

- Dans le cas d'un port ouvert
 - À présent, nous allons essayé de déterminer si le port 80 est ouvert sur la machine victime (babasse.matoufou.org), ce qui est le cas. Nous envoyons un segment SYN de la part de notre Zombie (c'est l'IP spoofing) en direction du serveur cible. Le port 80 est ouvert, la victime va répondre au Zombie par un segment où les champs SYN et ACK seront activés. Cependant, l'ordinateur Zombie n'ayant jamais rien envoyé, il va émettre un segment RST pour indiquer qu'il n'a rien à voir là dedans. C'est cette réponse qui va nous être cruciale car ainsi, le champ ID a été incrémenté!
- Et si le port est fermé?
 Envoyons maintenant une demande de connexion (SYN) vers le port 42 (qui est fermé), toujours en spoofant l'adresse source. La machine cible va alors refuser la connexion en répondant par un segment RST (voir les détails dans la partie sur le syn scanning). D'après la RFC 793, notre Zombie ne devra rien répondre à ce segment inattendu. Cette inactivité nous sera très intéressante.

ID final

- Si le port est ouvert
 - De la même façon que pendant l'étape initiale, nous allons prendre connaissance de l'*ID* courant et nous apercevoir que celui-ci est supérieur à l'*ID* qu'on avait prédit au départ.
 - On peut supposer que le *Zombie* a envoyé un segment RST vers la victime. Si c'est le cas, c'est que le port est ouvert.
- Si le port est fermé

De même, après tests, nous nous apercevons que l'*ID* courant est égal à celui qu'on avait prédit au départ. Aucun paquet n'a été envoyé. En conséquence, on peut considérer que le port est fermé.

2.4.3 Discussion

Comme vous venez de le voir, cette technique est sans aucun doute la plus anonyme, mais malheureusement pas la plus fiable : L'état d'un port ne peut être déterminer qu'après de nombreuses suppositions. C'est pour cette raison qu'il est nécessaire de faire plusieurs essais afin de pouvoir affirmer qu'un port est ouvert ou fermé.

La recherche de *Zombie* ne devrait pas être très difficile étant donné que les systèmes *Microsoft* sont vulnérables, ainsi que la plupart des ordinateurs embarqués : Imprimantes réseaux, routeurs...

2.4.4 nmap

nmap permet ce genre de scan depuis la version 2.54 Beta 56. La syntaxe d'utilisation est celle-ci :

nbareil@mouarf % sudo nmap babasse.matoufou.org -sI zombie

2.5 Conclusion

Toutes les techniques de scan que nous avons vu précédemment visaient à contourner les mécanismes organisés par les RFC. Ces techniques étaient plus ou moins "portables" en fonction des plates-formes puisque tout est normalisé.

Les segments que nous avons envoyés étaient tous normaux, mais émits à des moments invalides. On peut donc considérer ces paquets, si on les prend un par un, comme parfaitement légitime. Ce qui avait pour incidence de nous procurer une certaine forme de discrétion : Seul un firewall stateful, ou la présence d'IDS pouvait remarquer ces paquets étranges.

Nous allons voir que nous pouvons faire beaucoup plus sale, et beaucoup plus bruyant.

Chapitre 3

Au tour des segments maintenant...

3.1 Méthode de barbares

À mon avis, ces techniques de scan sont à éviter car elles reposent sur trop d'hypothèses. Les deux méthodes qui vont suivre exploitent des bugs dans les piles TCP/IP.

Les scans précédents pouvaient passer inaperçus s'ils étaient noyés dans beaucoup de trafic. Mais ce n'est plus le cas avec les techniques suivantes. En effet, les paquets que nous enverrons seront totalement "illégaux", et donc très rapidement repérés, ce qui ne va pas arranger nos affaires.

3.2 Xmass Scan

Littéralement, on peut traduire l'expression Anglaise par " $paquet\ de\ No\"el$ " car tous les champs de contrôle vont être activés dans le segment, c'est-à-dire les bits: RST, SYN, FIN, ACK, URG, PUSH...

D'après la *RFC 793*, à la réception d'un segment avec le bit RST activé, l'ordinateur doit ignorer le segment, que le port soit ouvert ou non. Le *bug* oublie ces commodités et agit comme il l'entend : si le port est ferm, il enverra un segment RST, et pour un port ouvert, il ne fera rien.

3.3 Null Scan

Cette *méthode* est complètement l'opposé de l'*Xmass Scan*, car au lieu d'activer tous les *bits*, nous allons laisser vide tous les champs TCP du segment. De la même manière, si le port est ouvert, la victime ne répond rien, autrement, un segment RST est émit.

3.4 À jeter à la poubelle?

Les problèmes sont les mêmes que ceux du FIN scanning : il faut implémenter un système de timeout, et le gérer efficacement, il en ressort donc des false-positives. De plus, ce type de scan n'est possible que contre les UNIX.

Cette spécificité est un bonus supplémentaire pour déterminer si le serveur distant est sous UNIX ou non.

À mon avis, il vaut mieux oublier la discrétion avec cette technique car un paquet aussi original que celui-ci ne peut pas passer inaperçu vis-à-vis des *firewalls* et autres sondes *IDS*.

Le seul avantage a utilisé cette technique est son faible "coût" en ressources : seulement un segment suffit. Mais cela reste une bien maigre consolation...

Chapitre 4

Il n'y a pas que TCP dans la vie...

4.1 User Datagram Protocol: UDP

4.1.1 Rappel sur ce protocole

Contrairement aux protocoles TCP, UDP ne permet pas le transport en mode connecté : il n'y a pas de notion de connexion. De ce fait, une connexion peut se résumer à un seul paquet. Aucune réponse n'est obligatoire (car UDP a été créé pour être rapide, et léger : il n'y a aucune garantie d'intégrité et de livraison). Il n'y a aucune sorte de handshake au début.

4.1.2 Intérêt limité?

Le scan de port non TCP peut paraître un peu inutile. En effet, peu de services majeurs existent en UDP, les plus connus sont : DNS et NFS suivant la configuration choisie par l'administrateur, mais aussi SNMP, TFTP, logiciels de $p2p^1...$

Chose amusante, le *scan* de port montrera très souvent la présence de *backdoors*, en effet, il semblerait que beaucoup de personnes aient du mal avec l'UDP : il est fréquent que tous les flux UDP soient autorisés. Par souci de simplicité sûrement? Cette faille béante est donc utilisée à mauvais escient : les pirates le savent et programment leurs *robots* pour écouter sur un port quelconque en UDP.

Les failles de l'année dernière sur les services \mathtt{rpc} touchaient aussi bien les serveurs GNU/Linux que Solaris, ou n'importe quel BSD. On ne devrait donc pas négliger l'UDP scan.

¹p2p : Peer To Peer

4.1.3 Description du scan

Le scan consiste simplement à envoyer un paquet UDP vide (aucune donnée) vers un port donné. Bien entendu, nous ne devons pas nous attendre à recevoir un quelconque segment RST qui est spécifique au protocole TCP.

Si le port est fermé, la RFC² indique qu'un paquet ICMP de type 3 doit être envoyé : "Destination unreachable" ³. Le code ICMP utilisé étant le numéro 3 : "Port inaccessible".

Si le port est ouvert, la victime ne répond alors rien du tout.

4.1.4 Mais il y a de nombreux problèmes

Le scan pourrait être efficace si les piles TCP/IP n'effectuaient pas une limitation des messages ICMP comme le préconise la RFC 1812 : le noyau Linux limite l'émission de ces paquets à 80 toutes les 4 secondes, et en cas de dépassement de cette limite, le noyau impose une pénalité d'un 1/4 de seconde. Pour plus de détails, cette limitation est faite par la fonction xrlim_allow() dans ~src-linux/net/ipv4/icmp.c.

Sachant qu'il y a 65 535 $(2^{16}-1)$ ports disponibles sur une machine. À raison de 40 essais par seconde, le temps nécessaire à scanner l'intégralité de l'ordinateur sera de 65535/(40*60) = 27 minutes. Un peu moins d'une demie heure alors que votre but est d'être le plus discret possible : pendant 30 minutes la console de l'administrateur va clignoter avec votre adresse IP en gros.

De plus, ICMP ne garantit pas l'arrivée des paquets, l'un d'entre eux peut donc s'égarer nous donnant alors une fausse joie. Le flux ICMP est également de moins en moins bien filtré par les administrateurs : soit tout est autorisé, soit tout est ouvert. Ou au mieux, il y aura un *firewall* devant la cible qui droppera certains paquets pour limiter le trafic ICMP.

Vous pouvez donc vous attendre à pas mal de faux positifs.

4.1.5 Deux techniques existent

En 1997, nmap avait deux méthodes pour scanner en UDP. La plus propre a été conservée, alors que l'autre a été jetée. Il faut dire qu'elle n'était pas très académique.

Pour les deux algorithmes, le concept est le même : recevoir un message ICMP, mais les moyens pour capter ce message sont différents.

La propre

Si nous sommes root, nous pouvons utiliser les Raw Sockets, celles ci nous permettent alors d'envoyer nos paquets, puis d'écouter tout le trafic

²rfc768 : User Datagram Protocol

³En français : Destination inaccessible.

entrant/sortant à la recherche du bon message ICMP.

Et l'autre...

Mais si nous ne sommes pas le grand chef de l'ordinateur⁴, nous pouvons dire adieu au mode *promiscuous* de la carte réseau. Mais *Linux* nous apporte parfois quelques cadeaux : après l'envoi d'un paquet UDP, le noyau renvoie une erreur si nous effectuons un recvfrom() sur une *socket* non bloquante. Cette erreur correspond à EAGAIN si le message ICMP n'a pas été reçu, autrement, l'appel système retournera ECONNREFUSED pour indique le refus de connexion.

Vous pouvez constatez que l'algorithme repose sur un bug (ou une fonctionnalitée, qui sait ?) du noyau. C'est peut-être pour cette raison que Fyodor a préféré retirer cette option.

4.1.6 Utilisation

nbareil@mouarf % sudo nmap -sU babasse.matoufou.org

⁴C'est root bien sûr!

Conclusion

Les principales techniques de scan n'ont, j'espère, plus de secrets pour vous. Comme vous avez pu le voir, il n'y pas beaucoup de techniques réellement furtives, il y a juste l'*Idle Host Scan* qui est garant de votre anonymat. Autrement, toutes les autres techniques sont facilement détectables par un IDS ou un logiciel fait pour (scandetd, scanlogd...).

N'oubliez pas que tant que vos daemons sont à jour, vous n'avez pas grand chose à craindre. Il n'y a pas lieu de porter plainte à chaque scan de vos serveurs :-)

Informations complémentaires

Cet article a été écrit comme support pour la conférence que j'animerai à la *Zelig Conf* le 15 décembre 2002 à Paris.

Remerciements

En vrac, et sans ordre particulier :

- $-\ \mathrm{mAdchAt}$ http://madchat.org/
- Léonard en particuler
- Lansciac, pour ses nombreuses corrections orthographiques,
- Christophe Casalegno http://www.digital-network.org/
- Amal B.
- Emacs, Gnus, LATEX, hping, Debian, RMS

Ce document

Cet article est disponible sur http://matoufou.org/linux/. Il a été édité sous Emacs (latex-mode). Quelques moulinettes *Perl* ont été utilisées avant que je me rende compte des options avancées du mode LATEX sous Emacs.

Auteur

Nicolas Bareil (nbareil@madchat.org).