Le contrôle d'intégrité et ses limites

Cyril Leclerc

ARSeO

1 Pourquoi contrôler l'intégrité ?

Définitions :

Intégrité propriété assurant que des données n'ont pas été modifiées ou détruites de façon non autorisée [ISO 7498-2].

Intégrité garantie que le système et l'information traitée ne sont modifiés que par une action volontaire et légitime [IGI 500].

Intégrité l'intégrité du système et de l'information traitée garantit que ceux-ci ne sont modifiés que par une action volontaire et légitime. Lorsque l'information est échangée, l'intégrité s'étend à l'authentification du message, c'est à dire à la garantie de son origine et de sa destination [IGI n°900].

Le contrôle d'intégrité consiste à s'assurer que celle-ci est préservée afin de détecter et limiter les risques d'erreur et de malveillance. Nous verrons par la suite les différents mécanismes mais regardons d'abord quels sont les risques :

- Un opérateur télécom français a subit une perte évaluée à plus de 20 millions d'euros après la perte d'intégrité d'une base de données gérant la signalisation des mobiles sur le réseau.
- Une banque japonaise détecte au dernier moment un transfert frauduleux de 424 millions de dollars. Le contrôle sur l'intégrité des transferts a fonctionné, cependant des outils de type « keyloggers » auraient été utilisés.
- Moins récemment une base d'adresses d'une administration française est corrompue, de nombreux citoyens reçoivent des documents papier concernant leurs voisins.
- L'intégrité du site internet de SCO (mis en avant sur la scène judiciaire) a été corrompue deux fois en deux jours.
- Plus fréquemment, des millions d'équipements de particuliers sont infectés par divers chevaux de Troie et autres spywares.

Un processus efficace s'assurant de l'intégrité des différentes données critiques d'un système (unique ou d'informations) doit permettre de détecter voire de réparer les pertes avant que l'impact ne se fasse sentir.

2 Panorama du contrôle d'intégrité actuel

L'offre du marché sur les outils de contrôle d'intégrité se décompose ainsi :

2.1 L'Intégrité réseau

Le contrôle d'intégrité réseau est intégré nativement dans les différents protocoles de chiffrement classique au niveau réseau, dans SSL ou dans IPSEC. L'algorithme le plus utilisé est HMAC (RFC 2104).

Le contrôle d'intégrité au niveau des messages réseaux permet de s'assurer que le paquet n'a pas été altéré pendant le parcours sur un simple média de transport ou à cause d'un équipement intermédiaire. L'altération intentionnelle d'un paquet passera inaperçue si le hash classique correspondant est modifié (CRC, MD5, SHA1...). Nous utiliserons donc dans les protocoles sécurisés une technique un peu plus évoluée qui permet d'authentifier l'émetteur (il doit posséder la bonne clé) en même temps que l'on contrôle l'état du paquet.

Le hmac d'un texte ou d'un paquet sera calculé ainsi :

- Deux « pads » (bourrage) différents sont définis : ipad et opad (deux caractères différents et de même taille que la clé d'authentification).
- K est la clé partagée par les deux parties de la communication.
- H est la fonction de hashage (SHA1 et MD5 sont les algorithmes classiquement utilisés).

```
H(K XOR opad, H(K XOR ipad, texte))
```

Sans rentrer dans les détails cryptographiques, on essaye ici de protéger la clé d'authentification K en même temps que l'on contrôle l'intégrité de la communication. Il n'y a pas d'attaques connues contre ce système. La sécurité dépend cependant de la qualités de l'algorithme de hash (particulièrement de la difficulté à trouver des collisions : deux messages clairs donnant le même hash) et bien sûr de la qualité des clés partagées et de leur mécanisme d'échange.

2.2 Au niveau système

Il existe un premier niveau de contrôle d'intégrité intégré au système sous Windows (XP et supérieur) : le WPF (Windows File Protection). Ce mécanisme est conçu pour se protéger des modifications accidentelles plus que comme une réelle protection contre les malveillances : il est assez simple de le désactiver sans provoquer d'alerte particulière.

On trouve ensuite quelques applications commerciales (ou non) classiques (TRIPWIRE, SAMHAIN, AIDE, etc.) chargées de réaliser une surveillance en temps-réel ou non d'un ensemble de fichiers vis-à-vis d'une base d'intégrité actualisée régulièrement. On n'observe pas cependant un énorme choix pour ce type d'outils. Il est de plus intéressant de constater que certains vendeurs ne présentent plus leur produit comme du « contrôle d'intégrité », destiné à se protéger des malveillances, mais plutôt comme un produit permettant de gérer lles modifications (« change management »).

Sous linux, outre le classique TRIPWIRE packagé en standard dans la plupart des distributions, on a vu récemment au moins une distribution majeure intégrer le patch permettant la signature de modules. Ce patch kernel implémente un

contrôle des modules chargés en vérifiant leur signature via un système de clés de type GPG. Il devient impossible de charger un nouveau module sans l'avoir préalablement signé avec la bonne clé. Pour que le système fonctionne, la clé privée doit être insérée dans le noyau avant qu'il ne démarre, et il faut ensuite faire disparaître la clé privée permettant de signer les modules de la machine à protéger.

Au niveau applicatif, le marché ne propose pas (ou peu) d'aide externe (solutions packagées) : le développeur doit aujourd'hui se débrouiller seul. Il existe (existait ?) quelques composants permettant à un serveur web de vérifier qu'il sert des fichiers intègres : si les fichiers présents dans la racine du serveur ne sont pas conformes à une base de référence, on arrête de les servir.

Plus concrètement, dans les applicatifs métier, les contrôles sont orientés sur les données :

Vérification de vraisemblance : la donnée est comparée a une donnée « vraisemblable » dans le contexte (plages de valeurs, seuils...).

Contrôle de format : on vérifie que l'on est en présence d'un nombre à 6 chiffres, une chaîne de 10 caractères alphanumérique, un booléen, un code de sécurité sociale, etc.

Le principe des 4 yeux : normalement un contrôle réalisé par deux personnes physiques, mais la technique est transposable au développement : il est possible de mettre en place deux mécanismes différents de calcul ou de lecture d'une donnée, si un des deux diverge une alerte peut être lancée.

Dans une optique plus préventive, on implémente au mieux le principe de la séparation des tâches/privilèges. Ainsi un processus autorisé à effectuer des opérations en lecture uniquement n'ira en aucun cas modifier la donnée.

3 Les contraintes du contrôle en production

Ces principes théoriques de contrôle peuvent paraître satisfaisants dans beaucoup de situations. Cependant, en production, on constate quils ne sont pas ou peu adaptés.

Dynamisme du système

Certaines parties du système dexploitation sont soumises à des changements fréquents et normaux :

- Les répertoires temporaires, (`/tmp`, `c :\windows\temp`).
- Les queues de mails (`/var/spool/mail`).
- Les caches applicatifs (`/var/cache/squid`, `/var/lib/ldap`, `\temporary internet files`).
- Etc.

Les outils de contrôle doivent être paramétrés pour ne pas déclencher d'alertes en cas de modification ou de création de fichiers dans ces répertoires : autant d'emplacements rêvés pour déposer quelques fichiers aux desseins peu recommandables.

Gestion du changement

Bien sûr les systèmes ne sont pas figés, nous devons mettre à jour nos produits et applications, pages webs, fichiers à transférer, base de données, etc. Si le contrôle d'intégrité se déclenche à chacun de ces changements (pourtant planifiés et normaux), il perdra très vite toute crédibilité et une réelle intrusion passera inaperçue. Il est donc nécessaire d'intégrer une mise à jour de la base d'intégrité à chaque modification.

Qui peut réaliser ce changement ? Comment doit marcher l'authentification avant de resceller la base d'intégrité ? Peut-on automatiser de façon sécurisée un tel processus ?

Autant de questions auxquelles il est difficile de donner une réponse, l'exploitant responsable des mises en production ne devrait pas avoir le droit de modifier le contenu. Il apparaît alors compliqué de faire taper un mot de passe pour mettre à jour le scellement de la base.

Autre cauchemard pour le contrôle d'intégrité : les mises à jour de sécurité « fréquentes ». Les besoins de réactivité face aux menaces virales imposent un niveau d'automatisation important du déploiement des correctifs. L'automatisation de la mise à jour de la base d'intégrité est un processus dont un attaquant peut tirer profit : il va attendre une mise à jour spécifique pour réaliser ses propres modifications et être automatiquement intégré à la nouvelle base de référence.

3.1 Manipulation de la base d'intégrité (faiblesse ou absence de scellement)

Les outils de scellement listent l'ensemble des fichiers présents dans un répertoire (ou sur l'ensemble d'un système de fichiers) et réalisent des hash (voir §2.1) pour chacun de ces fichiers qui constituent la base de signatures. Lorsqu'une vérification est réalisée, l'outil compare les hashs obtenus pendant le scan avec ceux de la base, chaque différence témoignant dune modification d'un fichier.

La protection de cette base de signatures est donc un sujet critique. Les outils de type Tripwire opèrent un scellement de cette base (avec une signature cryptographique).

Il reste cependant possible de modifier l'outil pour qu'il ne rapporte pas les modifications concernant certains fichiers, par exemple lui-même, mais aussi les fichiers contenant le cheval de Troie.

Il apparaît donc risqué de laisser l'outil de contrôle sur la machine à surveiller. Un bon contrôle devra être réalisé avec un outil déposé au moment de la vérification. L'intégrité de l'outil est aussi importante que l'intégrité de la base de signature.

4 Les différents vecteurs d'injection et de dissimulation des chevaux de Troie, rootkits et autres virus

4.1 API Hooking

Le hooking consiste à injecter du code dans l'environnement système ou d'un processus et à l'activer au bon moment, en général en interceptant une exécution (interception, appel de fonction, jump...) pour la rerouter vers le code injecté. La problématique du contrôle d'intégrité du système de fichiers confronté à cette technique devient beaucoup plus complexe : le hooking peut être utilisé pour modifier la fonction permettant de lister les fichiers présents dans un répertoire, et ainsi cacher la présence de nouveaux fichiers. Une autre fonction permettant la lecture d'un fichier peut être modifiée afin de renvoyer un contenu présent dans un autre fichier. Similairement, il est possible de changer le flag SUID d'un fichier et de faire en sorte que le contrôleur d'intégrité ne voie pas cette modification. Autre exemple, la base de registre peut être modifiée en sorte que la nouvelle clé soit invisible de tous les outils ne réimplémentant pas les appels du noyau système.

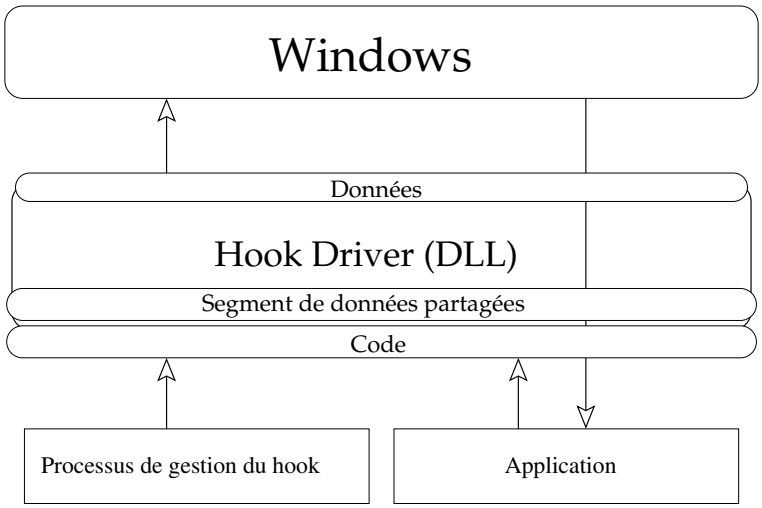
Plus spécifiquement, il est possible de hooker la fonction d'affichage des résultats de l'outil de contrôle afin de cacher l'alerte concernant une modification d'un fichier.

L'objectif étant exposé, rentrons maintenant dans le détail de cette technique. Le hooker a donc deux préoccupations : injecter et intercepter. Nous allons distinguer deux types de hooking, le premier nécessite des modifications sur le système de fichiers qui pourront être détectées par un outil de contrôle classique (si aucune autre mesure de protection n'est prise), le second type se contente de modifier l'exécution des différents processus et passera donc inaperçu.

Hooking Non volatile

Injection

- *Kernel Hooking.*- Cette méthode implémentée dans les chevaux de Troie modernes permet des hooks directement dans les API Kernel. Nous modifions alors les fonctions de très bas niveau.
- *System-wide Windows Hooks.*- C'est une méthode documentée et standard, parfaitement intégrée dans l'ensemble de la famille Windows (9x, NT, 2k, XP, 2k3). Il permet d'enregistrer de nouvelles fonctions (via une DLL) dans l'ensemble des processus du système (cf. figure 4.1) . Il est également possible de supprimer un hook « system wide » sans redémarrage, cette méthode est souple et fiable. Après injection et enregistrement, la DLL ne peut-être supprimée du système de fichiers. Il est nécessaire d'utiliser d'autres techniques pour rester non détectable. Un processus dit « Hook Server » se charge de l'enregistrement des DLL dans les différents processus actifs [4].
- *Injection via la fonction « `CreateRemoteThread` ».*- Cette méthode moins standard ne fonctionne pas sur les systèmes de type Windows 9x. L'idée



captionLes hooks systèmes Windows

est que chaque processus peut charger une librairie dynamiquement via la fonction « `LoadLibrary` ». Nous n'avons cependant pas la main sur l'exécution des différents processus. C'est alors qu'intervient la fonction « `CreateRemoteThread` » qui va nous permettre de lancer du code dans le contexte d'exécution d'un processus donné :

```
hThread = ::CreateRemoteThread(
        hProcessForHooking, // Le handler du process a hooker
        NULL, 0, pfnLoadLibrary, // Le pointeur vers loadlibrary
        "C:\\HookTool.dll", // La dll contenant les hooks
        0,
        NULL
);
```

Ici encore, si l'on utilise « `LoadLibrary` », on ne peut plus supprimer la DLL du système.

- *Injection via la base de registre.*- Il existe une clé qui spécifie l'ensemble des librairies devant être lié à la DLL `USER32.dll` `HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\Windows\AppInit_DLLs`. En indiquant l'emplacement d'une DLL contenant nos fonctions hostiles, nous nous assurons qu'elles seront chargées par l'ensemble des processus. Cette méthode manque de flexibilité, il est nécessaire de redémarrer le système à chaque modification.
- *Hook spécifiques.*- Il existe encore deux autres méthodes spécifiques d'injection :
  - La première cible Internet Explorer : les « Browser Helper Components » (BHO). Ces derniers sont des DLL spécifiques qui seront chargés par Internet Explorer avant son exécution.

– La seconde implémente un mécanisme similaire mais uniquement pour les applications MS Office.

Interception Une fois notre code injecté et utilisable dans le contexte d'exécution de notre cible (ou de nos cibles), il convient de faire en sorte qu'il soit effectivement lancé. Nous devons donc modifier le flux d'exécution du ou des processus : c'est l'interception.

– *Windows subclassing.*- On utilise la fonction documentée « `SetWindowLongPtr` » avec GWLP_WNDPROC. Une fois cet appel défini (au chargement de la librairie hookée), Windows va à chaque nouvel envoi de message vers une fenêtre spécifique chercher l'adresse de la procédure à appeler et donc utiliser notre nouvelle adresse. Cette fonction n'est utilisable que dans l'environnement d'un processus donné, il ne fonctionnera qu'avec les méthodes d'injections spécifiques à une application (BHO par exemple).
– *Utilisation d'une* DLL *« Proxy »*.- C'est une méthode simple et efficace mais elle nécessite une modification du système de fichiers. On remplace une dll existante par une autre du même nom exportant la même table de symboles puis on utilise une fonction de transfert (forwarder) pour renvoyer vers la DLL renommée qui contient les fonctions légitimes :
```
#pragma comment(linker,
      "/export:DoSomething=DllImpl.ActuallyDoSomething")
```
– *Hooking volatile.*- On constate aujourd'hui que certains chevaux de Troie se contentent de modifier l'exécution de certains programmes en mémoire ; il n'y a pas de modification persistante du système de fichiers pouvant être détecté. Nos outils classiques ne regardent évidemment pas les modifications de la mémoire (qui par nature change constamment). Les besoins de disponibilité sont de plus en plus élevés et les systèmes ne sont que rarement redémarrés, une telle modification sera active et non détectable tant que la machine ne redémarre pas.

L'Injection via la fonction « `CreateRemoteThread` » sans « `Loadlibrary` » Une alternative à la fonction « `LoadLibrary` » consiste à utiliser les fonctions permettant d'allouer de la mémoire dans le processus distant afin d'y intégrer nos nouvelles fonctions. Casper [5] utilise cette méthode :

```
pInjCode = (PDWORD)pData->fnVirtualAllocEx
                (hProcess, 0, INJECT_SIZE,
                MEM_COMMIT, PAGE_EXECUTE_READWRITE);

[..]

hThread = pData->fnCreateRemoteThread
                (hProcess, NULL, 0,
                 (DWORD(__stdcall *) (void *)) pInjCode,
                 pInjData, 0 , &dwThreadId);
```

L'interception

- *Réécriture du code.*- Cette technique est employée et documentée dans « Casper » Elle consiste à réécrire des portions de codes dans le processus une fois chargé en mémoire en utilisant la fonction « `WriteProcessMemory` » On va ici remplacer les adresses utilisées par un JMP ou un CALL dans le code pour pointer vers les fonctions préalablement injectées. C'est une opération délicate qui ne respecte pas les principes du multithreading et qui est difficilement portable. Avec un peu d'effort, elle fonctionne cependant.
- *Modification de l'*IAT.- Cette technique utilise les particularités du format Windows PE (Windows Portable Executable).
  Un en-tête PE a la structure suivante :

En-tête MS-DOS (« MZ ») et « stub »	Adresse 0
Signature PE (« PE »)	
.text code du programme	Le code du module et les données initialisées (globales et statiques)
.data données initialisées	Les informations concernant les fonctions et les données importées
.idata table d'import	Les informations concernant les fonctions et les données importées
.edata table d'export	Les informations concernant les fonctions et les données exportées
Symboles de Debug	

Tab. 1. Structure de l'en-tête PE

L'IAT est la section de l'en-tête de l'exécutable qui contient la table des symboles, c'est-à-dire le nom de chaque fonction et l'adresse à laquelle elle se situe. En modifiant l'adresse d'une fonction dans l'IAT pour la faire pointer vers notre code, nous nous assurons que chaque appel à cette fonction sera redirigé vers notre code (qui devra après son exécution remplacer ou rediriger l'appel vers la fonction originale). Cette technique est complètement portable et fonctionnera sur toute la famille Windows. Cependant certaines fonctions sont appelées sans chercher la correspondance dans l'IAT (« `Loadlibrary` » and « `GetProcAddress` »), on ne peut donc pas les hooker par ce biais. Il est à noter que les modifications de l'IAT peuvent être détectées en pointant une différence entre le fichier sur le disque et l'exécution en mémoire.

4.2 Furtivité sur les I/O disques / registry / etc.

Pour échapper à la vigilance des détecteurs de perte d'intégrité, nous pouvons, grâce à cette technique de hook, modifier les appels Kernel afin de rendre

le détecteur complètement aveugle aux nouveaux fichiers et clés de la base de registre présents.

Exemple de fonctions du Kernel NT qu'il peut être intéressant de modifier :

« NtCreateFile » Cacher tout les fichiers créés par un processus particulier,

« ZwOpenFile » Renvoyer une erreur lorsque l'on tente d'ouvrir un fichier caché,

« ZwQueryDirectoryFile » Cacher un fichier dans un répertoire,

« ZwOpenKey » Renvoyer une erreur lors de l'ouverture d'une clé dans la base de registre

« ZwQueryKey »

« ZwQueryValueKey » Modifier le contenu d'une clé

« ZwEnumerateValueKey »

« ZwEnumerateKey » Cacher des clés

« ZwSetValueKey »

« ZwCreateKey »

Le remplacement de ces fonctions (lectures/écritures sur le système de fichiers et dans la base de registre), au delà de leur fonction première (ie. se cacher de l'administrateur et de ses outils), permet de rendre les processus de détection de perte d'intégrité complètement aveugles.

Des nouveaux outils décris par MICROSOFT [2] et SYSINTERNALS [1] sont disponibles, il permettent de lire les fichiers « ruche » de la base de registre directement sur le système de fichiers et de comparer les différences entre la liste des clés obtenues via les appels kernel classique et le contenu directement analysé. Les différences sont probablement des clés cachées par un mécanisme de hook.

Il est possible de descendre encore le niveau de hook pour intercepter les IRP (I/O request packet, les API permettant les lectures sur le disque) mais aucune implémentation fiable et complète n'existe actuellement.

Un « proof of concept » a cependant été publié dans PHRACK 62 [3].

Il permet de cacher les N premiers octets d'un fichier en hookant les IRP suivants :

```
//This module hooks:

// IRP\_MJ\_READ, IRP\_MJ\_WRITE, IRP\_MJ\_QUERY\_INFORMATION,

// IRP\_MJ\_SET\_INFORMATION, IRP\_MJ\_DIRECTORY\_CONTROL,

// FASTIO\_QUERY\_STANDARD\_INFO FASTIO\_QUERY\_BASIC\_INFO
// FASTIO\_READ(WRITE)

//to hide first N bytes of given file
```

4.3 Furtivité réseau

En réalisant des hooks sur les appels réseaux, ou les descripteurs de la pile IP, il est possible de cacher des flux réseaux aux sniffeurs locaux (SEBEK implémente cette technique pour envoyer entre autres l'ensemble des touches tapées au clavier vers un serveur). HXDEF [6] implémente de plus une technique permettant d'intercepter les connexions sur les services actifs. Il ne nécessite donc pas la création d'un port en écoute supplémentaire qui pourra être détecté par un simple scanneur de port. Il pourra par exemple infecter un serveur web protégé par un firewall laissant passer les seuls flux à destination du port 80 et rester accessible.

5 Le risque résultant

La détection d'un cheval de Troie, d'un virus ou d'un vers peut-être complexe, comme nous venons de le voir. De plus, s'il sont bien faits, la détection sera obligatoirement issue d'une opération manuelle réalisée par une personne compétente.

La question se pose donc : « Doit-on réinstaller un serveur à chaque nouvelle vulnérabilité publique exploitable ? ».

« Ça dépend » nous soufflera le consultant normand.

En fonction de la criticité de l'environnement à protéger et de son niveau d'expositions, différentes mesures s'imposent. Il faut intégrer les différents paramètres et prendre une décision en conséquence :

- Existence d'un outil (ou pire d'un vers) permettant l'exploitation.
- Simplicité de son utilisation.
- Exposition de l'équipement (la machine est en frontal sur Internet, ou est située au fond d'un lan surprotégé).
- Durée de l'exposition de l'équipement.
- D'autres équipements ont déjà été touchés (dans le même SI ou dans d'autres entreprises).
- L'absence ou la présence d'outils permettant de limiter les risques (stack non executable, un reverse proxy en frontal d'un web, un mécanisme d'Intelligence Applicative dans le firewall, une sonde de détection, etc).

Le responsable sécurité doit donc arbitrer :

- Le correctif vient tout juste d'être diffusé et aucun exploit n'est disponible. Dans ce cas, une simple application des correctifs accompagnée d'un contrôle pour les machines exposées peut sembler suffisante.
- Autre cas, l'exploit a été diffusé publiquement avant l'existence d'un correctif efficace et notre machine est frontalement exposée. Un premier contrôle ne semble indiquer aucun problème apparent. Le dilemme prend alors corps : soit on réinstalle tout l'équipement à partir de support de sauvegarde sains, soit on prend le risque de continuer ainsi. Dans ce cas un niveau supplémentaire de surveillance s'impose : observation des courbes de traffic, utilisation des outils de SYSINTERNALS ROOTKITREVEALER et autre chasseurs de chevaux de Troie.

6 Démo d'infection, dissimulation avec un contrôle d'intégrité actif

Cette démonstration rapide pointe l'efficacité du rootkit HXDEF [6] (légèrement modifié pour l'occasion) pour cacher intégralement sa présence à un outil de contrôle d'intégrité, l'antivirus et le détecteur de spyware actifs, un scanneur de port et NETSTAT.

La démonstration se déroule ainsi :

1. Scellement du système (le répertoire racine, la base de registre et le répertoire windows dans son ensemble).
2. Un premier scan de port (nmap) ainsi qu'un dump des ports en écoute sur l'équipement (netstat).
3. Installation d'hxdef qui va modifier l'ensemble des appels Kernel permettant de le détecter et qui va activer sa backdoor.
4. Vérification de l'intégrité des fichiers systèmes et de la base de registre.
5. Vérifications des ports en écoute.
6. Détection du trojan avec d'autres outils spécifiques (ROOTKITREVEALER).

7 Conclusion

Comme nous l'avons vu dans cet article, l'approche classique consistant à détecter des incohérences et modifications des systèmes de fichier en faisant confiance aux fonctions système est largement insuffisante.

Une première solution consiste à n'être jamais vulnérable. Ainsi aucune perte d'intégrité n'est à craindre. Cette approche manque un peu de piment, nous devons donc trouver d'autres solutions. A priori, il apparaît difficile de laisser un système infecté contrôler sa propre intégrité.

Il conviendra de préférer des systèmes distribués pour conserver l'intégrité du contrôleur lui-même. L'outil d'intégrité peut réimplémenter la lecture des partitions (NTFS/FAT) lui-même afin de se protéger des hooks classiques (il reste cependant la cible d'autres hooks).

Dans le domaine de la prévention, les protections empêchant la modification de l'exécution du noyau (nécessité d'utiliser des drivers signés) peuvent être pertinentes dans des environnements sensibles et suffisamment statiques.

Des techniques assez récentes permettent de détecter des incohérences entre le système de fichier lui-même et le résultat renvoyé par les appels système et kernel potentiellement corrompus. Ces approches détectant le comportement déviant (cacher une clé de la base de registre, hooker un appel réseau) semblent ouvrir une voie prometteuse.

8 Remerciements

Merci à Stéphane Jourdois et Paul Grassart pour leurs multiples corrections éclairées et pour la mise en page.

Références

1. SysInternals, *rootkitrevealer*, `http://www.sysinternals.com/ntw2k/freeware/rootkitreveal.shtml`, 2005.
2. Microsoft, *MSR Strider Project*, `http://research.microsoft.com/rootkit`, 2005.
3. Firew0rker, *Kernel Mode Backdoor for NT*, Phrack Vol. 62, article 6, `http://www.phrack.org/show.php?p=62\&a=6`, 2004.
4. I. Ivanov, *API hooking revealed*, `http://www.codeproject.com/system/hooksys.asp?df=100\&forumid=3602\&exp=0\&select=983089`, 2002.
5. Valgasu, *Casper, The Friendly Trojan*, `http://valgasu.rstack.org/casper/`
6. Holy Father, *Hacker deffender - Rootkit*, `http://hxdef.czweb.org`