Autopsie d'une intrusion « tout en mémoire » sous Windows

Nicolas Ruff

EADS-IW SE/CS
nicolas.ruff@ eads.net

Résumé Cet article est un état de l'art sur les techniques d'autopsie dans le cadre d'une intrusion « tout en mémoire » en environnement Windows. Nous ne nous intéresserons pas ici aux techniques d'analyse « live » (exécution d'outils sur la machine compromise), qui ont pour inconvénient d'altérer grandement l'état du système ; mais plutôt à l'analyse « off-line » d'une copie de la mémoire physique. Malgré la jeunesse des techniques employées, nous verrons que les résultats obtenus sont plutôt encourageants.

1 Introduction

L'intrusion dite « tout en mémoire », c'est-à-dire sans laisser de traces sur le disque de la machine compromise, est passée du laboratoire à la réalité industrielle, accessible à tous, grâce à l'outil Meterpreter pour Metasploit. D'autres outils comme Immunity CANVAS et Core Impact offraient déjà cette fonctionnalité, mais elle restait réservée à des professionnels de par le coût élevé de ces produits.

Les procédures et outils d'analyse « classiques » (ex. EnCase, SleuthKit) sont très orientés « disque » et prennent rarement en compte le problème du « tout en mémoire ».

Les procédures de réponse aux incidents (lorsqu'elles existent) se classent le plus souvent en deux catégories. Dans le cas où la réponse doit être la plus rapide possible, il est d'usage d'analyser le système « live » avec des outils plus ou moins adaptés tels que NETSTAT, Process Explorer ou autres. Dans le cas où des suites judiciaires sont envisagées, il est d'usage d'éteindre brutalement la machine compromise jusqu'à l'arrivée des autorités compétentes.

Aucune de ces deux méthodes n'est adaptée à l'intrusion « tout en mémoire », qui ne pourrait être détectée en « live » que par des outils très intrusifs (ex. débogueur). C'est pourquoi de nouvelles méthodes et de nouveaux outils se sont avérés nécessaires - d'autant que depuis le rachat de Sysinternals par Microsoft, la licence d'utilisation de leurs outils a été restreinte.

Ces deux dernières années, le sujet de l'autopsie « tout en mémoire » en environnement Windows a connu une ébullition considérable, dans la lignée du challenge DFRWS 2005 [17]. Au cours de cette intervention, nous aborderons successivement les thèmes suivants :

- Les outils de collecte de la mémoire d'un système « live ».
- Les outils d'analyse.
- L'application dans différents cas de figure : utilisation de l'outil Meterpreter, challenge Securitech 2005 et challenge DFRWS 2005.
- Les contre-mesures connues.

2 Collecte des traces

Lors d'une intervention sur une machine potentiellement compromise, la démarche que nous allons explorer dans la suite de l'article se déroule en deux étapes : la collecte de la mémoire physique, aussi fiable que possible ; et son analyse, aussi exhaustive que possible.

La collecte de la mémoire sur le système cible est la partie la plus délicate de l'opération. En raison du caractère volatile de la mémoire, cette collecte ne peut se faire que sur un système qui a fonctionné de manière ininterrompue depuis l'incident. Il convient également de ne pas trop perturber le fonctionnement du système par les outils qu'on y exécute. En effet une simple allocation mémoire peut provoquer une défragmentation du tas...

Plusieurs modes opératoires ont été élaborés pour la collecte de la mémoire d'un système, ils ont tous leurs avantages et leurs inconvénients comme nous allons le voir ci-dessous.

2.1 La carte d'acquisition matérielle

Cette solution peut être considérée comme le fin du fin en matière de collecte de données sur un système. En effet cette tâche est confiée à une carte PCI qui accède matériellement à la mémoire physique pour en dupliquer le contenu sur un port dédié [25,20]

Les avantages de cette solution sont :

- La procédure de collecte n'altère pas les données collectées.
- Cette solution est résistante à la plupart des techniques de dissimulation rencontrées « dans la nature » (mais pas en laboratoire, voir [22]).

Toutefois cette solution n'est pas non plus sans inconvénients :

- Tout d'abord, cette solution n'est pas disponible dans le commerce actuellement, et les brevets qui la protègent ne facilitent pas l'émergence d'acteurs dans le domaine.
- La présence de la carte sur le bus PCI peut être détectée par un attaquant d'un niveau technique suffisant, voire contournée en manipulant astucieusement le *chipset*.
- Seules les machines équipées préalablement à l'intrusion sont analysables par cette méthode. C'est un inconvénient majeur, qui nous oblige à inventer d'autres méthodes de collecte.

Il est difficile d'en dire plus sur ces solutions, en l'absence de produit disponible « sur étagère » pour évaluation. Des questions restent en suspens, comme par exemple « l'activité du processeur est-elle stoppée lors de l'acquisition » ? Dans le cas contraire, l'image obtenue risque d'être incohérente.

2.2 Le bus Firewire

Pour les machines qui n'ont pas de carte d'acquisition intégrée, une autre solution matérielle est envisageable : le bus IEEE 1394, dit « FireWire », qui autorise un accès direct à la mémoire du système.

Cette technique a été initialement étudiée pour compromettre des machines à l'aide d'un iPod [21] modifié... Mais elle présente bien d'autres applications, dont celle qui nous intéresse ici.

Les avantages sont les mêmes que la solution précédente ; les inconvénients aussi, parfois en pire :

- Toutes les machines ne sont pas équipées « en standard » d'un port FireWire, en particulier les serveurs.
- Le système d'exploitation continuant à tourner en parallèle de l'acquisition, l'image obtenue peut être incohérente.
- Cette solution n'est pas infaillible, comme l'a démontrée Joanna Rutkowska [22].

La technique proposée par Joanna est basée sur la reprogrammation du *chipset* NorthBridge afin de masquer une partie de la mémoire vive aux périphériques connectés au SouthBridge. Il ne s'agit pas d'un *hack*, mais bien d'une fonction du *chipset*, ce qui rend l'attaque difficile à contrer. De plus sa conclusion est que l'avènement prochain des *chipsets* [15] va rendre ces attaques triviales.

– D'autres problèmes erratiques peuvent survenir, par exemple lors d'une tentative d'accès à une zone très particulière de la mémoire appelée *Upper Memory Area*.
– Enfin cette solution ne fonctionne pas « par défaut » sur un système Windows, qui configure le contrôleur FireWire de manière plus restrictive qu'un Linux ou un Mac OS.

Adam Boileau [23] a montré lors de la conférence Ruxcon 2006 que cette limite pouvait être levée en émulant un périphérique de stockage de masse, tel qu'un iPod. Mais ce *hack* peut disparaître en fonction des évolutions du système Windows.

Outre le bus FireWire, on notera que ces techniques matérielles peuvent être déclinées à l'envi. Ainsi la norme PCMCIA autorise également l'accès direct à la mémoire par le périphérique. Il ne reste plus qu'à se procurer une carte PCMCIA basée sur des composants reprogrammables (FPGA), telle que les cartes sponsorisées par David Hulton [26].

2.3 Les outils « dd » et « nc »

Les outils « dd » et « nc », disponibles sur le site de G. M. Garner [4], sont des versions modifiées des outils éponymes bien connus. Parmi les adaptations au monde Windows qui ont été apportées, la plus intéressante est la possibilité d'ouvrir le périphérique spécial « \Device\PhysicalMemory » [9].

Cette solution n'a pas de prérequis (ex. installation de *driver*) et offre l'avantage d'une empreinte mémoire très légère. Elle est facilement utilisable sur n'importe quelle machine, même par un opérateur sans expérience dans l'acquisition de données (ce qui est souvent le cas de l'administrateur présent sur le site).

Par exemple, il est possible d'envoyer la mémoire physique vers un centre de collecte distant à travers le réseau *via* la ligne de commande :

```
nc -v -n -I\\.\PhysicalMemory <ip> <port>
```

Tout n'est cependant pas si rose, car à bien y réfléchir cette solution présente des inconvénients :
– Le temps d'acquisition peut s'avérer assez long (de l'ordre de plusieurs heures).
– Il est possible pour un attaquant de masquer des pages (en interceptant l'appel système par exemple), ou de faire simplement échouer l'accès (voire le chapitre « contre-mesures »).
– L'accès au périphérique « \Device\PhysicalMemory » depuis l'espace utilisateur (*Ring 3*) a été bloqué par Microsoft à partir de Windows 2003 SP1, et ne sera *a priori* pas rétabli dans les futures versions de Windows.

Une mise à jour de l'outil « dd » a été promise par son auteur, mais n'est pas disponible à l'heure où j'écris ces lignes. Il faut dire que le *challenge* n'est pas trivial...

2.4 Le « CrashDump » (par clavier)

Une autre solution qui pourrait paraître surprenante de prime abord est ... de faire *crasher* le système ! En effet, il est possible dans ce cas de forcer le système à écrire un fichier journal (appelé « *crash dump* » qui contient l'intégralité de la mémoire physique, ainsi que des informations essentielles comme l'adresse de la table de pages au moment du *crash* (ceci est configurable, comme nous le verrons plus loin).

Le fichier de sortie, d'extension « .DMP », se trouve être dans un format propriétaire Microsoft compréhensible uniquement par les outils de débogage Microsoft. Ce format a toutefois été partiellement documenté par des tiers [10].

Comme on peut s'y attendre, provoquer un *crash* est une opération relativement simple ;) Un raccourci a même été intégré au système par Microsoft pour pouvoir provoquer des *crashes* à volonté, il s'agit de la clé de base de registre suivante [11] :

`HKLM\SYSTEM\CurrentControlSet\Services\i8042prt\Parameters\CrashOnCtrlScroll`

Lorsque cette clé de type `REG_DWORD` vaut 1, la séquence « Ctrl droit + ScrollLock » répétée 2 fois provoque un *crash*.

Bien que la solution du CrashDump reste la plus simple et la plus rapide, elle n'est pas sans inconvénients :

- Tous les systèmes ne peuvent pas forcément être arrêtés de cette manière, même s'ils sont compromis. De plus, le *crash* peut laisser le système dans un état instable (c'est souvent le cas des bases de données, par exemple).
- En théorie, cette solution n'est utilisable que sur un système préalablement configuré pour générer un CrashDump, car un *reboot* est requis après modification de clé. La possibilité d'activer « dynamiquement » (sans *reboot*) cette fonction sera étudiée plus loin.
- Windows utilise le fichier d'échange (par défaut « c :\pagefile.sys ») pour enregistrer le CrashDump. Ceci implique que le contenu initial de ce fichier sera perdu, et que la taille de ce fichier au moment du *crash* doit être au moins égale à la mémoire physique disponible, plus 1 Mo (pour les informations d'état).
- Compte-tenu des limitations du fichier d'échange, la taille maximale du fichier « .DMP » est de 2 Go. Il est donc impossible d'obtenir un *dump* complet sur un système possédant plus de 2 Go de mémoire physique.

Ce dernier point doit cependant être nuancé sur les versions récentes de Windows. Comme l'indique l'article de base de connaissance [13], en utilisant le paramètre non documenté « /MAXMEM » dans le fichier « BOOT.INI », il est possible de générer un dump de taille supérieur à 2 Go. Ce paramètre doit avoir été activé au boot.

Pour les systèmes disposant de plus de 4 Go de mémoire physique, il est également nécessaire d'activer « /PAE ».

On notera que Windows supporte nativement jusqu'à 16 fichiers d'échange de 4 Go chacun, soit 64 Go au total. A ma connaissance, le cas des fichiers d'échange multiples n'est toutefois pas supporté par les outils d'analyse disponibles aujourd'hui.

2.5 Le « CrashDump » (par EMS)

Windows 2003 Server possède une fonctionnalité intéressante pour générer un CrashDump, il s'agit de la console EMS (*Emergency Management Services*). Si cette fonction a été activée au boot (paramètre « /REDIRECT » dans le fichier « BOOT.INI »), une console est disponible sur le port COM1 du système. Ce port peut ensuite facilement être déporté à l'aide d'un boitier si nécessaire.

Cette console donne accès à différentes commandes, dont la commande « crashdump ». Aucune configuration préalable n'est nécessaire - le CrashDump est généré selon la configuration active. Le code d'erreur est STOP `0x000000E2` comme avec la méthode précédente.

2.6 Le « snapshot »

Un cas particulier notable est celui du service hébergé dans un logiciel de virtualisation de type « VMWare ».

Dans ce cas, l'obtention d'une image mémoire est immédiate : il suffit de mettre la machine virtuelle en pause et de récupérer le fichier « .vmem ». Il est également possible de monter le disque virtuel en lecture seule pour récupérer le fichier « pagefile.sys ».

Les informations ainsi collectées sont « parfaites », dans le sens où l'activité du système est interrompue pendant la collecte, et aucune empreinte n'est laissée par l'outil de collecte. Cette méthode a par exemple été utilisée pour le Challenge Securitech 2005.

Bien évidemment, cette méthode a un pré-requis fort : le système compromis doit être une machine virtuelle. Ce cas a longtemps été l'apanage des *honeypots*, mais se rencontre aujourd'hui sur des systèmes de production compte-tenu de l'engouement pour la virtualisation.

2.7 Le problème du fichier d'échange (swapfile)

La collecte de la mémoire physique est une chose, la collecte du fichier d'échange (*swapfile*) en est une autre.

Le fichier d'échange par défaut s'appelle « c :\pagefile.sys ». Comme vu précédemment, il est possible de déplacer le fichier d'échange sur un autre disque ou d'en créer plusieurs, mais ce cas est rare sur le terrain. Les choses pourraient toutefois changer avec Vista, si la technologie ReadyBoost (fichier d'échange en miroir sur une clé USB) s'avère massivement utilisée.

Dans le cas où un CrashDump est généré, l'espace disque alloué au fichier d'échange est réutilisé pour générer le fichier « MEMORY.DMP ». Le fichier d'échange original est donc définitivement perdu.

Pour la petite histoire, l'article de base de connaissance [14] détaille le fonctionnement de ce mécanisme. Le processus SMSS.EXE est en charge de la gestion du fichier d'échange. Lorsqu'au reboot il détecte qu'un CrashDump a été généré en lieu et place du fichier d'échange, il se contente de signaler ce fait à WINLOGON.EXE *via* la clé de base de registre :

```
HKLM\SYSTEM\CurrentControlSet\Control\CrashControl\MachineCrash
```

WinLogon appelle alors l'utilitaire SAVEDUMP.EXE qui se charge de renommer « c :\pagefile.sys » en « MEMORY.DMP ».

Lorsque Windows est actif, les fichiers d'échange sont verrouillés par le noyau. Il serait possible d'y accéder *via* un *driver* conçu pour l'occasion, ou le périphérique spécial « \Device\PhysicalDrive » (c'était la méthode utilisée par Joanna Rutkowska pour injecter des *drivers* non signés dans Windows Vista64 RC1).

Parmi les outils (payants) réputés pouvoir réaliser cette tâche, on peut citer :
- Disk Explorer [33],
- Forensic Toolkit [34],
- X-Ways Forensics [35],
- iLook (gratuit pour les agents du gouvernement US) [36].

Ces outils doivent être apportés sur le système analysé, sauf s'ils y étaient préalablement installés. La méthode la plus simple et la moins chère pour collecter le fichier d'échange reste donc ... de couper le courant !

Après avoir collecté la mémoire physique, il suffit d'éteindre brutalement la machine et de récupérer le fichier d'échange sur disque. Bien entendu, compte-tenu de l'activité du système, la cohérence de l'ensemble « mémoire physique + fichier d'échange » n'est pas garantie... mais en pratique, sauf sur un système manquant cruellement de mémoire, il s'avère que les données obtenues sont souvent exploitables.

La méthode peut sembler violente, mais il faut rappeler que l'extinction brutale d'une machine compromise est la méthode la plus couramment rencontrée dans la pratique actuelle de l'autopsie !

Afin de fusionner intelligemment les données collectées, il faut s'intéresser de plus près au format du fichier d'échange.

Sans rentrer dans les détails de l'architecture dite « Intel x86 », il faut savoir que la mémoire est découpée en pages physiques de taille 4 Ko (ou 4 Mo dans certains cas). Le catalogue des pages visibles par un processus est stocké dans un arbre à 2 niveaux, pointé par le registre spécial CR3. Le catalogue de premier niveau est appelé *Page Directory* et le catalogue de second niveau *Page Table*.

Lorsqu'une page mémoire a été déplacée dans le fichier d'échange, son descripteur (*Page Table Entry*, PTE) est marqué comme « invalide » (bit « Valid » égal à 0). Dans un PTE, le champ « PFN »

Bits	31..12	11	10	9..5	4..1	0
Contenu	Offset	Transition	Prototype	Protection	PFN	Valid

Tab. 1: Format des PTE

(*Page File Number*) représente le numéro du fichier d'échange dans lequel est stockée la page, et le champ « Offset » le déplacement à l'intérieur de ce fichier. Ce déplacement étant sur 20 bits et les pages de taille 4 Ko par défaut, on retrouve la limite des 4 Go par fichier d'échange (Source : « *Windows Internals, 4th Edition* », page 440).

La reconstruction de la mémoire incluant les pages *swappées* ne pose donc pas de problème technique particulier, bien qu'il n'existe pas à ma connaissance d'outil public permettant de réaliser cette opération. L'outil [31] FATKit annonce toutefois en être capable.

2.8 Le fichier d'hibernation

L'hibernation est un mécanisme permettant de stocker l'intégralité de l'état du système sur disque, afin de rester en veille sans consommer d'énergie.

Pour ce faire, un fichier (« c:\hiberfil.sys ») est créé sur le disque lors de l'activation de la fonction (« Option d'alimentation / Mise en veille prolongée »).

Il n'est pas recommandé d'activer cette option *après* l'incident, car un espace disque au moins égal à la mémoire physique du système sera pré-alloué, écrasant potentiellement des données dans l'espace libre du disque.

Toutefois si l'hibernation a été préalablement activée sur une machine, cette méthode de collecte de la mémoire physique peut sembler intéressante. Il reste plusieurs problèmes techniques à résoudre :

- Le fichier d'hibernation est dans un format non documenté, intégrant un algorithme de compression « propriétaire » Microsoft (heureusement accessible *via* des fonctions exportées).
- Le fichier d'hibernation ne stocke que les pages mémoire utilisées, faisant perdre potentiellement de l'information sur les pages récemment libérées.

Cette méthode pourrait être utilisée pour détecter des *rootkits* en mémoire. Son intérêt pour le *forensics* reste à démontrer par des travaux complémentaires.

3 Plus loin avec le « CrashDump »

3.1 Générer un crash

La clé « CrashOnCtrlScroll » n'est pas toujours utilisable sur le terrain, pour 2 raisons :

1. Elle nécessite un reboot pour être prise en compte.
2. Elle est associée au pilote du *chipset* Intel 8042 – et ne fonctionne donc que sur les claviers contrôlés par ce *chipset* (ce qui exclut les claviers USB par exemple).

Le premier problème n'admet pas de solution simple. En effet la clé « CrashOnCtrlScroll » est lue dans la routine `I8xKeyboardServiceParameters()` qui n'est appelée que par `I8xKeyboardStartDevice()`, elle-même appelée par `I8xPnP()` en réponse à la commande majeure `IRP_MJ_PNP (0x1B)`, mineure `IRP_MN_START_DEVICE (0x00)`. Les clés de configuration du service sont copiées dans une zone allouée dynamiquement à l'initialisation (fonction `ExAllocatePoolWithTag()`).

Contrairement à ce que pourrait laisser croire le mot « Plug-and-Play » (PnP), débrancher puis rebrancher le clavier ne provoque pas le rechargement du pilote : l'évènement n'est tout simplement pas détecté par le système !

Plusieurs autres idées viennent à l'esprit pour contrer ce problème :

1. **Envoyer un IRP bien formaté au service** `i8042prt`. Cette solution n'est pas stable : le pilote « i8042prt.sys » effectue de nombreuses opérations au moment de son initialisation ; le *crash* est assuré sur l'envoi direct d'une commande `IRP_MN_START_DEVICE` ou même `IRP_MN_STOP_DEVICE`. D'ailleurs la documentation Microsoft mentionne explicitement que les codes `IRP_MJ_PNP` ne doivent jamais être utilisés – ils sont réservés au système.
2. **Désinstaller / réinstaller le service**. Un simple « net stop i8042prt » ou équivalent ne fonctionne pas, car le service ne peut pas être arrêté, mis en pause, ou redémarré.
   La solution consistant à désinstaller/réinstaller le service n'a pas été explorée. Les fonctions du service « Plug-and-Play Manager » de Windows qui viennent à l'esprit pour réaliser cette tâche seraient `SetupDiRemoveDevice()` et `SetupDiInstallDevice()`.
   Mais quelle que soit la méthode envisagée, il est probable que le problème persiste : le service ne pouvant être arrêté, il sera marqué pour suppression au prochain *reboot*.
3. **Rechercher la configuration dans les zones mémoire allouées par le service.** Cette solution, d'un intérêt purement intellectuel, consiste à rechercher la variable de configuration susmentionnée dans les zones de mémoire allouées par le service `i8042prt`.
   Pour ceux qui l'ignorent, l'allocation dynamique de mémoire dans le noyau peut se faire dans un *pool* nommé, dont le nom sur 4 caractères est au choix de l'application (fonction `ExAllocatePoolWithTag`). Cette fonction permet d'associer un bloc mémoire à un *driver*, ce qui facilite grandement le débogage. Les noms potentiellement intéressants ici sont « 8042 » (`i8042prt`) et « Devi » (*Device Manager*). Cette solution n'a pas été explorée plus avant car elle ne sera probablement ni stable (problème des faux positifs) ni portable (*offset* dans la structure variant en fonction des versions de Windows). Elle n'est toutefois pas irréaliste, ainsi avec le débogueur LiveKD il est possible d'effectuer cette modification localement « à la main ».

Aucune solution simple n'a donc été trouvée au problème 1. Cette étude a toutefois l'intérêt de démontrer pourquoi certains paramètres de configuration Windows nécessitent un *reboot* pour être pris en compte ... Afin de corriger ce problème, le service `i8042prt` devrait demander à être notifié en cas de modification de la clé de base de registre concernée. Cela représenterait une mise à jour conséquente du code source.

Le problème 2 (clavier USB non supporté) est également rédhibitoire dans les salles d'hébergement où un clavier fixe n'est pas toujours attaché à chaque serveur.

Au final il s'avère beaucoup plus simple de *crasher* le système que de configurer le pilote du clavier ! La méthode utilisée pour provoquer un *crash* est triviale, comme le montre l'extrait de code suivant :

```
xor ecx, ecx
push ecx ; BugCheckParameter4
push ecx ; BugCheckParameter3
push ecx ; BugCheckParameter2
push ecx ; BugCheckParameter1
push MANUALLY_INITIATED_CRASH ; BugCheckCode
mov [eax], ecx
call ds:__imp__KeBugCheckEx@20 ; KeBugCheckEx(x,x,x,x,x)
```

`MANUALLY_INITIATED_CRASH` étant une constante définie à `0xE2`.

Sysinternals fournissait un outil appelé « NotMyFault » permettant de générer différents types de *crash*. Pour une raison inconnue, cet outil n'a pas été republié par Microsoft après le rachat de Sysinternals... Heureusement il reste l'outil SystemDump [30] de Dmitry Vostokov.

3.2 Configurer le CrashDump

Le type de CrashDump généré est configurable dans l'interface graphique, rubrique Panneau de Configuration / Système / Avancé / Démarrage et Récupération / Paramètres, ce qui revient à configurer des valeurs de base de registre sous la clé :

```
HKLM\System\CurrentControlSet\Control\CrashControl
```

Les valeurs qui nous intéressent sont :

```
CrashDumpEnabled (REG_DWORD)
DumpFile (REG_EXPAND_SZ)
Overwrite (REG_DWORD)
```

Ces valeurs sont documentées par Microsoft dans l'article de base de connaissance [12]. *CrashDumpEnabled* doit avoir la valeur « 1 » pour générer un *dump* complet.

Les modifications effectuées *via* le panneau de configuration sont prises en compte automatiquement, sans avoir à *rebooter* la machine. Comme il est toujours plus agréable de comprendre, voici ce qui se passe sous le capot.

L'icône « système » du panneau de configuration est gérée par l'exécutable « sysdm.cpl ». Lorsque l'utilisateur ferme le panneau de configuration, la fonction interne `CoreDumpHandleOk()` est appelée. Cette fonction sait si des changements de configuration ont été apportés, *via* la variable globale `gfCoreDumpChanged`. Dans ce cas, plusieurs vérifications de cohérence sont effectuées *via* les fonctions internes `CoreDumpValidFile(),GetMemoryConfiguration()` et `CoreDumpGetRequiredFileSize()`.

Si tout se passe bien, et c'est là le point crucial, l'appel système `NtSetSystemInformation()` est invoqué avec un paramètre `SystemInformationClass` égal à 34. C'est cet appel qui force le noyau à prendre en compte les nouveaux paramètres positionnés en base de registre.

Ceci se confirme en analysant la chaine des appels côté noyau, depuis `NtSetSystemInformation()` jusqu'à la lecture des clés de base de registre, à savoir :

- `NtSetSystemInformation()`
- `IoConfigureCrashDump()`
- `IoInitializeCrashDump()`
- `IopInitializeDCB()`
- `IopReadDumpRegistry()`

Muni de ces informations, il serait très simple d'écrire un outil en ligne de commande permettant de configurer le CrashDump sur un système.

3.3 Configurer le fichier d'échange

Le fichier d'échange est configurable dans l'interface graphique, rubrique « Panneau de Configuration / Système / Avancé / Performances / Paramètres / Avancé », ce qui revient à configurer des valeurs de base de registre sous la clé :

```
HKLM\System\CurrentControlSet\Control\Session Manager\Memory Management
```

La valeur qui nous intéresse est :

```
PagingFiles (REG_MULTI_SZ)
```

Cette valeur contient la liste des fichiers d'échange, sous la forme :

```
<nom complet>  <taille min> <taille max>
<nom complet>  <taille min> <taille max>
...
```

Sous le capot, c'est l'API `NtCreatePagingFile()` qui est appelée.

Comme le suggère la clé de base de registre, c'est le processus SMSS.EXE (*Session Manager SubSystem*) qui est en charge de l'initialisation des fichiers d'échange. Lors d'un changement de configuration, le panneau de configuration effectue des appels à `NtCreatePagingFile()` *via* la fonction interne `VirtualMemCreatePagefileFromIndex()`. Le privilège `SeCreatePagefilePrivilege` est requis pour cette opération.

Comme expliqué ci-après, il est possible de lister les fichiers de pagination actifs, ainsi que leur taux d'utilisation, par script.

3.4 Utiliser WMI

La plupart des opérations précédentes peuvent être réalisées *via* les langages de script intégrés à Windows. L'expérience montre que les administrateurs Windows négligent souvent la puissance des scripts, qui rendent pourtant de fiers services.

Par exemple la configuration du CrashDump peut être lue et écrite *via* la classe :

- *Win32_OSRecoveryConfiguration*.

Ce qui donne en VBScript :

```
strComputer = "."

Set objWMIService = GetObject("winmgmts:\\" &
                    strComputer & "\root\CIMV2")
Set colItems = objWMIService.ExecQuery( _
```

```
"SELECT * FROM Win32_OSRecoveryConfiguration",,48)
For Each objItem in colItems
Wscript.Echo "-----------------------------------"
Wscript.Echo "Win32_OSRecoveryConfiguration instance"
Wscript.Echo "-----------------------------------"
Wscript.Echo "AutoReboot: " & objItem.AutoReboot
Wscript.Echo "Caption: " & objItem.Caption
Wscript.Echo "DebugFilePath: " & objItem.DebugFilePath
Wscript.Echo "DebugInfoType: " & objItem.DebugInfoType
Wscript.Echo "Description: " & objItem.Description
Wscript.Echo "ExpandedDebugFilePath: " &
              objItem.ExpandedDebugFilePath
Wscript.Echo "ExpandedMiniDumpDirectory: " &
              objItem.ExpandedMiniDumpDirectory
Wscript.Echo "KernelDumpOnly: " & objItem.KernelDumpOnly
Wscript.Echo "MiniDumpDirectory: " &
              objItem.MiniDumpDirectory
Wscript.Echo "Name: " & objItem.Name
Wscript.Echo "OverwriteExistingDebugFile: " &
              objItem.OverwriteExistingDebugFile
Wscript.Echo "SendAdminAlert: " & objItem.SendAdminAlert
Wscript.Echo "SettingID: " & objItem.SettingID
Wscript.Echo "WriteDebugInfo: " & objItem.WriteDebugInfo
Wscript.Echo "WriteToSystemLog: " & objItem.WriteToSystemLog
Next
```

Et au final sur un Windows XP SP2 :

```
C:\> cscript CrashDump.vbs
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. Tous droits reserves.
-----------------------------------
Win32_OSRecoveryConfiguration instance
-----------------------------------
AutoReboot: Vrai
Caption:
DebugFilePath: %SystemRoot%\MEMORY.DMP
DebugInfoType: 3
Description:
ExpandedDebugFilePath: C:\WINDOWS\MEMORY.DMP
ExpandedMiniDumpDirectory: C:\WINDOWS\Minidump
KernelDumpOnly: Faux
MiniDumpDirectory: %SystemRoot%\Minidump
Name: Microsoft Windows XP Professional|C:\WINDOWS|\Device\Harddisk0\Partition1
OverwriteExistingDebugFile: Vrai
SendAdminAlert: Faux
SettingID:
```

```
WriteDebugInfo: Vrai
WriteToSystemLog: Vrai
```

La même opération est réalisable *via* la ligne de commande WMIC. Pour une raison qui m'échappe, les alias WMIC ne sont pas cohérents avec les noms des instances WMI. Ainsi la commande devient :

```
C:\> wmic
wmic:root\cli>recoveros list full
AutoReboot=TRUE
DebugFilePath=%SystemRoot%\MEMORY.DMP
Description=
KernelDumpOnly=FALSE
Name=Microsoft Windows XP Professional|C:\WINDOWS|\Device\Harddisk0\Partition1
OverwriteExistingDebugFile=TRUE
SendAdminAlert=FALSE
SettingID=
WriteDebugInfo=TRUE
WriteToSystemLog=TRUE
```

De même tous les éléments concernant le fichier d'échange peuvent être accédés *via* les classes :

- `Win32_PageFile`
- `Win32_PageFileElementSetting`
- `Win32_PageFileSetting`
- `Win32_PageFileUsage`

Ce qui donne :

```
wmic:root\cli>pagefile list full
AllocatedBaseSize=2046
CurrentUsage=8
Description=C:\pagefile.sys
InstallDate=20061129104253.031250+060
Name=C:\pagefile.sys
PeakUsage=8
Status=
TempPageFile=
wmic:root\cli>pagefileset list full
Description='pagefile.sys' @ C:\
InitialSize=2046
MaximumSize=4092
Name=C:\pagefile.sys
SettingID=pagefile.sys @ C:
```

Et pour `Win32_PageFileUsage` (qui n'est pas accessible par WMIC mais par VBScript) :

```
AllocatedBaseSize: 512
Caption: C:\pagefile.sys
CurrentUsage: 247
Description: C:\pagefile.sys
InstallDate: 31/08/2006 13:49:34
```

```
Name: C:\pagefile.sys
PeakUsage: 319
Status:
TempPageFile:
```

Il n'existe pas *a priori* de classe WMI permettant d'accéder à la configuration du fichier d'hibernation. Il faut donc accéder par script aux valeurs de base de registre concernées, à savoir :

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SessionManager\Power\*
```

Comme les valeurs concernées sont des *blobs* binaires, il pourrait sembler plus judicieux d'utiliser la commande `POWERCFG`. Malheureusement cette commande autorise les actions « POWERCFG /HIBERNATE ON » et « POWERCFG /HIBERNATE OFF », mais elle ne permet pas de connaitre l'état courant. Cet état se trouve en fait dans le 7ème octet de la valeur de base de registre « Heuristics ».

4 Outils d'analyse

4.1 Reconstruction de la mémoire virtuelle

Nous venons de voir que la collecte de la mémoire physique n'est pas une chose simple. Mais l'analyse des données obtenues à la recherche de preuves (ou tout au moins d'indices) n'est pas moins ardue !

L'analyse d'un « dump » de mémoire physique pose plusieurs problèmes. Le premier est de retrouver les structures intéressantes, telles que la liste des processus. Le deuxième est d'analyser ces structures, non documentées par Microsoft. Voyons comment ces deux problèmes peuvent être abordés.

Pour ceux qui n'ont pas lu les manuels de référence du processeur Intel [7], je rappelle que la mémoire physique est un gruyère de pages de 4 Ko. Reconstituer la vision « virtuelle » de la mémoire qu'ont les applications est donc un puzzle de 262 144 pièces sur un système avec 1 Go de mémoire. Or reconstruire cette vision est l'étape préliminaire indispensable pour travailler comme sur le système *live*.

L'approche proposée par Andreas Schuster [1] avec son outil PTFinder [28] consiste à rechercher dans chaque page physique une structure pouvant ressembler à une structure critique du système (EPROCESS). Les propriétés particulières des membres de cette structure permettent d'éliminer une bonne partie des faux positifs :

- Pointeurs dans la mémoire du noyau donc supérieurs à `0x80000000`.
- Heure et date dans un intervalle donné.
- Etc.

La structure EPROCESS possède une sauvegarde du registre CR3, donc un pointeur vers la page physique qui contient la table de pages du processus. Cet élément est la clé de la reconstruction.

Cette approche a d'autres avantages :

- Elle permet de retrouver les processus terminés, lorsque la mémoire n'a pas été réallouée.
- Elle permet également de retrouver les processus masqués par des techniques de type DKOM (*Direct Kernel Object Manipulation*), si les objets ont simplement été déliés mais pas effacés.

Cette approche n'est toutefois pas parfaite, elle présente les inconvénients suivants :

– Elle n'est pas fiable à 100% - ce qui peut poser problème pour une expertise judiciaire. Mais la conclusion de cet article semble être qu'aucune méthode d'analyse n'est fiable de manière prouvable...
– Elle nécessite la connaissance des structures internes du noyau, qui changent avec chaque *Service Pack* de Windows, et ne sont pas *officiellement* documentées. Cependant les outils de débogage Microsoft [8] intègrent une description détaillée de toutes ces structures, accessible par la commande « dt ».

```
kd>.reload
Connected to Windows 2000 2195 x86 compatible target, ptr64 FALSE
Loading Kernel Symbols
.....................................................................
Loading User Symbols
Loading unloaded module list
.................
kd> dt _EPROCESS
+0x000 Pcb : _KPROCESS
+0x06c ExitStatus : Int4B
+0x070 LockEvent : _KEVENT
+0x080 LockCount : Uint4B
+0x088 CreateTime : _LARGE_INTEGER
+0x090 ExitTime : _LARGE_INTEGER
+0x098 LockOwner : Ptr32 _KTHREAD
+0x09c UniqueProcessId : Ptr32 Void
+0x0a0 ActiveProcessLinks : _LIST_ENTRY
+0x0a8 QuotaPeakPoolUsage : [2] Uint4B
+0x0b0 QuotaPoolUsage : [2] Uint4B
+0x0b8 PagefileUsage : Uint4B
+0x0bc CommitCharge : Uint4B
+0x0c0 PeakPagefileUsage : Uint4B
+0x0c4 PeakVirtualSize : Uint4B
+0x0c8 VirtualSize : Uint4B
+0x0d0 Vm : _MMSUPPORT
[...]
```

D'autres outils que PTFinder [28] existent, tels que Volatools [29]. Néanmoins tous les outils publics qu'il m'a été donné d'analyser semblent fonctionner sur le même principe.

5 Exemples d'audits post-instrusion

5.1 Détection du Meterpreter

Le Meterpreter est un outil d'intrusion « tout en mémoire » distribué comme *payload* depuis la version 2.2 de Metasploit (publiée en août 2004). Il a été développé par Jarkko Turkulainen et Matt Miller, sur la base d'une recherche originale [19].

Le principe est d'injecter une librairie (une DLL sous Windows) dans la mémoire du processus exploité, sans l'écrire sur le disque. Pour se faire, les *hooks* suivants sont installés en mémoire dans le processus cible (toutes ces fonctions étant exportées par NTDLL.DLL) :

- `NtOpenSection()`
- `NtQueryAttributesFile()`
- `NtOpenFile()`
- `NtCreateSection()`
- `NtMapViewOfSection()`

A l'aide de ces 5 *hooks*, il est possible d'abuser la fonction `LoadLibrary()` pour lui faire charger toutes les données utiles depuis une zone mémoire plutôt qu'un fichier (l'ensemble des opérations étant réalisées en *userland*). Simple, mais plutôt efficace !

Pour ceux qui souhaitent en savoir plus, les sources sont disponibles dans « external/source/meterpreter ». La lisibilité du code et la qualité des commentaires sont exceptionnelles.

Prenons le cas d'un Windows 2000 SP4 (version anglaise), compromis par Metasploit + Meterpreter. Après avoir généré un CrashDump, nous allons utiliser la méthode « manuelle », à savoir les Microsoft Debugging Tools.

La première étape est de charger le CrashDump dans le débogueur et d'énumérer les processus.

```
kd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS 81841380 SessionId: 0 Cid: 0008 Peb: 00000000 ParentCid: 0000
        DirBase: 00030000 ObjectTable: 81841e68 TableSize: 155.
        Image: System
PROCESS 816a86c0 SessionId: 0 Cid: 00a0 Peb: 7ffdf000 ParentCid: 0008
        DirBase: 0291f000 ObjectTable: 816a1c68 TableSize: 33.
        Image: SMSS.EXE
PROCESS 8168a140 SessionId: 0 Cid: 00b8 Peb: 7ffdf000 ParentCid: 00a0
        DirBase: 037fe000 ObjectTable: 8168a488 TableSize: 335.
        Image: CSRSS.EXE
PROCESS 81683820 SessionId: 0 Cid: 00b4 Peb: 7ffdf000 ParentCid: 00a0
        DirBase: 03bc3000 ObjectTable: 81683c88 TableSize: 374.
        Image: WINLOGON.EXE
PROCESS 81671020 SessionId: 0 Cid: 00e8 Peb: 7ffdf000 ParentCid: 00b4
        DirBase: 03f63000 ObjectTable: 816725e8 TableSize: 505.
        Image: SERVICES.EXE
PROCESS 816705c0 SessionId: 0 Cid: 00f4 Peb: 7ffdf000 ParentCid: 00b4
        DirBase: 03f2b000 ObjectTable: 81670b28 TableSize: 271.
        Image: LSASS.EXE
PROCESS 8164ab40 SessionId: 0 Cid: 01b4 Peb: 7ffdf000 ParentCid: 00e8
        DirBase: 04a6b000 ObjectTable: 8164ae28 TableSize: 262.
        Image: svchost.exe
 [...]
```

Pour pouvoir analyser un processus, il faut se placer dans son contexte du point de vue du débogueur. Nous allons nous placer ici dans le contexte du processus SVCHOST.EXE (pid 0x1b4 = 436d, structure EPROCESS à l'adresse virtuelle 0x8164ab40).

```
kd> .process 8164ab40
Implicit process is now 8164ab40
```

Dès lors, il est possible de parcourir la liste des DLLs chargées par ce processus.

Version courte :

```
kd> !peb
PEB at 7FFDF000
InheritedAddressSpace: No
ReadImageFileExecOptions: No
BeingDebugged: No
ImageBaseAddress: 01000000
Ldr.Initialized: Yes
Ldr.InInitializationOrderModuleList: 71f40 . 8fa28
Ldr.InLoadOrderModuleList: 71ec0 . 8fa18
Ldr.InMemoryOrderModuleList: 71ec8 . 8fa20
    Base TimeStamp Module
    1000000 3814ad86 Oct 25 21:20:38 1999 C:\WINNT\system32\svchost.exe
    77f80000 3ef274dc Jun 20 04:43:40 2003 C:\WINNT\system32\ntdll.dll
  [...]
    775a0000 3ef274dc Jun 20 04:43:40 2003 C:\WINNT\system32\CLBCATQ.DLL
    10000000 439e49c1 Dec 13 05:10:41 2005 C:\WINNT\system32\metsrv.dll <--
    c30000 4435ac71 Apr 07 02:04:01 2006 C:\WINNT\system32\ext635732.dll <--

SubSystemData: 0
ProcessHeap: 70000
ProcessParameters: 20000
WindowTitle: 'C:\WINNT\system32\svchost.exe'
ImageFile: 'C:\WINNT\system32\svchost.exe'

CommandLine: 'C:\WINNT\system32\svchost -k rpcss'

DllPath: 'C:\WINNT\system32;.;C:\WINNT\system32;
C:\WINNT\system;C:\WINNT;C:\WINNT\system32;C:\WINNT;C:\WINNT\
System32\Wbem'
Environment: 0x10000
```

Version longue :

```
kd> !dlls

0x00071ec0: C:\WINNT\system32\svchost.exe
Base 0x01000000 EntryPoint 0x010010b8 Size 0x00005000
Flags 0x00005000 LoadCount 0x0000ffff TlsIndex 0x00000000
      LDRP_LOAD_IN_PROGRESS
      LDRP_ENTRY_PROCESSED

0x00071f30: C:\WINNT\system32\ntdll.dll
Base 0x77f80000 EntryPoint 0x00000000 Size 0x0007b000
Flags 0x00004004 LoadCount 0x0000ffff TlsIndex 0x00000000
      LDRP_IMAGE_DLL
      LDRP_ENTRY_PROCESSED
```

```
[...]

0x000bfbb8: C:\WINNT\system32\CLBCATQ.DLL
Base 0x775a0000 EntryPoint 0x7760f150 Size 0x00086000
Flags 0x000c4004 LoadCount 0x00000001 TlsIndex 0x00000000
      LDRP_IMAGE_DLL
      LDRP_ENTRY_PROCESSED
      LDRP_DONT_CALL_FOR_THREADS
      LDRP_PROCESS_ATTACH_CALLED

0x0009caf0: C:\WINNT\system32\metsrv.dll
Base 0x10000000 EntryPoint 0x10004a73 Size 0x00013000
Flags 0x002c4004 LoadCount 0x00000002 TlsIndex 0x00000000
      LDRP_IMAGE_DLL
      LDRP_ENTRY_PROCESSED
      LDRP_DONT_CALL_FOR_THREADS
      LDRP_PROCESS_ATTACH_CALLED
      LDRP_IMAGE_NOT_AT_BASE

0x0008fa18: C:\WINNT\system32\ext635732.dll
Base 0x00c30000 EntryPoint 0x00c36068 Size 0x00023000
Flags 0x00284004 LoadCount 0x00000001 TlsIndex 0x00000000
LDRP_IMAGE_DLL
LDRP_ENTRY_PROCESSED
LDRP_PROCESS_ATTACH_CALLED
LDRP_IMAGE_NOT_AT_BASE
```

Les deux dernières DLLs sont celles du Meterpreter, facilement identifiables par leur nom.

Bien entendu nous sommes allés directement au résultat ici. Dans un cas réel, il est beaucoup plus difficile de déterminer la faille utilisée et le processus fautif ; de plus les librairies Meterpreter auraient pu être renommées. Dans ces conditions on touche rapidement aux limites de WinDbg.

- WinDbg ne permet pas de parcourir les processus terminés (ce qui est souvent le cas après l'exploitation d'une faille).
- L'énumération des DLLs pour chaque processus est possible *via* un script. Néanmoins le langage de script est très limité (on aurait aimé une interface avec des langages de plus haut niveau comme Python, par exemple :). Voici un exemple trouvé sur [5] :

```
$$
$$ Script parcourant la liste des processus
$$

r $t0 = nt!PsActiveProcessHead
.for (r $t1 = poi(@$t0); (@$t1 != 0) & (@$t1 != @$t0);
  r $t1 = poi(@$t1))
  {
   .catch {
```

```
          r? $t2 = #CONTAINING_RECORD(@$t1, nt!_EPROCESS,
          ActiveProcessLinks);
          .process @$t2
          .reload
          !peb
         }
}
```

On admirera la parfaite illisibilité de ce script. La directive « .catch » permet d'éviter la fermeture du débogueur en cas d'erreur (!). Le résultat de ce script est le suivant :

```
kd> $$><script.txt
Implicit process is now 81841380
Loading Kernel Symbols
...............................................................................
Loading User Symbols

Loading unloaded module list
.................
PEB at 00000000
    *** unable to read PEB
Implicit process is now 816a86c0

Loading Kernel Symbols
...............................................................................
Loading User Symbols
...
Loading unloaded module list
.................
PEB at 7FFDF000
    InheritedAddressSpace: No
    ReadImageFileExecOptions: No
    BeingDebugged: No
    ImageBaseAddress: 48580000
    Ldr.Initialized: Yes
    Ldr.InInitializationOrderModuleList: 161f40 . 1627a0
    Ldr.InLoadOrderModuleList: 161ec0 . 162790
    Ldr.InMemoryOrderModuleList: 161ec8 . 162798
            Base TimeStamp                     Module
        48580000 3d5cebca Aug 16 14:10:50 2002 \SystemRoot\System32\smss.exe
        77f80000 3ef274dc Jun 20 04:43:40 2003 C:\WINNT\system32\ntdll.dll
        68010000 3ef27500 Jun 20 04:44:16 2003 C:\WINNT\System32\sfcfiles.dll
    SubSystemData: 0
    ProcessHeap: 160000
    ProcessParameters: 110000
        WindowTitle: '(null)'
    ImageFile: '\SystemRoot\System32\smss.exe'
```

```
    CommandLine: '\SystemRoot\System32\smss.exe'
    DllPath: 'C:\WINNT\System32'
    Environment: 0x100000

Implicit process is now 8168a140
Loading Kernel Symbols
...............................................................................
Loading User Symbols
........
Loading unloaded module list
.................
PEB at 7FFDF000
    InheritedAddressSpace: No
    ReadImageFileExecOptions: No
    BeingDebugged: No
    ImageBaseAddress: 5FFF0000
    Ldr.Initialized: Yes
    Ldr.InInitializationOrderModuleList: 161f40 . 162fb0
    Ldr.InLoadOrderModuleList: 161ec0 . 163188
    Ldr.InMemoryOrderModuleList: 161ec8 . 163190
        Base TimeStamp                     Module
      5fff0000 3ef2750b Jun 20 04:44:27 2003 \??\C:\WINNT\system32\csrss.exe
      77f80000 3ef274dc Jun 20 04:43:40 2003 C:\WINNT\system32\ntdll.dll
      5ff90000 3ef274e9 Jun 20 04:43:53 2003 C:\WINNT\system32\CSRSRV.dll
      5ffa0000 3ef274e6 Jun 20 04:43:50 2003 C:\WINNT\system32\basesrv.dll
      5ffb0000 3ef27505 Jun 20 04:44:21 2003 C:\WINNT\system32\winsrv.dll
      77e10000 3ef274dc Jun 20 04:43:40 2003 C:\WINNT\system32\USER32.DLL
      7c4e0000 3ef274dc Jun 20 04:43:40 2003 C:\WINNT\system32\KERNEL32.DLL
      77f40000 3ef274dc Jun 20 04:43:40 2003 C:\WINNT\system32\GDI32.DLL
    SubSystemData: 0
    ProcessHeap: 160000
    ProcessParameters: 110000
       WindowTitle: '(null)'
       ImageFile: '\??\C:\WINNT\system32\csrss.exe'
       CommandLine: 'C:\WINNT\system32\csrss.exe
           ObjectDirectory=\Windows
    SharedSection=1024,3072,512 Windows=On SubSystemType=Windows
    ServerDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,3
    ServerDll=winsrv:ConServerDllInitialization,2 ProfileControl=Off
    MaxRequestThreads=16'
          DllPath:
    'C:\WINNT\system32;C:\WINNT\system32;C:\WINNT;C:\ WINNT\System32\Wbem'
          Environment: 0x100000
[...]
```

– On aurait également aimé pouvoir *dumper* chaque binaire à des fins de contrôle d'intégrité. Il est possible d'accéder à la mémoire *via* les commandes du débogueur, mais pas plus. *Dumper*

et reconstruire le binaire ne peut se faire qu'à travers un plugin, or l'écriture de plugins pour WinDbg est ardu de part l'absence de documentation et la complexité de l'API. Pour donner une idée de l'ampleur de la tâche, il faut savoir que les exemples fournis dans le SDK ne compilent pas par défaut - il est nécessaire de rajouter des *#define* particuliers dans l'entête des fichiers source...

```
              Entête de la librairie METSRV.DLL
kd> d 0x01000000

01000000 4d 5a 90 00 03 00 00 00-04 00 00 00 ff ff 00 00 MZ..............
01000010 b8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00 ........@.......
01000020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
01000030 00 00 00 00 00 00 00 00-00 00 00 00 c0 00 00 00 ................
01000040 0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68 ........!..L.!Th
01000050 69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f is program canno
01000060 74 20 62 65 20 72 75 6e-20 69 6e 20 44 4f 53 20 t be run in DOS
01000070 6d 6f 64 65 2e 0d 0d 0a-24 00 00 00 00 00 00 00 mode....$.......

             Entête de la librairie EXT635732.DLL
kd> d 0xc30000

00c30000 4d 5a 90 00 03 00 00 00-04 00 00 00 ff ff 00 00 MZ..............
00c30010 b8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00 ........@.......
00c30020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
00c30030 00 00 00 00 00 00 00 00-00 00 00 00 f8 00 00 00 ................
00c30040 0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68 ........!..L.!Th
00c30050 69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f is program canno
00c30060 74 20 62 65 20 72 75 6e-20 69 6e 20 44 4f 53 20 t be run in DOS
00c30070 6d 6f 64 65 2e 0d 0d 0a-24 00 00 00 00 00 00 00 mode....$.......
```

En conclusion, l'outil `WinDbg` est le plus « portable » car il est compatible avec toutes les versions de Windows. Mais il est également le plus limité pour une analyse *post-mortem*, comme nous allons le voir en le comparant à d'autres outils.

5.2 Challenge Securitech 2005

Le niveau 16 du Challenge Securitech 2005 [18] a été conçu par Kostya Kortchinsky. Le principe était de récupérer les éléments suivants dans une capture de mémoire physique :

1. *L'identifiant Microsoft de la vulnérabilité utilisée (par exemple MS01-123).*
2. *L'adresse IP à partir de laquelle l'attaque a été lancée.*
3. *Le mot de passe de l'administrateur du serveur.*
4. *La chaine retournée par le programme « validnivo » installé par l'attaquant.*

Le système hôte était un Windows 2000 SP4.

Afin de simplifier la tâche de l'analyste, le fichier de pagination avait été désactivé. Toute la mémoire du système, soit 256 Mo, avait été rendue disponible grâce à la fonction *snapshot* du logiciel VMWare.

Par contre, le système de fichiers n'a pas été distribué, ce qui rend la réponse à la question 4 particulièrement délicate. En effet, il faut reconstruire l'exécutable à partir du processus en mémoire (terminé) ou du tampon de réception réseau (en supposant que le programme a été *uploadé* par l'attaquant). Nous allons d'abord utiliser PTFinder [28] pour reconstruire la liste des processus, y compris les processus terminés.

```
C:\>perl ptfinder_w2k.pl --nothreads --dotfile output-no-thread.dot ..\win2000pro.vmem

No.  PID  Time created        Exited   Offset     PDB        Remarks
---- ---- ------------------- -------- ---------- --------   --------
  1  0                                 0x0046f930 0x00030000 Idle
  2  896  2005-06-22 09:00:53 09:00:53 0x01573700 0x0a8c6000 validnivo.exe
  3  536  2005-06-22 08:59:25 09:00:47 0x01574200 0x093e7000 IEXPLORE.EXE
  4  868  2005-06-22 08:57:24          0x01590cc0 0x071cf000 internat.exe
  5  848  2005-06-22 08:57:24          0x01593880 0x06d3f000 VMwareUser.exe
  6  836  2005-06-22 08:57:24          0x015954c0 0x06c73000 VMwareTray.exe
  7  752  2005-06-22 08:57:22 08:57:46 0x015a8d00 0x06086000 userinit.exe
  8  760  2005-06-22 08:57:22          0x015a9a20 0x06092000 explorer.exe
  9  956  2005-06-22 08:58:48          0x01613d60 0x07de1000 cmd.exe
 10  620  2005-06-22 08:57:03          0x01627180 0x04ac5000 svchost.exe
 11  588  2005-06-22 08:57:02          0x0162c020 0x04b3c000 VMwareService.exe
 12  528  2005-06-22 08:57:02          0x01638b00 0x047b8000 mstask.exe
 13  504  2005-06-22 08:57:01          0x0164b020 0x04575000 regsvc.exe
 14  468  2005-06-22 08:57:00          0x0164e520 0x040dc000 svchost.exe
 15  436  2005-06-22 08:57:00          0x01653d60 0x04211000 SPOOLSV.EXE
 16  408  2005-06-22 08:56:59          0x0165bce0 0x040c9000 svchost.exe
 17  224  2005-06-22 08:56:56          0x0167eb80 0x03cd8000 lsass.exe
 18  212  2005-06-22 08:56:56          0x01680020 0x03d0f000 services.exe
 19  184  2005-06-22 08:56:54          0x016f1c40 0x03a97000 winlogon.exe
 20  164  2005-06-22 08:56:53          0x016f72a0 0x037d2000 csrss.exe
 21  140  2005-06-22 08:56:50          0x016fdce0 0x029c6000 smss.exe
 22  8                                 0x0188d520 0x00030000 System
```

L'outil PTFinder [28] permet également de générer une représentation graphique des processus (basée sur leur PPID). Les blocs en gris correspondent aux processus terminés. On identifie immédiatement le processus `VALIDNIVO.EXE`, lancé par `CMD.EXE`, lui-même lancé par `LSASS.EXE` (ce qui est fortement suspect !). Le processus `VALIDNIVO.EXE` et sa table de pages sont identifiés. Malheureusement l'outil MemDump (du même auteur) ne permet pas de reconstruire le binaire dans ce cas précis, car la table de pages a été endommagée.

Par contre si nous utilisons MemDump sur le CMD.EXE parent, nous obtenons le résultat suivant :

```
C:\> perl memdump.pl win2000pro.vmem 0x07de1000
Reading page directory at file offset 0x7de1000... done.
C:\> type 0x7de1000.map

virt. addr.   file offset   size
------------ ------------ --------
   0x10000           0        0x1000
   0x20000     0x1000         0x1000
```

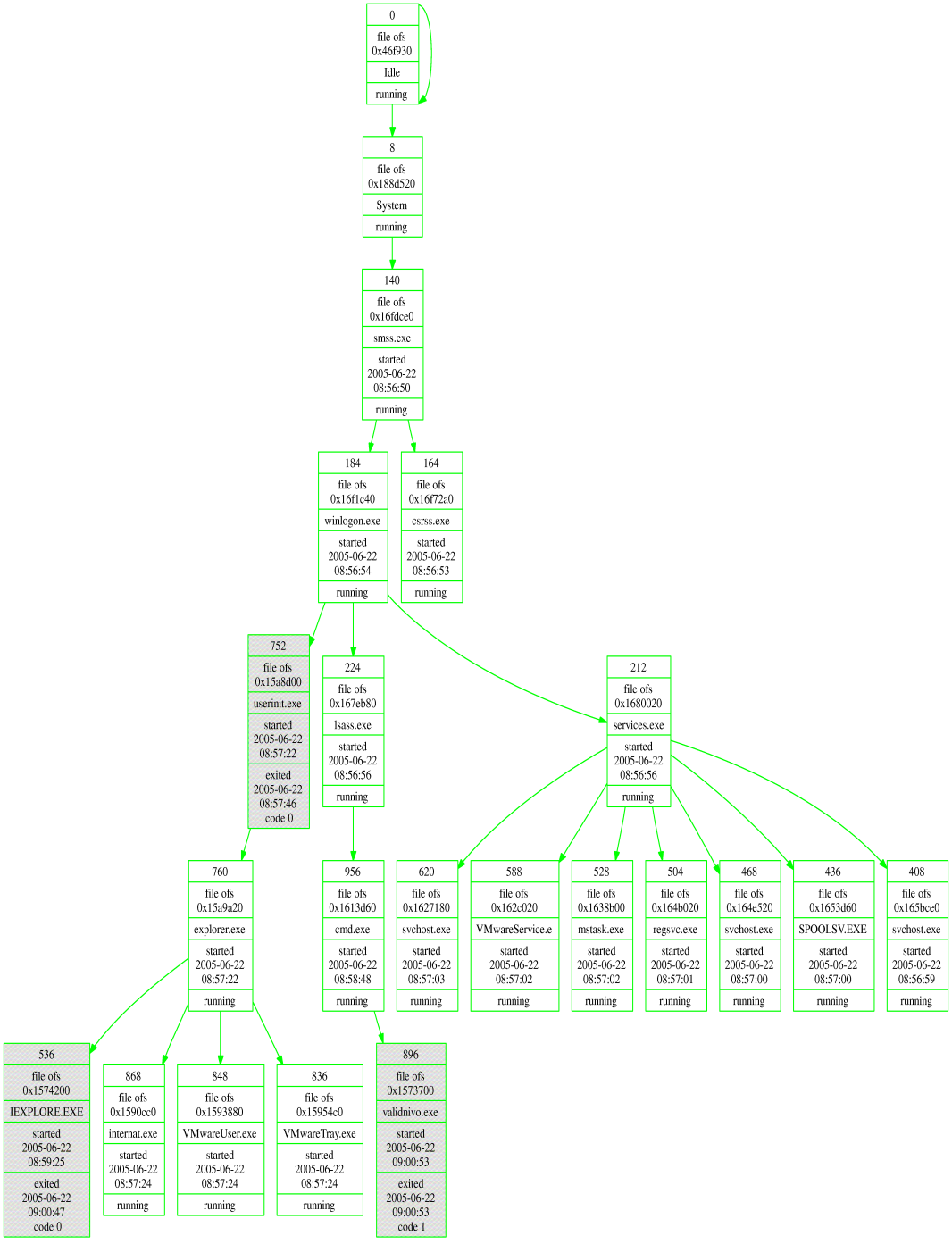

FIG. 1: L'arborescence des processus, reconstituée à partir de la mémoire

```
  0x12e000     0x2000        0x1000
  0x12f000     0x3000        0x1000
  0x130000     0x4000        0x1000
[...]
0x4ad00000    0x12000        0x1000
0x4ad01000    0x13000        0x1000
0x4ad04000    0x14000        0x1000
[...]
0x7ffde000    0x8c000        0x1000
0x7ffdf000    0x8d000        0x1000
0x7ffe0000    0x8e000        0x1000
0xc0000000    0x8f000        0x1000
0xc0001000    0x90000        0x1000
0xc012b000    0x91000        0x1000
[...]
```

Le fichier *0x7de1000.mem* contient l'ensemble des pages dans l'espace du processus, ce qui inclut le fichier exécutable, les DLLs, la mémoire allouée dynamiquement et les contextes en espace noyau. Il est alors possible de reconstruire le binaire à partir de ce *dump*, *modulo* les problèmes classiques tels que la reconstruction des imports.

Il est assez facile d'identifier que ce CMD était en fait un « Metasploit Courtesy Shell » (titre de la fenêtre créée par Metasploit après exploitation).

```
C:\> strings -o 0x7de1000.mem | grep -i metasploit
5504:Metasploit Courtesy Shell (TM)
31696:Metasploit Courtesy Shell (TM)
37368:Metasploit Courtesy Shell (TM)
50844:Metasploit Courtesy Shell (TM) - v
29288:Metasploit Courtesy Shell (TM)
```

La même information peut être obtenue avec l'outil MemParser.

```
C:\> memparser.exe win2000pro.vmem

MemParser v1.3 Chris Betz, (c) 2005
No process list loaded.
In Windows 2000 Mode
Options:

l:       Load the process list
<enter>: Quit

l
Searching for processes in memory dump

00%--05%--10%--15%--20%--25%--30%--35%--40%--45%--50%--55%--
60%--65%--70%--75%--80%--85%--90%--95%--100%
```

```
Enumerating process structures.
Sorting processes by PID
Checking for processes hidden by detachment from process
  link-list or processes no longer active
Searching for all threads.
MemParser v1.3 Chris Betz, (c) 2005
Process List:

   Proc#     PPID     PID   InProcList  Name:
 Threads:
    0         0        0      Yes       Idle
    1         0        8      Yes     System
    2         8      140      Yes   smss.exe
    3       140      164      Yes  csrss.exe
    4       140      184      Yes winlogon.exe
    5       184      212      Yes services.exe
    6       184      224      Yes lsass.exe
    7       212      408      Yes svchost.exe
    8       212      436      Yes SPOOLSV.EXE
    9       212      468      Yes svchost.exe
   10       212      504      Yes regsvc.exe
   11       212      528      Yes mstask.exe
   12       212      588      Yes VMwareService.e
   13       212      620      Yes svchost.exe
   14       752      760      Yes explorer.exe
   15       760      836      Yes VMwareTray.exe
   16       760      848      Yes VMwareUser.exe
   17       760      868      Yes internat.exe
   18       224      956      Yes cmd.exe
   19         0 29718073       No
   20         0 29718073       No
   21         0 29718073       No

In Windows 2000 Mode
Options:

#: Select a process
s: Show System Information
<enter>: Quit
18

956: cmd.exe selected:
1 Dump Process Memory (No System Memory Included) to Disk
2 Dump Process Memory (Including System Memory Space) to Disk
3 Dump Process Strings (No System Memory Included) to Disk
4 Dump Process Strings (Including System Memory Space) to Disk
```

```
  (Takes a long time)
5 Display Process Environment Information
6 Display all DLLs loaded by process

<enter>: quit
5
Process Environment Information:
        Executable File: C:\WINNT\system32\cmd.exe
        Command Line: cmd.exe
        Window Title: Metasploit Courtesy Shell (TM)
        Desktop Info: WinSta0\Default
        Shell Info:
        Runtime Data:
        Dll Path:
C:\WINNT\system32;.;C:\WINNT\system32;C:\WINNT\system;C:\WINNT;
C:\WINNT\System32
```

La reconstruction du processus terminé (`VALIDNIVO.EXE`) pose un problème qui n'a pu être surmonté qu'à la main. Pour cela nous avons utilisé le fait que le binaire se trouve en mémoire sous forme de processus et sous forme de paquets réseau (ayant été *uploadé* sur la machine). Comme la taille des blocs de données et leur alignement est différent dans ces deux zones mémoire, il est possible de retrouver les « jointures » manquantes entre chaque page.

Sur un processus de taille raisonnable (ici 10 pages de 4 Ko), le réassemblage peut être fait à la main. Mais cela démontre les limites des techniques et outils actuels ...

5.3 Challenge DFRWS 2005

Le principe est à peu près le même que celui du Challenge Securitech. Là encore le système cible était un Windows 2000. Les questions posées étaient les suivantes (en anglais dans le texte) :

1. *What hidden processes were running on the system, and how were they hidden ?*
2. *What other evidence of the intrusion can be extracted from the memory dumps ?*
3. *Why did « plist.exe » and « fport.exe » not work on the compromised system ?*
4. *Was the intruder specifically seeking Professor Goatboy's research materials ?*
5. *Did the intruder obtain the Professor's research ?*
6. *What computer was the intrusion launched from ?*
7. *Is there any indication of who the intruder might be ?*

Afin de ne pas paraphraser les solutions des auteurs, je vous invite à lire leurs papiers publiés sur le site [17]. Les techniques utilisées sont *grosso modo* les mêmes que celles présentées précédemment.

Deux outils ont été publiés à l'occasion du challenge :

- MemParser [32], publié sur SourceForge, qui combine PTFinder et MemDump dans un outil écrit en C donc beaucoup plus rapide que les scripts PERL d'Andreas Schuster.
- KnTTools [38], qui est aujourd'hui vendu par son auteur.

6 Contre-mesures

Comme on vient de le voir, l'analyse mémoire reste un sujet largement expérimental. Et pourtant certains s'intéressent déjà aux techniques anti-analyse !

Il faut dire que la détection en mémoire est déjà largement mise en œuvre par les anti-rootkits existants, il est donc normal que les auteurs de rootkits se soient penchés depuis longtemps sur la dissimulation en mémoire.

Nous avons déjà survolé les techniques « anti-analyse matérielle » connues, basées sur la reprogrammation du NorthBridge. D'autres verront très certainement le jour, vu la complexité de l'architecture PC. Des pistes de réflexion sont par exemple la dissimulation de code dans les *firmwares* ou dans la zone SMM [24].

Les techniques logicielles connues sont plus nombreuses :

- Une technique « basique » mais toujours efficace consiste à bloquer les points d'entrée pouvant servir à l'analyse, tels que « `\Device\PhysicalMemory` » ou le chargement de nouveaux *drivers*. Bien entendu il suffit d'un seul point d'entrée oublié pour pouvoir contourner la protection. Il ne s'agit donc pas à proprement parler d'une protection, même si bloquer l'utilisation de l'outil « dd » sur un système reste gênant face à un intervenant dont la mission se limite à acquérir la mémoire le plus rapidement possible selon une procédure préétablie.
- Historiquement la technique DKOM (*Direct Kernel Object Manipulation*) est la plus ancienne, elle consiste à masquer des objets (tels que des processus) en les supprimant dans les listes d'objets maintenues en interne par le noyau. Cette technique est également la plus facile à détecter : un *thread* n'appartenant à aucun processus est une chose assez peu courante sur un système sain ...
- Une autre technique est la création d'un nouveau *thread* dans un processus existant. En mode utilisateur les processus `EXPLORER.EXE` ou `IEXPLORE.EXE` sont couramment ciblés. En mode noyau le pilote NULL.SYS est une cible idéale. Néanmoins le code du *thread* injecté se trouve dans une zone mémoire allouée dynamiquement, ce qui est peu courant (sauf dans les logiciels *packés*).
- La technique la plus puissante connue actuellement est basée sur les « Split TLBs », popularisée sous Linux par l'outil de protection PaX. Cette technique consiste à désynchroniser les deux *Translation Lookaside Buffers* (TLBs) du processeur, à savoir celui des données et celui des instructions. Ainsi le contenu d'une adresse mémoire ne sera pas le même sur un accès en lecture et sur un accès en exécution ! Dans ces conditions, un simple « dd » est leurré. Bien que la page physique contenant le code apparaisse dans le *dump* final, il sera très difficile de faire le lien avec son contexte d'exécution. L'utilisation d'un *driver* prenant soin de resynchroniser les caches avant chaque accès à une page mémoire permettrait de résoudre le problème.

On peut noter également que le Meterpreter, qui est l'implémentation publique la plus couramment rencontrée de l'intrusion « tout en mémoire » sous Windows, est également la plus facile à détecter !

7 Conclusion

Malgré une activité de recherche intense sur le sujet, la collecte et l'analyse de la mémoire d'un système Windows reste une activité peu mature.

Au niveau de la collecte, aucune procédure fiable à 100% n'existe (y compris les systèmes d'acquisition matérielle) et des compromis doivent être trouvés entre la qualité de l'image obtenue, la

possibilité de *crasher* la cible, la quantité de mémoire physique, le temps disponible pour l'acquisition, etc.

Sur un système quelconque qui n'a pas été préalablement préparé, les options disponibles sont restreintes : un « dd » sur le périphérique « PhysicalMemory » (quand celui-ci est accessible) ou un CrashDump par un *driver*.

Les outils disponibles dans le domaine public restent rudimentaires et nécessitent de nombreuses adaptations à la cible (ex. adresses « en dur » selon la version de Windows). Ceci ne présume en rien de la qualité des outils payants et/ou réservés aux autorités policières, le plus mystérieux étant l'outil WOLF (Windows OnLine Forensics) de Microsoft [37].

Malgré toutes ces contraintes techniques, la généralisation des malwares « tout en mémoire » (tels que les malwares chiffrés sur disque et dont la clé de déchiffrement est récupérée sur un site Web) va contraindre les experts du domaine à s'adapter rapidement. Un simple « grep » (ou tout autre outil payant équivalent) sur la mémoire physique ne suffit pas.

La bonne nouvelle de ce point de vue est que, comme j'ai tenté de le démontrer dans ce papier, l'information obtenue en mémoire est largement exploitable ; en particulier lorsque l'intrusion a eu lieu selon une technique connue (ex. utilisation de Metasploit) - c'est-à-dire dans la majorité des cas pratiques. A défaut d'obtenir un historique complet de l'intrusion, on en obtient au moins la preuve alors qu'elle aurait disparu avec l'extinction de la machine.

A mon sens, il serait donc dommage de se priver de l'analyse de la mémoire en complément des analyses de disques « classiques ». D'autant que la qualité des outils disponibles ne peut que s'améliorer avec le temps. Ce qui ne veut pas dire que l'analyse mémoire est à la portée de n'importe qui, y compris parmi les « experts » de l'autopsie !

Références

Blogs

1. Andreas Schuster, `http://computer.forensikblog.de/en/`
2. Windows Incident Response, `http://windowsir.blogspot.com/`
3. Mariusz Burdach, `http://forensic.seccure.net/`
4. George M. Garner, `http://users.erols.com/gmgarner/forensics/`
5. Dump Analysis, `http://www.dumpanalysis.org/`
6. Alexandre Garaud, `http://c4rtman.blogspot.com/`

Sites d'information

7. Intel® 64 and IA-32 Architectures Software Developer's Manuals, `http://www.intel.com/products/processor/manuals/index.htm`
8. Debugging Tools for Windows, `http://www.microsoft.com/whdc/devtools/debugging/default.mspx`
9. PhysicalMemory, \Device\PhysicalMemory, `http://technet2.microsoft.com/WindowsServer/en/library/e0f862a3-cf16-4a48-bea5-f2004d12ce351033.mspx?mfr=true`
10. DMP File Structure, `http://computer.forensikblog.de/en/2006/03/dmp_file_structure.html`
11. Windows feature lets you generate a memory dump file by using the keyboard, `http://support.microsoft.com/kb/244139`

12. Overview of memory dump file options for Windows Server 2003, Windows XP, and Windows 2000, http://support.microsoft.com/kb/254649
13. How to overcome the 4,095 MB paging file size limit in Windows, http://support.microsoft.com/kb/237740
14. What to consider when you configure a new location for memory dump files in Windows Server 2003 http://support.microsoft.com/kb/886429
15. IOMMU, http://en.wikipedia.org/wiki/IOMMU
16. Memory dumping over FireWire - UMA issues. http://ntsecurity.nu/onmymind/2006/2006-09-02.html
17. DFRWS 2005 Challenge, http://www.dfrws.org/2005/challenge/index.html
18. Securitech 2005, Challenge 16, http://www.challenge-securitech.com/archives/2005/displaylevel.php?level=21

Présentations

19. Remote Library Injection, http://www.nologin.org/Downloads/Papers/remote-library-injection.pdf
20. « A Hardware-Based Memory Acquisition Procedure for Digital Investigations », http://www.digital-evidence.org/papers/tribble-preprint.pdf
21. iPod « Firewire - all your memory are belong to us », http://md.hudora.de/presentations/firewire/2005-firewire-cansecwest.pdf
22. Joanna Rutkowska, « Beyond The CPU : Defeating Hardware Based RAM Acquisition Tools (Part I : AMD case) », http://invisiblethings.org/papers/cheating-hardware-memory-acquisition-updated.ppt
23. Adam Boileau, Hit by a Bus : Physical Access Attacks with Firewire http://www.security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf
24. Fonctionnalités matérielles pour contourner la sécurité des OS, http://actes.sstic.org/SSTIC06/Fonctionnalites_materielles_pour_contourner_la_securite_des_OS/SSTIC06-Duflot_Grumelard-Fonctionnalites_materielles_pour_contourner_la_securite_des_OS.pdf

Sociétés spécialisées

25. Komoku, http://www.komoku.com/
26. PicoComputing, http://www.picocomputing.com/
27. Lexfo, http://www.lexfo.fr/

Outils gratuits

28. PTFinder 0.3.0, http://computer.forensikblog.de/en/2006/09/ptfinder_0_3_00.html
29. Volatools, http://www.komoku.com/forensics/basic.html
30. SystemDump, http://citrite.org/blogs/dmitryv/2006/09/12/new-systemdump-tool/
31. FATKit, http://www.4tphi.net/fatkit/
32. MemParser, http://sourceforge.net/projects/memparser

Outils payants et/ou privés

33. Disk Explorer, `http://www.runtime.org/`
34. Forensic Toolkit, `http://www.accessdata.com/catalog/partdetail.aspx?partno=11000`
35. X-Ways Forensics, `http://www.x-ways.net/forensics/index-m.html`
36. iLook, `http://www.ilook-forensics.org/`
37. WOLF, `http://blogs.technet.com/robert_hensing/archive/2005/01/17/354471.aspx`
38. KnTTools, `http://users.erols.com/gmgarner/KnTTools/`