



Théorie

Application mapping

Marc Ruef 

Degré de difficulté



En scannant les ports, il est possible d'observer les services proposés par un système. Mais, il reste à déterminer, en fait, les fonctionnalités cachées derrière tel ou tel port. C'est ici qu'entre en jeu le application mapping, processus permettant d'identifier le type et le protocole des services ciblés. Vous allez apprendre, dans le présent article, à utiliser une telle méthode de rapport et à vous défendre contre ce genre d'approches.

L'analyse de l'état des ports d'un hôte est une étape importante en termes de sécurité des réseaux, dans la mesure où le processus permettant de déterminer les services proposés ainsi que le protocole de l'application utilisé doit être lancé assez souvent. C'est dans un tel contexte que le application mapping constitue une solide méthode permettant d'obtenir ce genre d'informations avec rapidité et un accès plus efficace.

Le scan de ports représente une discipline parmi les plus classiques de la sécurité informatique. Avec les tentatives de connexion automatisées et l'analyse du comportement de ces approches, il est possible d'obtenir l'état des ports sur le système ciblé. Il existe un grand nombre d'outils faciles à utiliser pour ce genre d'analyse : Nmap, par exemple, ou SuperScan, connu pour son efficacité et sa convivialité [Ruef 1999, Ruef et al. 2002, McClure et al. 2005]. Il est essentiel de connaître cette information notamment en cas d'attaques basées sur le réseau ou sur les applications en réseau, puisque les ports ouverts et les services proposés servent de point d'entrée à ce type d'attaques, et permettent donc de définir la plus grande partie de la surface d'une attaque dirigée contre l'hôte.

Toutefois, déterminer l'état des ports par un scan de ports dans le but de lancer une attaque sur une application particulière n'est pas toujours suffisant. Après le scan, il faut parvenir à déterminer le protocole de l'application utilisé sur tel ou tel port. Bien que de nombreux administrateurs utilisent les ports standard, conformément aux recommandations de l'IANA (Internet Assigned Numbers Authority), il est impossible

Cet article explique...

- Le application mapping et quand l'utiliser.
- Les techniques sur lesquelles nous reviendrons.
- Comment automatiser le processus grâce à amap.
- Comment installer votre propre outil.
- Comment compliquer une telle fonction de rapport.

Ce qu'il faut savoir...

- Définition d'un port.
- Fonctionnalité du protocole TCP/IP et du principe client/serveur.

de leur faire confiance entièrement. Dans de nombreux cas, en effet, il est possible de trouver un serveur Web (tel qu'on peut trouver sur le port TCP/IP 80) sur d'autres ports, comme les ports 81 ou 8000. Il est même possible de trouver des variantes plus originales encore, comme les ports 12345 ou 55555 [Ruef 2004, 2006].

Afin d'identifier le protocole d'application utilisé, il est possible de faire appel à une technique appelée *application mapping* (ou *application mapping* en anglais) [Ruef 2003]. Cette technique devrait toutefois s'appeler mappage de protocole d'applications ou mappage de protocoles, puisque son objectif ne consiste pas à trouver l'implémentation d'une application réseau, mais le protocole de l'application en question [Packetwatch 2004]. L'expression application mapping provient initialement de l'outil dénommé Amap (Application Mapper), permettant d'automatiser certaines des techniques mentionnées plus haut. Nous y reviendrons un peu plus tard.

Messages d'accueil automatisés

La technique de application mapping repose sur le principe de base client/serveur. Un serveur fournit une application en réseau qu'il relie à un port de type TCP ou UDP. Ce port se met ensuite en mode écoute. Un client, désireux d'utiliser ce service en réseau, se connecte alors sur le port en question du serveur [Stevens 1994]. Le serveur Web constitue l'un des exemples les plus fréquents de ce principe. Ce serveur est lié au port TCP 80 (HTTP), conformément aux recommandations formulées par l'IANA. Un navigateur Web, dans ce cas, le client, appelle alors la cible et son port afin d'obtenir des documents Web. Le procédé serait le même avec un serveur de messagerie électronique et son client, puisque l'ensemble des services réseau repose sur l'architecture client/serveur.

Certaines applications textuelles ont été développées dans un souci d'interactivité. Une fois la connexion établie, ce genre d'applications pourrait, par exemple, accueillir le client au

moyen d'un message de bienvenue, contenant des informations sur l'implémentation ainsi que sur d'autres possibilités d'ordre technique. Ces applications sont alors connues comme SMTP (tcp/25) et FTP (tcp/21). Il est possible d'initier de telles connexions très facilement grâce à un client Telnet, afin de lire les messages d'accueil. Cette technique est connue sous le nom d'interception de messages d'accueil [Ruef 1999, Ruef et al. 2002, McClure et al. 2005].

La même technique peut s'appliquer aux implémentations en ligne, que vous pouvez trouver sur l'ensemble des systèmes d'exploitation les plus répandus, en tapant *telnet <target host> <target port>* [Ruef 2004, 2006]. Par exemple, si vous souhaitez établir une connexion avec le serveur mail.computec.ch sur le port TCP-port 25 (smtp), il suffit de taper *telnet mail.computec.ch 25*. Après avoir établi une connexion avec la couche TCP, vous pourrez consulter dans la sortie suivante le message d'accueil de ce serveur de messagerie électronique. Vous y trouverez notamment le nom *sendmail* et le numéro de version 8.12.6 de ce serveur SMTP.

```
C:\Documents and Settings\mruef>
telnet mail.computec.ch 25
220 mail.computec.ch ESMTP
Sendmail 8.12.6/8.12.6/taifun-1.0;
Wed, 30 Nov 2005 15
:54:54 +0100
```

De nombreuses applications réseau sont dotées de messages d'accueil automatisés. Il s'agit donc d'un signe d'implémentations connues, dirigées vers des services largement utilisés de type SMTP ou FTP. Toutefois, une réponse automatique va à l'encontre de l'usage d'un serveur Web HTTP. Ce petit signe ne suffit donc pas à reconnaître des applications réseau implémentées. Il faut donc avoir recours à des techniques complémentaires, présentées ci-après.

Simple filtrage de modèles

La technique la plus simple consiste à filtrer les modèles de ces réponses.

Le filtrage de modèle est une technique consistant à chercher une chaîne ou une correspondance de modèle. Il suffit, pour ce faire, de chercher des chaînes linéaires. Par souci de clarté, nous allons vous montrer un exemple complexe permettant d'illustrer le principe de application mapping. Prenons un fichier intitulé data.txt, contenant les lignes suivantes (le numéro précédant chaque ligne permet d'indiquer les exemples) :

```
01 This is the first line
02 This is the second line
03 This is the third line
04 This is the end of file
```

A l'aide de la commande `grep` sous Unix, il est possible de consulter le contenu d'un fichier. Pour ce faire, utilisez la syntaxe habituelle `grep <pattern> <file>`. Dans notre exemple, nous allons nous intéresser à la chaîne `This` dans le fichier data.txt. Comme ce modèle est présent dans les quatre lignes du fichier, nous obtenons toutes les lignes :

```
mruef@debian:~$ grep data.txt „This“
01 This is the first line
02 This is the second line
03 This is the third line
04 This is the end of file
```

Si nous cherchons des chaînes plus spécifiques, nous n'obtiendrons que les lignes correspondantes, comme par exemple le modèle *line* (lignes 01, 02 et 03) ou *end* (uniquement présent dans la ligne 04). Dans le même ordre d'idée, il est possible d'utiliser la commande chercher sur un système fonctionnant avec Windows, qui utilise la même syntaxe :

```
C:\Documents and Settings\mruef>
find data.txt "end"
04 This is the end of file
```

Utilisons maintenant cette technique de filtrage de modèles sur une réponse d'une connexion établie. Vous trouverez ci-après les données de sortie d'une connexion établie sur le port TCP 21 de l'hôte *ftp.computec.ch* :

**Tableau 1.** Modèles simples destinés à reconnaître les services

Nom	Proto- cole	Port	Déclencheur
Cisco PIX Firewall SMTP	tcp	25	^220.**********
Dict	tcp	2628	^220.* dictd
FTP	tcp	21	^220.*\n ^220.*FTP
ftp-darwin	tcp	21	^220 Inactivity timer
Giftp	tcp	21	^220.*SSH
Hylafax	tcp	4559	^220.*hylafax
NNTP	tcp	119	^200.*NNTP
Oracle HTTPS	tcp	1526	^220- ora
POP3	tcp	110	^220.*poppassd 500 Pas- sword
SMTP	tcp	25	^220.*\n250 ^220.*SMTP
TrendMicro InterScan SMTP	tcp	25	^220.*InterScan
vmware-authd	tcp	902	^220 VMware Authentication

```
C:\Documents and Settings\mruef>
telnet ftp.computec.ch 21
220 ProFTPD 1.2.10 Server
(ftp.computec.ch) [80.74.129.35]
```

Une méthode simple de filtrage de modèles à utiliser pour un application mapping consiste à utiliser le nom du protocole [Ruef 2004, 2006]. La chaîne `FTP`, par exemple, pourrait servir d'indice selon lequel le port ciblé serait un service FTP (File Transfer Protocol). La même technique peut s'appliquer au protocole SMTP pour les messageries électroniques et au protocole HTTP pour les implémentations de serveurs Web. Lorsque vous utilisez ces acronymes comme modèles, vous constaterez des correspondances avec FTP, mais pas avec SMTP ni avec HTTP. Vous pouvez donc en déduire que la connexion établie relève probablement du protocole FTP, ce qui correspond au port cible utilisé, recommandé par l'IANA pour ce genre de communications : port TCP (`ftp-control`).

Fausses alertes dans le filtrage de modèles

Bien que le filtrage de modèles soit utilisé dans de nombreuses disciplines, il faut à chaque fois tenir compte de deux problèmes fondamentaux que pose cette méthode :

- [Database] seuls les modèles définis en tant que tels seront trouvés ;
- [deflection] lorsque le modèle est strictement défini, il ne détecte pas de correspondances (fausses alertes) ou reconnaît des modèles incorrects (fausses alertes).

Le premier problème n'est pas vraiment important, contrairement aux détections d'intrusions d'anti-virus électroniques. En raison des spécifications et des normes strictes dans ce domaine, il est assez facile de définir ces défauts. Il existe toutefois des problèmes plus graves lorsqu'il s'agit de gérer la précision de la cible au moyen d'expressions modèles.

Nous supposons que SMTP indiquera le protocole Simple Mail Transfer Protocol (RFC 821), ce qui est normalement le cas, puisque la plupart des implémentations de type SMTP s'identifient grâce à leur nom et au protocole de l'application. Mais que se passe-t-il lorsque nous utilisons un mappage SMTP sur un serveur Web ? Nous forçons ce dernier, en quelque sorte, à nous renvoyer quelque chose, et plus particulièrement un document Web. En effet, il est possible d'y détecter la chaîne SMTP, si par hasard il s'agit d'un document sur les connexions à une messagerie

électronique et les normes qui s'y appliquent. Le filtrage de modèles permettra désormais d'identifier la chaîne SMTP en tant que protocole d'application éventuellement utilisé.

Lorsque le application mapping est en grande partie automatisé, cette technique peut ramener de faux résultats. Pour éviter cet écueil, il faut avoir recours à des techniques de application mapping bien plus complexes. Il est possible, par exemple, de définir des champs d'application plus clairs grâce aux expressions régulières, afin d'obtenir des correspondances uniquement en cas de présence de la chaîne en question. l'implémentation proposée par Amap s'efforce de régler ce problème par une analyse de la longueur des réponses. Vous trouverez une explication plus détaillée dans la partie consacrée aux fonctionnalités de l'outil Amap.

Dans notre exemple de application mapping à la recherche de la chaîne, nous pourrions, par exemple, limiter les recherches aux premières lignes de la réponse. Un en-tête SMTP est en effet chargé de lancer le message d'accueil, contenant les informations visées, une fois la connexion établie (soit les premiers 100 octets). Une connexion HTTP à un serveur Web contenant un document avec la chaîne SMTP, retournera cette chaîne après l'en-tête HTTP (soit après 250 octets environ). Ainsi, il est possible d'exclure une complication, ou du moins d'en minimiser les effets.

Il existe toutefois des situations où il est nécessaire d'avoir recours à des expressions régulières plus complexes. C'est le cas, par exemple, des chaînes dynamiques attendues, assez semblables à leur structure de base. Le service horaire et jour, pouvant être proposé sur les protocoles TCP et UDP, illustre à merveille ce cas de figure. L'IANA recommande le port 13 pour ce genre de services. En guise de réponse à la connexion établie, le service horaire et jour donne l'heure et la date du moment, dans un format compréhensible des hommes. S'il s'agit d'un système d'exploitation Unix, la réponse serait du genre suivant (ligne 02) :

Tableau 2. Commandes supportées selon les protocoles les plus utilisés

	FTP	SMTP	POP3	NNTP
HELP	X	X		X
USER	X		X	
PASS			X	
PORT	X			
STOR	X			
HELO		X		
VRFY		X		
EXPN		X		
RSET		X		
NOOP	X	X		
QUIT	X	X	X	X

```
01 C:\Documents and Settings
\mruef>telnet 192.168.0.10 13
02 Mon Nov 28 13:42:23 2005
```

Avec Amap, une des premières applications automatisées de la technique de application mapping, le fichier de réponse appdefs.resp présenterait la ligne suivante, capable de capturer la sortie mentionnée :

```
daytime-unix:::26:^(A-Z).*
[A-Z].* [0-3].
[0-9][0-9]:[0-9][0-9]:
[0-9][0-9] 200.\rn
```

La configuration de ce modèle destiné aux sorties du service horaire et jour sur un environnement Unix est assez exacte sans être encore parfaite. Toutefois, les possibilités d'optimisation restent limitées en raison de la configuration des données de sortie d'un service horaire et jour. Tout d'abord, le jour est donné sous forme de chaîne à trois lettres (par exemple, Mon pour Monday, Tue pour Tuesday, etc.). L'expression régulière peut bien reconnaître ce genre de modèle, mais la longueur de cette chaîne reste indéfinie. En théorie, des données de sortie étendues de type *Monday Nov 28 15:03:23 2005* seraient également reconnues. Le problème est le même en ce qui concerne l'affichage des mois avec trois lettres (par exemple, Nov pour November, et Dec pour December). Toutefois, les implémentations Unix restent très standard,

aucune erreur de reconnaissance n'est possible dans ces champs. La configuration de l'année pose beaucoup plus de problèmes, puisque celle-ci est limitée entre 2000 et 2009. Un système mal configuré (par exemple resté en 1984) ne sera pas reconnu correctement avec cette expression.

Même avec ce genre de réponses dépourvues d'information sur le système d'exploitation utilisé, comme le jour et l'heure, par exemple, notre sous-programme d'analyse est capable, en réalité, de détecter le système utilisé. La base de données d'empreintes d'Amap contient cinq entrées différentes pour l'heure

et le jour, dont deux reconnaissent les systèmes d'exploitation Unix (lignes 01 et 03) et une, les systèmes Windows (ligne 02). Les autres entrées sont génériques.

```
01 daytime-unix:::26:^(A-Z).* [A-Z]
[0-3].* [0-9][0-9]:[0-9][0-9]:
[0-9][0-9] 200.\rn
02 daytime-windows:::26-50:^(
[A-Z][a-z]+, [A-Z][a-z]+ [0-9]+,
200[0-9] [0-9]+:[0-9]+:[0-9]+
\x0a\x00
03 daytime-unix:::20-36:^(
[A-Z][a-z]+ [A-Z][a-z]+
[0-9] [0-9] [0-9]+:[0-9]+:
[0-9]+ 200[0-9]\x0d\x0a
04 daytime:::25-30:^(
[0-9][0-9] [A-Z][A-Z][A-Z]
200[0-9] [0-9][0-9]:
[0-9][0-9]:[0-9][0-9] .*
05 daytime:::26-45:^(
[A-Z][a-z][a-z]*,
[A-Z][a-z][a-z]* [0-9]+, 200
```

Codes d'état à trois lettres

De nombreuses applications réseau développées dans un souci d'interactivité directe avec l'utilisateur, ont recours au principe de codes d'état. Chaque entrée ou sortie est liée à un code d'état numérique, contenant en règle générale trois chiffres. L'ensemble des protocoles classiques,

Tableau 3. Comportement de temporisation typique de plusieurs services

Comportement de temporisation	Exemples
Aucun délai d'attente	Telnet, SSH, NNTP, Echo, Discard, Chargen
Interruption de la connexion après le délai d'attente	FTP, netbios-ssn (ca. 30 sec)
Interruption de la connexion après certains types de commande(s)	HTTP, HTTPS/SSL, Finger, Time
Interruption de la connexion après la dernière commande	HTTP, HTTPS/SSL, Finger, Time
Interruption de la connexion après une fausse commande	Certaines implémentations de type SMTP, certaines authentifications de terminal (comme Telnet ou SSH, par exemple), serveurs mandataires avec listes d'accès
Commande NOOP après délai d'attente	FTP



comme FTP et SMTP, utilisent la valeur 220 afin d'indiquer une requête traitée avec succès. Les programmes appliquent le même principe : le premier chiffre indique la catégorie (par exemple, traitement réussi), le second, le type de message (par exemple ressource trouvée), et le troisième, une description plus particulière (par exemple, ressource demandée envoyée). Seules des nombres entiers sont utilisés ici, dans la mesure où les clients peuvent plus facilement les traiter. Il leur suffit de lire les trois premiers octets d'une sortie afin de connaître l'état de l'opération (autre-

ment, une reconnaissance complexe de mots serait nécessaire).

Juste après ces trois chiffres vient un texte écrit sur l'état en question sur la même ligne, après un espace. Ce texte est toujours présent lorsque le service fonctionne sans client, ce qui permet d'éviter une interprétation automatisée du code d'état (par exemple, pendant une session Telnet interactive). Une communication interactive réussie avec le protocole FTP, pourrait, par exemple, se traduire par *220 Found* dans la première ligne. D'autres codes d'état peuvent se référer à des erreurs présentés chez le

client ou du côté serveur, ou encore indiquer une absence d'implémentation.

Jetons un coup d'oeil à ce modeste exemple d'une session FTP :

```
01 C:\Documents and Settings\mruef>
ftp 192.168.0.10
02 Connected to 192.168.0.10.
03 220 debian.computec.ch FTP server
   (Version 6.4/OpenBSD/
   Linux-ftpd-0.17) ready.
04 User (192.168.0.10:(none)): mruef
05 331 Password required for mruef.
06 Password: xxxxxxxx
07 230- Linux debian 2.6.12-1-386 #
   1 Tue Sep 27 11:02:18 JST
   2005 i686 GNU/Linux
08 230 User maru logged in.
09 ftp>
```

Nous commençons par établir une connexion FTP entre notre machine fonctionnant avec Windows et le système ciblé 192.168.0.10 (ligne 01). Notre client FTP indique une tentative de connexion réussie (ligne 02). Vous pouvez voir dans la première sortie du serveur (ligne 03) le message d'accueil. Vous pouvez également lire le code d'état préfixé 220 à trois chiffres, indiquant un accès réussi. Après avoir entré le nom d'utilisateur mruef (ligne 04), notre mot de passe nous est demandé (ligne 05). Vous pouvez lire le code d'état 331 qui indique une authentification exigée.

Ce principe de code d'état, utilisé par différents systèmes, peut se révéler utile en termes de application mapping. Ces types sémantiques indiquent généralement des protocoles ASCII NVT interactifs de type FTP, SMTP ou POP3 [Stevens 1994, Ruef et al. 2002]. Nous avons exposé dans le Tableau 1 les protocoles connus de la couche d'application qui utilise le principe des codes d'état. La dernière colonne recense les déclencheurs après lesquels le type spécifique de protocole peut être identifié. Nous les avons présentés aussi simplement que possible, afin de vous en faciliter la compréhension. Toutefois, il est possible de les optimiser afin de minimiser le retour de fausses réponses.

Tableau 4. Modèles complexes d'Amap

Nom	Protocole	Longueur	Modèle
Dante	tcp	2	\x05\x02
Daytime-unix	tcp/udp	26	^[A-Z].* [A-Z].* [0-3].* [0-9][0-9]
Daytime-windows	tcp/udp	26-50	^[A-Z][a-z]+, [A-Z][a-z]+ [0-9]+, 200[0-9] [0-9]+
Daytime-unix	tcp/udp	20-36	^[A-Z][a-z]+ [A-Z][a-z]+ [0-9][0-9] [0-9]+
Daytime	tcp/udp	25-30	^[0-9][0-9] [A-Z][A-Z][A-Z] 200[0-9] [0-9][0-9]
Daytime	tcp/udp	26-45	^[A-Z][a-z][a-z]*, [A-Z][a-z][a-z]* [0-9]+, 200
Dhcp3d-isc	tcp	8	^\x00\x00\x00\x64\x00\x00\x00\x18
Dns-pdnsd	tcp/udp	2	^\x00\x0c
Finger	tcp	1	\x66
http-compaqinsightmanager	tcp	6	^HTTP/1
http-iis	tcp	34	^<h1>Bad Request .Invalid URL.</h1>
Mailhurdle	udp	10	\x00\x08\x06\x9f\x7a\x06\x00\x00\x00\x00
Mldonkey	tcp	1	\x31
ms-distribution-transport	tcp(/udp)	6	^..\x0a\x00
Ms-dtc	tcp/udp	3	..\n
Netmeeting	tcp	4	^\x03\x00\x00\x11
NTP	udp	48	^...\x00\x00..\x00\x00
QOTD	tcp/udp	5-1000	^[A-Z].* .* *"[?]"\r\n[A-Za-z].*\r\n
SNMP	tcp	3	\x41\x01\x02
Spamd	tcp	1	\x32
SSL	tcp	1	\n
Time	tcp/udp	4	^\xc[2-5]

Commencer par une requête formelle

Il existe plusieurs services toutefois pas aussi généreux dépourvus de message d'accueil après une connexion établie. On peut parler d'implémentations assez strictes ou peu conviviales du moins du point de vue de l'hôte. De telles communications se présentent sous la forme d'écrans noirs après la connexion. Le serveur (application réseau) attend des requêtes supplémentaires avant d'entrer en action. Si aucune requête n'est enregistrée au bout de plusieurs secondes ou minutes, le délai d'attente est dépassé provoquant une interruption automatique de la connexion. De cette manière, l'utilisation de ressources dans le cas de connexions persistantes est totalement évitée.

Parmi les implémentations de protocoles d'applications les plus répandus restant en attente d'une requête formelle, citons le fameux protocole HTTP (Hyper Text Transfer Protocol). Jetons un coup d'oeil à l'exemple ci-dessous dans lequel notre client Telnet tente de se connecter à l'hôte 192.168.0.10 sur le port TCP 80 :

```
01 C:\Documents and Settings\mruef>
   telnet 192.168.0.1 80
02 HEAD / HTTP/1.0
03
04 HTTP/1.1 200 OK
05 Date: Mon, 28 Nov
   2005 16:39:07 GMT
06 Server: Apache/
   1.3.34 (Debian)
07 Connection: close
08 Content-Type: text/
   html; charset=iso-8859-1
09
10
11
12 Connection to host lost.
```

Après avoir lancé la connexion (ligne 01), le client doit formuler une requête. Nous savons, toutefois, qu'il s'agit d'une implémentation d'un serveur HTTP. Nous avons donc utilisé la syntaxe appropriée. Grâce à la requête „HEAD / HTTP/1.0“ (complétée

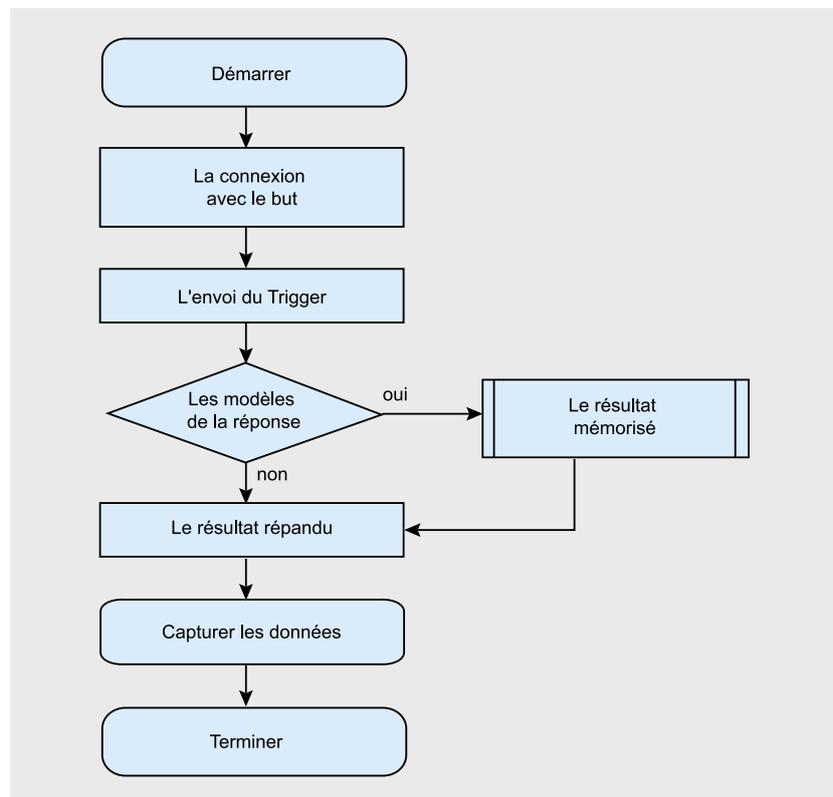


Figure 1. Fonctionnement d'un application mapping basé sur le mécanisme action/réaction

de deux nouveaux signes de lignes ; lignes 02 et 03), nous provoquons une réponse (lignes 04 à 11). C'est ici que vous pouvez voir les indicateurs typiques du protocole HTTP (par exemple, réponse commençant par la chaîne HTTP, et une ligne débutant pas Server:). Sans notre impulsion initiale via la requête HTTP HEAD, rien n'aurait été possible.

Le service du port ciblé est inconnu dans le cas normal d'un application mapping. Il est donc impossible de formuler la requête adéquate dès la première tentative, comme nous l'indiquons dans l'exemple ci-dessus. Dans un tel cas de figure, il est recommandé, en guise de première tentative, d'appuyer sur le bouton *entrée* plusieurs fois de suite afin de pro-

Tableau 5. Pondération des indicateurs au cours de la reconnaissance

Nom	Poids	Description
Numéro de port	5,00%	S'il existe un service proposé sur le port testé pour lequel l'IANA recommande ce port comme port standard.
Protocole de transport	5,00%	Si le protocole de transport est utilisé pour une implémentation standard (TCP pour HTTP, et UDO pour TFTP, par exemple)
Actions d'accueil	5,00%	Si le protocole de transport possible utilise des messages d'accueil automatisés ou pas.
Actions de temporisation	5,00%	Si l'application génère un délai d'attente automatique après un certain temps.
Réponse sans action	35,00%	Sorties du message d'accueil reçues automatiquement une fois la connexion établie
Réponse avec action	45,00%	Sorties d'une réponse provoquée

**Listing 1.** Implémentation basique d'un outil de application mapping

```
01  #!/bin/sh
02
03  echo "pmap v1.0"
04
05  if [ $# == 2 ]; then
06      echo "Starting protocol mapping ..."
07      RESPONSE=`echo -e "QUIT\n" | nc -vv -w 3 $1 $2`
08
09      echo -n "Testing for trigger ftp ... "
10      MATCH=`echo "$RESPONSE" | grep ftp`
11      if [ "$MATCH" ]; then
12          echo "Found!"
13      else
14          echo "Not found."
15      fi
16  else
17      echo "Please use the following syntax:"
18      echo "pmap.sh <dhost> <dport>"
19  fi
```

voquer un message d'erreur (comme par exemple *non supporté* ou *syntaxe inconnue*), lequel contient de nombreuses informations utiles à l'analyse de protocoles. Toutefois, à en croire les statistiques, les implémentations de type HTTP restent les plus répandues, en raison de leur simplicité, même lorsqu'elles sont propriétaires. Ainsi, des entrées de type HTTP telles que `GET / HTTP/1.0` peuvent retourner de bons résultats.

Impulsions et réactions

Nous venons d'apprendre que les applications, visant l'interactivité, se caractérisent par l'envoi d'un court message d'accueil au client une fois la connexion établie. Mais, nous avons également constaté la présence de plusieurs services beaucoup plus sévères comme le protocole HTTP, qui ne réagissent que lorsqu'une requête leur est envoyée. Ces dernières implémentations plus strictes nous intéressent plus particulièrement aussi. C'est ici l'occasion de voir dans quelle mesure le application mapping peut être optimisé afin de provoquer des réactions uniques.

Une possibilité classique dans ce cas consiste à utiliser la fonction aide qui ne brille pas par sa performance. Les services interactifs primaires de type FTP ou SMTP proposent une commande aide (`help`), chargée de lister les commandes supportées.

Grâce aux sorties de la commande aide et de la liste des commandes, il est également possible d'identifier le protocole d'application utilisé. Nous avons exposé dans la Tableau 2 les commandes que vous pouvez trouver sur la plupart des protocoles d'applications. Cette liste n'est pas exhaustive, ni du point de vue des protocoles ni de celui des commandes.

Juste après un message, si une commande existe ou pas, il est possible de se faire une idée précise du protocole utilisé. Il faut toutefois être attentif : certaines implémentations de serveurs évitent d'avoir recours à plusieurs commandes pour des raisons de sécurité ; certains serveurs SMTP récents, par exemple, ne supportent pas les commandes classiques de type `VERFY`, `EXPN`, `DEBUG` ou même `HELP`. Il n'est donc pas rare d'avoir des messages d'erreur sur un accès interdit ou sur une commande non installée, ce qui bien sûr peut engendrer de faux résultats, si vous vous appuyez sur de telles caractéristiques.

Si plusieurs protocoles sur une couche d'application supportent la même commande, il faut lancer une analyse plus poussée. Pour ce faire, il suffit de tester les implémentations au niveau de leurs paramètres de gestion. Par ailleurs, les réponses habituelles retournées peuvent différer. De nombreux serveurs FTP, par exemple, sont dotés de paramètres

relatifs à la commande aide. Taper la requête `HELP PORT` peut engendrer des informations particulières sur la commande FTP PORT. Le protocole SMTP agit souvent de la même façon.

Comportement de temporisation

Nous venons de voir que l'action de base de lancer un message d'accueil pouvait caractériser une application réseau ainsi que son protocole. La manière dont est accueilli le client peut indiquer une application de nature interactive (par exemple SMTP) ou non (par exemple HTTP).

Une autre caractéristique semblable peut également révéler la présence d'un service spécifique. Il s'agit du comportement de temporisation. Un délai d'attente se met généralement en place dans une application en l'absence d'échange de données pendant un certain temps. Ce délai d'attente peut durer plusieurs millisecondes jusqu'à quelques heures parfois. Les applications dites semi-interactives présentent en règle générale un délai d'attente de plusieurs secondes (par exemple 10 secondes), ou de plusieurs minutes (par exemple 2 minutes). Il existe plusieurs états qu'une analyse du délai d'attente peut révéler.

Ce délai d'attente ne dépend pas toutefois du protocole de l'application, mais de son implémentation et de sa configuration. De nombreux services autorisent un paramétrage confortable (ou dissimulé dans un fichier de configuration) de cette valeur du délai d'attente. Voici listés ci-après six états différents.

Vous aurez bien évidemment compris que le application mapping devient très difficile en cas d'absence de délais d'attente ou en cas de délais d'attente assez long. A l'instar d'une dérivation de preuve logique, il est plutôt difficile d'être certain de l'existence d'un de ces états, lorsqu'aucun n'est visible. Par ailleurs, vérifier l'état d'un délai d'attente est une opération naturellement très longue. C'est une des raisons pour lesquelles il est déconseillé d'implémenter une technique de ce type. Et, si une telle implémentation se révèle réellement nécessaire,

il vaut mieux choisir une valeur de courte durée pour la vérification. Il existe certains programmes qui n'ont pas recours au délai d'attente comme les services classiques echo, discard et chargen. De même, des émulations de terminaux, comme Telnet et SSH ne sont pas souvent interrompues.

L'interruption automatique de connexion se déclenche généralement après une certaine durée limitée. Les communications FTP constituent un excellent exemple puisqu'elles ferment une connexion du côté serveur après un certain temps (ou après l'utilisation répétée de la commande NOOP).

L'interruption de connexion dite semi-automatique est légèrement plus originale puisqu'elle surgit après l'entrée d'une commande et génère une réponse correspondante. Ce phénomène s'observe généralement dans le cas de connexions orientées session, dont l'objectif est d'essayer de minimiser le plus possible les connexions persistantes. Le protocole HTTP est un des exemples les plus connus. Il provoque la fermeture d'une connexion par le serveur après avoir répondu à une requête. Parmi les implémentations de type similaire, citons les services horaire et jour, heure et QOTD.

Certains services plus interactifs ferment automatiquement la connexion lorsque le client ne répond pas à plusieurs exigences. Certains serveurs, par exemple, clôturent une tentative de connexion vers une session de terminal avec Telnet ou SSH, lorsque le mot de passe est incorrectement entré plusieurs fois de suite. Certains serveurs SMTP stoppent les communications très rapidement lorsque le client tente d'utiliser une commande non installée. De tels comportements dépendent largement de la conception de l'application et n'ont pas de lien avec son protocole.

Enfin, certains délais d'attente sont engendrés par l'utilisation de la commande NOOP générée automatiquement. NOOP signifie No Operation (aucune opération). Cette commande est utilisée régulièrement afin d'éviter une interruption automatique via un délai d'attente. Le client

ou le serveur peuvent ainsi rester connectés en échangeant de très faibles quantités de données. Plusieurs applications, comme FTP, par exemple, ont recours à ce mode opératoire. Toutefois, après un usage répété de la commande NOOP pendant quelques minutes, le serveur ferme la connexion automatiquement.

L'analyse des comportements de temporisation n'est pas aisée, car elle demande beaucoup de temps, l'élaboration de tests séparés et reste souvent non définissable. C'est la raison pour laquelle les développeurs n'implémentent pas ce genre d'analyse (c'est le cas d'Amap, par exemple). Rien ne porte à croire que cette tendance va changer, dans la mesure où le rapport entrée/sortie d'un tel rapport ne va que dans un sens.

Techniques automatisées avec Amap

Amap est l'une des premières implémentations automatisées de application mapping. Son nom s'inspire du célèbre utilitaire de scan et de rapport Nmap de Fyodor. Amap est distribué sous la licence General Public License (GPL), et a été développé pour les systèmes UNIX/Linux par Van Hauser et DJ RevMoon du groupe allemand THC (The Hacker's Choice, <http://www.thc.org>).

L'utilisation de base de cet outil est assez similaire à celle de Nmap. Ici aussi, le premier argument à entrer est le nom ou l'adresse IP du système ciblé. Le deuxième paramètre doit désigner un numéro de port habituellement compris entre 1 et 65,535. Si l'option -u reste inutilisée, l'accès standard est paramétré sur le port TCP. Sinon, le port UDP est utilisé comme protocole de transport. Si seuls les protocoles consacrés doivent être testés, il vaut mieux essayer l'option -p <protocol>, permettant d'indiquer les ports qui vous intéressent (par exemple ftp ou smtp).

Le déclencheur -d est strictement réservé aux développeurs, aux testeurs d'intrusions, et aux corrections de bogues. Grâce à ce déclencheur, Amap doit pointer les réponses du

port ciblé directement sur *stdout*. En affichage hexadécimal et ASCII, vous pouvez déchiffrer les réponses pour une analyse exacte qui déclenchera une réaction particulière, afin d'adapter le filtrage de modèles. L'option -b est très semblable. Elle permet, en effet, de sortir l'ensemble des réponses ASCII de déclencheurs réussis.

Outre le fichier binaire d'Amap, véritable coeur de l'outil, trois fichiers modulaires de bases de données sont également utilisés. Ces fichiers, sauvegardés sous forme de fichiers texte classiques, peuvent être consultés dans le répertoire `/usr/share/amap`. Le fichier intitulé `appdefs.trig` est extrêmement important ici ; c'est lui qui fournit les déclencheurs pour les impulsions. Ainsi, pour le test réalisé sur le protocole SMTP, par exemple, sur le port standard `port tcp/25`, l'impulsion `HELO AMAP\r\n` est utilisée.

Autre fichier important, le fichier `appdefs.resp`, contenant les réponses possibles de services spécifiques comme les expressions régulières. Amap attend, par exemple, en tant que réponse d'un serveur FTP, certaines chaînes de type `Login name: or ^220.*FTP`. Il n'existe aucune expression complexe, intégrées ou récursive. Plus

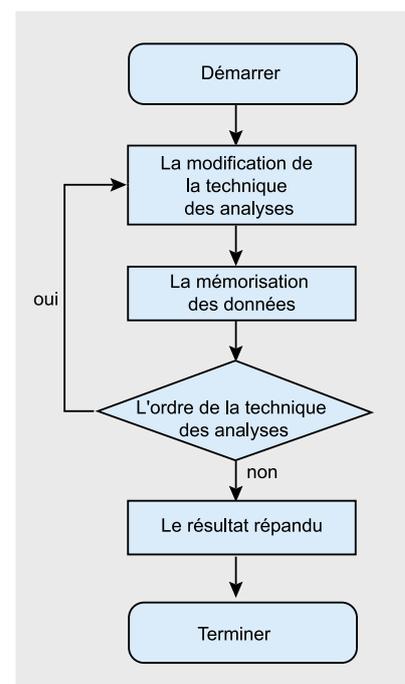


Figure 2. Application mapping intelligent



sieurs lignes sont toutefois disponibles pour les services réagissant à des sorties particulières. Le fichier indépendant `appdefs.rpc` permet d'analyser les réactions sur les ports RPC. Ce fichier contient, en effet, plusieurs identifiants de services RPC connus.

Automatisation avec les sorties de Nmap

Amap propose une fonction particulière non mentionnée (même pas au moment de développer vos propres solutions). Cette fonction permet d'analyser la complexité d'une réaction. Vous trouverez dans la quatrième colonne du fichier de réponses `appdefs.resp`, des valeurs d'entiers indiquant la longueur attendue des réponses en question. Ces indications complémentaires permettent de garantir l'exactitude de l'analyse.

Développer votre propre solution

Vous venez de découvrir les fonctionnalités techniques et relatives aux connexions du application mapping, ainsi qu'une possible implémentation de cette technique avec l'exemple de la société THC, Amap. Nous vous proposons donc maintenant, d'essayer de créer votre propre implémentation. Nous allons donc vous présenter quelques exemples génériques, dont nous analyserons la conception et l'implémentation particulière. L'objectif ici n'est pas de rechercher la manière la plus parfaite de créer votre propre logiciel, mais de vous donner quelques pistes.

Nous allons donc nous concentrer sur une procédure d'interpréteur de commandes assez simple. En dépit d'une performance et d'une ergonomie peu remarquables, ce script est facile à expliquer et à utiliser. Il est écrit dans un langage de script générique destiné à un large public. Notre modeste projet de développement nécessite les éléments suivants :

- [Software-Meaning] Implémentation d'une application pour un mappage d'application semi-automatique,

- [Complexity] solution par ligne via une procédure d'interpréteur de commandes simple,
- [Pattern-Matching] filtrage de modèle simple appliqué sur les réponses.

Comme vous pouvez le constater à partir de la liste des exigences ci-dessus, nous n'utiliserons pas certaines techniques d'analyse possibles (comme, par exemple le comportement de temporisation). Notre méthode s'appuie sur une simple manipulation des réponses provoquées par des déclencheurs, dans lesquelles nous allons chercher des modèles sauvegardés. Le graphique suivant résume ce comportement très simple (voir la Figure 1).

Nous allons commencer par lancer le test pour voir si le port ciblé est en mode écoute. Si ce n'est pas le cas, l'état sera sauvegardé et le programme se fermera en générant un message d'erreur. Si au contraire nous pouvons établir une connexion vers le port ciblé, nous pourrons lancer tout d'abord un déclencheur, censé provoquer une réponse. Nous pourrons ensuite analyser cette dernière grâce à un filtrage de modèles. Si le modèle recherché est présent dans la réponse, le résultat est sauvegardé puis préparé pour être émis en sortie. Avant de terminer l'application, nous obtenons les données de sorties résultant du processus de scan.

Implémentation d'une simple procédure d'interpréteur de commandes

Nous allons vous présenter dans cette partie une implémentation plutôt basique des exigences mentionnées plus haut, permettant d'obtenir un outil semi-automatique orienté par lignes. Nous allons donc utiliser une procédure d'interpréteur de commandes, composée de 19 lignes (voir le Listing 1).

À la ligne 03, nous retournons le titre de notre application à chaque départ. Nous appellerons notre implémentation `pmap`, pour désigner l'expression *protocol mapping*

(mappage de protocole), dirigée vers Nmap et Amap.

À la ligne 05, vous pouvez constater que nous avons utilisé une requête de type SI afin de contrôler le nombre d'arguments donnés. S'il n'y a pas deux arguments donnés, la requête bascule sur autre à la ligne 16. Dans ce cas un message d'erreur retourné, en expliquant la syntaxe correcte, et met fin au processus (lignes 17 à 19).

S'il y a exactement deux arguments donnés, le programme lance alors le application mapping (ligne 06). La ligne 07 constitue en quelque sorte le coeur de notre programme. Ici, l'utilitaire de réseau NetCat est utilisé afin d'établir une connexion au système ciblé (premier argument `$1`) sur le port visé (second argument `$2`). Si la connexion peut être établie, de nouveaux signes par ligne sont envoyés par ce canal. Le résultat de cet accès est sauvegardé dans la variable temporaire `$RESPONSE`.

Ensuite, il faut contrôler la présence du déclencheur `ftp` dans la variable `$RESPONSE`. Pour

le faire, nous utilisons la commande `grep`, afin de lancer le déclencheur spécifique sur les données sources puis sauvegarder le tout sur une autre variable temporaire intitulée `$MATCH` (ligne 10).

Le processus central de `pmap` va désormais contrôler si la variable `$MATCH` contient ou non quelque chose (lignes 11 à 15). Si la commande `grep` n'a pas ramené de résultats, la variable ne contient alors aucune entrée. Toutefois, si une ligne est effectivement trouvée, elle sera sauvegardée dans la variable. Selon les correspondances trouvées, un état spécifique sera retourné (lignes 12 à 14). Un lancement réussi (lignes 01 à 04) et non réussi (lignes 06 à 09) de notre modeste script apparaîtra comme suit sur un système Linux :

```
01 mruef@debian:~$ ./pmap.sh ftp.  
    computech 21  
02 pmap v1.0  
03 Starting protocol mapping ...
```

```

04 Testing for trigger
    ftp ... Found!
05
06 mruef@debian:~$ ./
    pmap.sh smtp.computec.ch 25
07 pmap v1.0
08 Starting protocol mapping ...
09 Testing for trigger
    ftp ... Not found.

```

Comme nous l'avons déjà mentionné précédemment, cette implémentation est très basique afin de vous expliquer les principes sous-jacents de manière assez simple. Une application professionnelle est dotée de plusieurs contrôles d'erreurs, nécessitera de nombreux tests et une fonction de rapport plus complète. Toutefois, les principes de base restent les mêmes en règle générale.

Algorithme de fiabilité

Le application mapping repose sur un principe de base facile à comprendre et une implémentation simple peut être rédigée en quelques lignes. Des applications comme Amap peuvent vraisemblablement être fortement optimisées en implémentant certains mécanismes complémentaires de rapport. Cette analyse extrêmement différente des caractéristiques d'une connexion ou des réactions d'un service doit pouvoir s'appuyer sur un algorithme capable de couvrir les pondérations et les dépendances logiques. Nous avons exposé dans la Tableau 5 les éventuelles techniques possibles, leur poids logique dans l'analyse et les parties qu'elles peuvent cibler.

Nous souhaitons vous montrer ici le lancement possible de cet algorithme, étape par étape, ce qui devrait vous servir dans la pratique :

- si le port ciblé est recommandé par l'IANA pour un service R, cette valeur R doit être sauvegardée avec une pondération logique de 5%. $A = \{ R, 5 \}$;
- si le protocole de transport standard est utilisé pour cette connexion, la valeur est sauvegardée avec une pondération logique de 5%. $B = \{ R, 5 \}$;

- l'action d'accueil générée via un message de bienvenue une fois la connexion établie est sauvegardée avec une pondération de 5%. $D = \{ R, 5 \}$;
- l'action de délai d'attente est sauvegardée avec une pondération de 5%. $R = \{ R, 5 \}$;
- la réponse automatique une fois la session établie est contrôlée via un filtrage de modèles ; les éventuels résultats sont sauvegardés avec une pondération de 35%. $E = \{ R, 35 \}$;
- les réponses provoquées par des actions sont également contrôlées via un filtrage de modèles ; les éventuels résultats sont sauvegardés avec une pondération de 45%. $F = \{ R, 45 \}$;
- en déterminant les services ayant la pondération la plus forte, nous connaissons alors les résultats les plus probables. $Z = \{ A, B, C, D, E, F \}$ |.

Afin d'implémenter cet algorithme, vous pouvez utiliser une base de données, virtuelle avec des tableaux, ou une base de données de fichiers plats. Grâce à des opérations arithmétiques simples (additions, divisions), il est ensuite possible de calculer les moyennes et les probabilités basées sur les pondérations logiques.

Contre mesures

Voici nous sommes arrivés à la fin du processus. Il est temps de nous demander dans quelle mesure les administrateurs concernés par la sécurité de leur système peuvent se protéger contre telles analyses automatisées ou manuelles comme le application mapping (avec Amap, par exemple). En règle générale, les mêmes techniques peuvent être utilisées telles quelles pendant des années par les programmeurs de virus ou des testeurs d'intrusions extrêmement expérimentés afin de modifier leur comportement et laisser tourner à vide une analyse d'empreintes sur une base de données de modèles. Alors que la guerre fait rage entre les développeurs de virus et les programmes anti-virus, entre les testeurs

d'intrusions et les systèmes de détection d'intrusions, notre propos vise ici à combattre le processus d'analyse d'un application mapping.

Nous venons de voir que l'implémentation d'expressions régulières détaillées et efficaces relève d'un grand savoir-faire. Dans le même ordre d'idée, il est possible d'éviter le filtrage de modèles en modifiant le comportement d'un tel processus et donc les résultats retournés. Nous sommes toutefois tenus d'observer plusieurs règles interdisant l'adoption de protocoles ou leur réécriture. Les programmeurs de virus ne peuvent pas non plus modifier ne serait-ce qu'un seul octet X d'un programme anti-virus. Une telle intervention provoque en général des conséquences dévastatrices sur le comportement du produit en question, ce qui n'est pas dans notre intérêt.

Si vous observez l'exemple de la base de données d'Amap, vous constaterez que la plupart des contrôles reposent sur un nom de protocole dans une réponse. Par exemple, la chaîne SMTP indique l'utilisation du protocole Simple Mail Transfer. La même technique s'applique à FTP, HTTP et à bien d'autres protocoles. Les administrateurs de systèmes doivent donc veiller à ne pas laisser de côté telles informations somme toute assez basiques. Les messages d'accueil de certaines applications doivent notamment être modifiés, de manière à ne donner aucune information sur les fonctions de l'application. De cette manière, les pirates ne peuvent avoir recours qu'aux actions ciblées afin de provoquer certaines réactions. Avec des solutions automatisées comme Amap, ce procédé permettra de maîtriser la plupart des attaques. Sur la version 4.8 d'Amap, par exemple, cette technique permettra de rejeter 78 variantes d'accès réussis sur un total de 349 modèles, dont les plus répandus, comme FTP ou SMTP.

Une autre possibilité consiste à découpler les services pouvant être protégés au moyen d'une passerelle d'application via un serveur mandataire destiné à un tel processus [Ruef et al. 2002, Ruef 2004, 2006].



Il est impossible d'éviter directement la détermination réussie de certains protocoles d'applications proposées. Mais, dans la plupart des cas, il est possible de compliquer considérablement l'identification d'une implémentation spécifique (par exemple, IIS ou Apache).

Résumé

Grâce au application mapping, il est possible de détecter les protocoles utilisés par un service lié à un port ouvert via l'analyse de son comportement. Grâce à différentes méthodes de mappages disponibles, il est possible de distinguer diverses sortes d'applications, et parfois, dans certains cas, les caractéristiques de leur implémentation. Le filtrage de modèles appliqué sur les réponses des services contrôlés est une des méthodes les plus efficaces dans ce domaine.

Quoique considérée comme une discipline plutôt classique de la sécurité informatique, peu d'ouvrages ont été consacrés au application mapping par le passé. En effet, la reconnaissance de protocole n'a jamais été considérée comme une action utile à la sécurité informatique. Mais grâce à l'apparition d'outils comme Amap, première implémentation largement utilisée d'une solution automatisée, le application mapping commence à trouver sa place dans le domaine des audits de sécurité sur les réseaux.

Nous avons donc présenté les fonctionnalités de base d'Amap, pour inciter les lecteurs à développer leur propre version, inspirée des mêmes principes. Nous avons procédé de manière assez simple afin de mieux présenter les principes de l'implémentation en question. Nous avons, par ailleurs, évoqué quelques possibilités d'optimiser nos résultats pour plus de fiabilité. Grâce à l'ajout de techniques supplémentaires, il est possible d'obtenir certains résultats très précis.

Nous avons enfin parlé des efforts de certains administrateurs, particulièrement concernés par la sécurité, consistant à rendre complexe le application mapping auto-

matisé ou manuel. En changeant le comportement des services proposés, il est possible de laisser fonctionner des accès particulièrement automatisés, comme c'est

le cas avec Amap, un anonymat total. Le processus du application mapping demande alors beaucoup d'efforts, et de temps, et exige un travail manuel. ●

À propos de l'auteur

Marc Ruef travaille en tant que Consultant en Sécurité auprès de la société suisse SCIP (security, consulting, information, process) AG, et dirige le service Audit de Sécurité et Test d'Intrusions. En octobre 2002, son second livre intitulé *Hacking Intern*, a été publié par la maison d'édition Data Becker. Outre de nombreuses publications spécialisées en informatique et en sécurité informatique théorique, Marc Ruef supporte également certains projets internationaux dans ce domaine. Depuis 1997, il gère le site [Web compute.ch](http://www.compute.ch), qualifié d'une des plus importantes archives des publications allemandes consacrées à la sécurité informatique. Par ailleurs, il développe la solution Attack Tool Kit (ATK), cadre d'exploitation open-source, ainsi que d'autres logiciels comme Nessus, afin de faciliter les tests d'intrusions pour en optimiser la transparence. Vous pouvez contacter l'auteur à l'adresse : marc.ruef@compute.ch, <http://www.compute.ch/>

Sur Internet

- <http://www.compute.ch>, publications gratuites en sécurité informatique,
- THC amap, <http://thc.org/thc-amap/>, implémentation largement utilisée de la technique de application mapping,
- nmap, <http://www.insecure.org/nmap/>, utilitaire de scan et de rapport largement utilisé,

Bibliographie

- McClure, Scambray, Kurtz, 16 mai 2005, Hacking Exposed – Network Security Secrets and Solutions, McGraw-Hill/Osborne Media, ISBN 0072260815, 5ème édition, <http://www.amazon.de/exec/obidos/ASIN/0072260815/>, traduction allemande par Ian Travis *Das Anti-Hacker-Buch*, novembre 2005, Vmi Buch, ISBN 3826681673, <http://www.amazon.de/exec/obidos/ASIN/3826681673/>
- Packet watch, 19 février 2004, Identifying Services, <http://www.packetwatch.net/>
- Ruef, Marc, 1999, Security of Windows, security-guide.ch und compute.ch, <http://www.compute.ch/download.php?view.283>, réédité sur astalavista.com
- Ruef, Marc, 2003, Auditing with Linux – Discover the vulnerability of your system, scip monthly Security Summary, réédité sur compute.ch, <http://www.compute.ch/download.php?view.528>
- Ruef, Marc, février 2004, Seminar Computer security, University of Luzern, Master of Advanced Studies eLearning and management of knowledge, [compute.ch, http://www.compute.ch/download.php?view.481](http://www.compute.ch/download.php?view.481)
- Ruef, Marc, 17 avril 2006, Seminar computer security script, University of Luzern, Master of Advanced Studies eLearning and management of knowledge, [compute.ch, http://www.compute.ch/download.php?view.793](http://www.compute.ch/download.php?view.793)
- Ruef, Marc, Rogge, Marko, Velten, Uwe, Gieseke, Wolfram, novembre 2002, Hacking Intern – attacks, strategies, defense, Data Becker, Düsseldorf, ISBN 381582284X, <http://www.amazon.de/exec/obidos/ASIN/381582284X/>, 1ère édition en rupture de stock depuis mi 2004
- Stevens, Richard W., janvier 1994, TCP/IP Illustrated, Volume 1 : The Protocols, Addison-Wesley Professional, ISBN 0201633469, <http://www.amazon.de/exec/obidos/ASIN/0201633469/>, traduction allemande, TCP/IP, septembre 2003, Hüthig Telekommunikation, ISBN 3826650425, <http://www.amazon.de/exec/obidos/ASIN/3826650425/>