WiShMaster

---

1 Préambule

L'article précédent a présenté le principe de WiShMaster, un outil permettant de générer des shellcodes pour Windows à partir d'un ensemble de fichier sources.
Ce second article présente un exemple d'utilisation basé sur la shellcodisation d'une backdoor toute simple afin d'améliorer ses capacités. Après une rapide présentation du principe de la backdoor, il présentera les possibilités ouvertes par l'opération de shellcodisation et se terminera par une évaluation de la détection de la backdoor sous sa nouvelle forme par quelques firewalls personnels.

Il faut bien noter que l'objectif de WiShMaster est de transformer un code en shellcode afin de pouvoir le manipuler aisément.
L'utilisation exposée dans cet article reste donc un exemple parmi un ensemble de possibilités, que j'ai choisi car il s'inscrit dans la continuité de recherches personnelles sur la problématique de filtrage des flux sortants. Il permettra de plus de détailler un peu le fonctionnement des firewalls personnels.
WiShMaster n'a en aucun cas pour but ultime de faire de la protection ou de l'amélioration de code malicieux.
WiShMaster est un freeware et peut être téléchargé sur mon site personnel [1]. Cette page contient notamment quelques vidéos de démonstration complétant cet article.

2 Présentation de RConnect

2.1 Principe de RConnect

RConnect est une backdoor très simple programmée en moins d'une heure. Son principe est de se connecter sur un serveur (technique de « reverse-connect »), puis de servir d'intermédiaire entre un processus « cmd.exe » caché s'exécutant sur la cible et le serveur.
Pour le serveur, nous utiliserons simplement l'outil « netcat » [1].

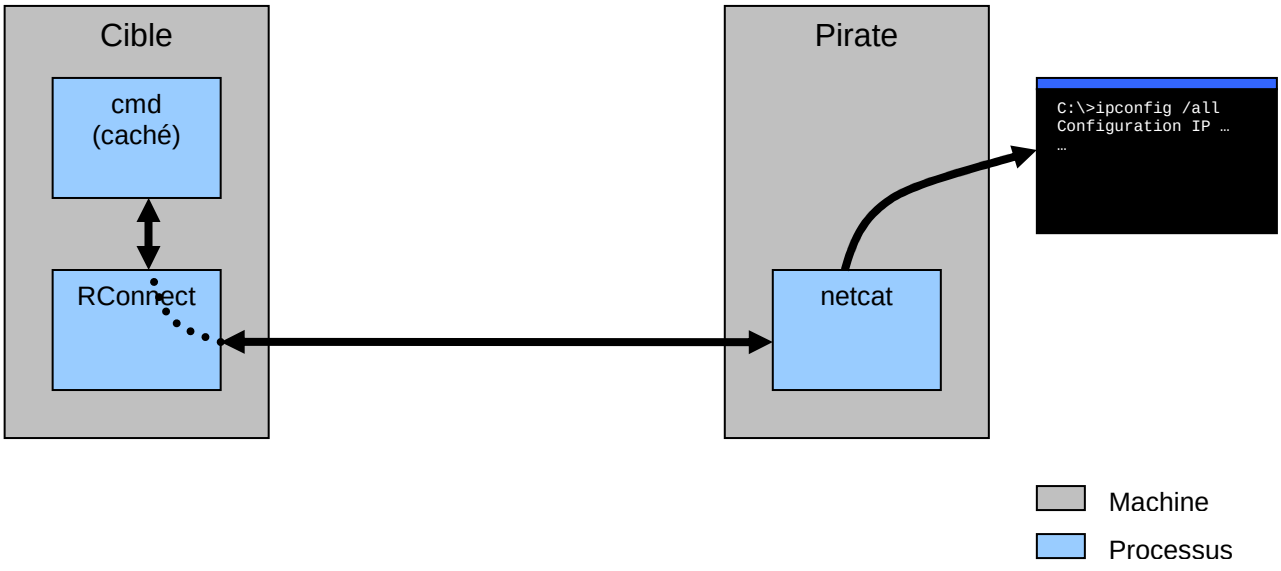

Fig. 1 Principe de fonctionnement de RConnect

Une fois la connexion réseau établie, les commandes saisies au niveau de netcat sont envoyées à RConnect, qui les transmet au processus « cmd » caché où elles sont interprétées. La sortie standard du processus « cmd » est récupérée par RConnect et transmise au serveur netcat où elle est affichée.
Le pirate dispose ainsi d'un accès « cmd distant ».

Il faut noter que les données circulent au niveau du réseau directement encapsulées dans TCP. La connexion entre la cible et le pirate ne peut donc s'établir à travers un proxy web.

Le code de RConnect représente une centaine de lignes. Plusieurs versions peuvent être trouvées assez facilement sur le web.

2.2 Les défauts de RConnect

RConnect comporte plusieurs limitations importantes :

L'exécution de la backdoor n'effectue pour l'instant aucune opération visible, ce qui risque fortement d'éveiller les soupçons de l'utilisateur. L'idéal serait qu'elle soit inclue dans une application réelle par exemple dans un petit jeu. Cependant, cette solution requiert d'avoir les sources de ce logiciel, afin de pouvoir inclure la backdoor et compiler l'exécutable final.

Ensuite, la vie de la backdoor est liée à celle du processus d'origine. Le pirate perdra donc l'accès à la cible lorsque l'utilisateur terminera ce processus, par exemple en quittant le petit jeu.
Pour résoudre ce problème, nous pouvons par exemple simuler la fin du programme en cachant la fenêtre principale et en poursuivant l'exécution en arrière plan. Il faut cependant noter que le processus apparaîtra toujours dans la liste des processus actifs.

Ensuite, le programme contenant RConnect sera inconnu des firewalls personnels. Si le poste cible est équipé d'une telle protection, la tentative d'accès de la backdoor au réseau sera détectée et généralement une popup de confirmation sera affichée.

Enfin, le code de RConnect est « statique ». Il est donc aisé de construire une signature antivirale le caractérisant. RConnect sera ensuite instantanément arrêtée par les anti-virus lors de son arrivée sur le poste cible.
Il est possible de la rendre à nouveau indétectable en modifiant la portion de code sur laquelle est basée la signature, cependant cette opération nécessite d'avoir une idée de l'antivirus installé, afin de connaître la partie du code à modifier et de recompiler la backdoor.

RConnect comporte donc actuellement des limitations très importantes qui la rendent relativement peu utilisable dans un contexte réel.

3 La shellcodisation de RConnect

RConnect a été écrite en respectant les conventions de codage de WiShMaster et peut donc être shellcodisée. Analysons si cette transformation permet d'éliminer certaines des limitations évoquées ci-dessus.

3.1 Le brouillage de RConnect

Après l'opération de shellcodisation, RConnect se présente sous la forme d'un tableau d'octets continus qu'il est très facile d'encoder par un algorithme type XOR. L'exécutable final est alors composé de deux parties : la partie déchiffrement et le shellcode RConnect brouillé.

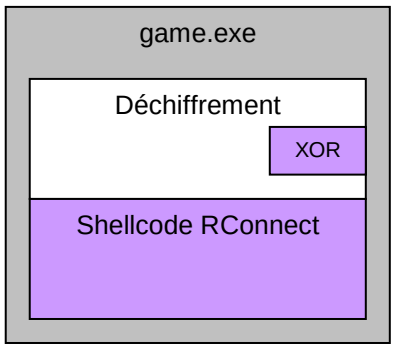

Fig. 2 Structure de l'exécutable contenant la backdoor shellcodisée et encodée

Il devient très difficile de construire une signature antivirale de la backdoor ; en effet la partie déchiffrement peut être rapidement réécrite pour produire un code totalement différent. La partie shellcode peut être intégralement changée par simple modification de la clé XOR de brouillage.

Dans sa forme la plus simple, la partie déchiffrement se contente de déchiffrer le shellcode RConnect et de lui transmettre l'exécution. Il est cependant possible d'écrire un code plus évolué contenant par exemple des fonctionnalités anti-émulation afin de déjouer les antivirus utilisant cette technique.

3.2 L'injection de RConnect dans un autre processus

3.2.1 Principe de la technique d'injection de thread

Sous Windows, les processus sont constitués d'un espace mémoire privé « à plat » de 4 Go, d'au moins un thread et d'un ensemble de ressources (handles de fichiers ouverts,…).
En règle générale, les ressources d'un processus sont privées et ne peuvent être altérées par un autre processus.
Il existe cependant certaines fonctions permettant de contourner cette barrière. Dans notre cas, les trois fonctions suivantes sont particulièrement intéressantes :

- VirtualAllocEx : permet d'allouer un bloc de mémoire dans l'espace mémoire d'un autre processus.
- WriteProcessMemory : permet d'écrire des données dans l'espace mémoire d'un autre processus.
- CreateRemoteThread : permet de créer un thread dans un autre processus (en lui fournissant également l'adresse de début d'exécution).

Un processus peut exécuter du code dans un autre processus en combinant ces trois fonctions : il commence par allouer une portion de mémoire dans le processus cible en appellant VirtualAllocEx et la remplit avec du code via WriteProcessMemory. Il crée ensuite un nouveau thread dans ce processus en indiquant le début du buffer alloué comme point d'entrée :

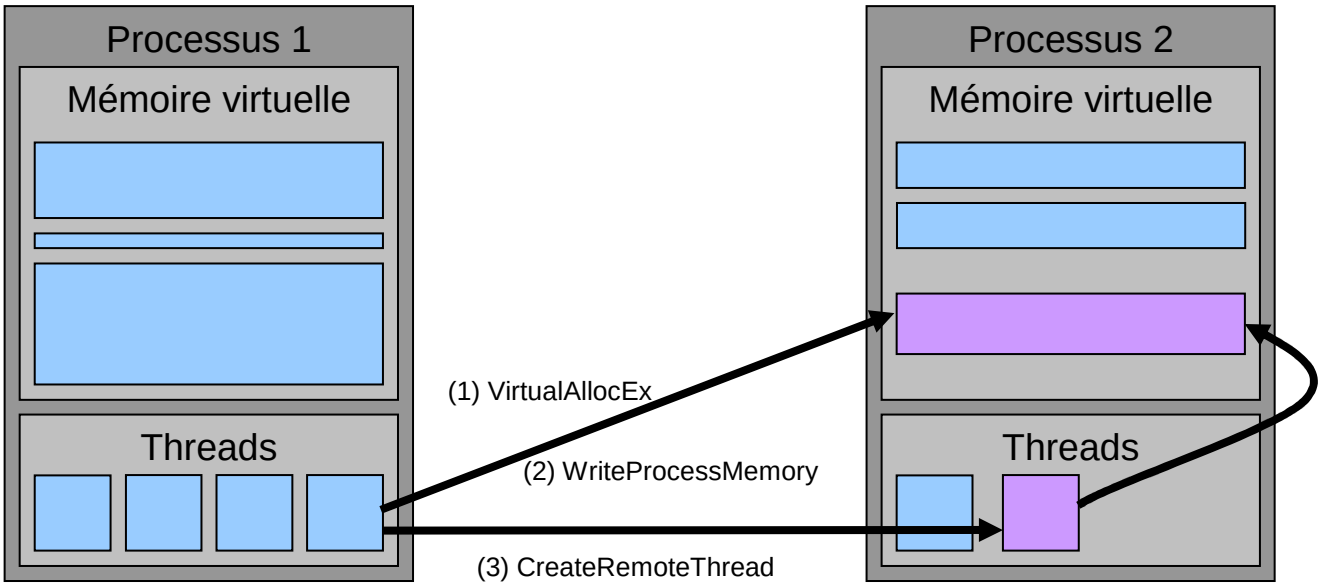

Fig. 3 Principe de la création l'injection de thread dans un processus distant

Cette technique d'injection de thread impose cependant deux contraintes :
Tout d'abord, les processus sont des éléments représentés dans le noyau au niveau de l'Object Manager et possédant des droits ; ces opérations sont donc soumises à un contrôle d'accès. Par exemple, un processus lancé par un utilisateur restreint ne pourra (heureusement) pas injecter du code dans un processus « SYSTEM ».
Il n'est par contre nul besoin d'être administrateur pour effectuer ces opérations ; un processus lancé dans une session « utilisateur restreint » pourra agir sur tous les processus de la session.
Ensuite le code injecté s'exécute à une adresse inconnue dans un espace mémoire inconnu. Il doit donc être totalement autonome et relocalisable, c'est-à-dire être un shellcode.

3.2.2 Utilisation de l'injection de thread

Maintenant que RConnect est un shellcode, il devient donc possible de l'injecter dans un autre processus.
Par exemple, nous pouvons lancer un navigateur caché et l'injecter avec RConnect. Cette technique résout deux des problèmes cités précédemment :
Tout d'abord, la backdoor va s'exécuter dans le processus navigateur et devient indépendante du processus d'origine. L'arrêt de celui-ci n'aura aucun impact sur la vie de la backdoor.
Ensuite, les connexions établies par la backdoor seront vues par les firewalls personnels comme provenant d'un processus navigateur, autorisé à accéder au web, et ne provoqueront aucune alerte.

3.2.3 Mise en œuvre de l'injection

Pour réaliser l'injection, nous allons utiliser un « injecter ». Il s'agit d'un programme contenant une fonction qui prend en paramètre un buffer et sa taille, le déchiffre avec une clé XOR, récupère le chemin vers le navigateur par défaut, lance une instance suspendue et cachée de ce programme et l'injecte avec le shellcode.

L'« injecter » va bien sûr lui-même être shellcodisé par WiShMaster et brouillé avec une autre clé XOR, si bien que nous obtenons finalement un shellcode composé de la version shellcodisée de « injecter », suivie par RConnect shellcodisé.

Si nous incluons ce shellcode dans un programme effectuant directement le déchiffrement et transférant l'exécution à l'injecter, nous obtenons un exécutable ayant la structure suivante :

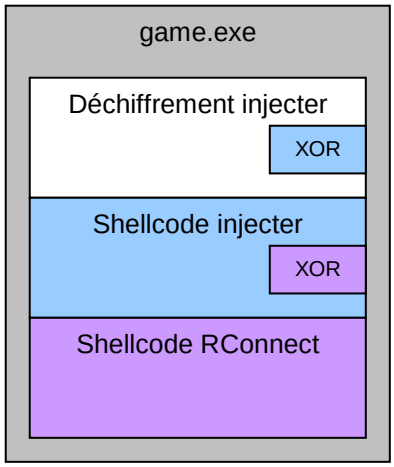

Fig. 4 Structure de l'exécutable formé de l'injecter et de RConnect shellcodisés

Les parties en couleur sont encodées par la clé de la couleur correspondante.

Lors de l'exécution, la partie « Déchiffrement injecter » va déchiffrer l'injecter et lui transmettre l'exécution. Celui-ci va lancer une instance cachée du navigateur par défaut, déchiffrer RConnect et l'injecter dans ce processus.
L'instance de RConnect dans le navigateur va alors se connecter sur le serveur du pirate, connexion qui sera autorisée par le firewall personnel.

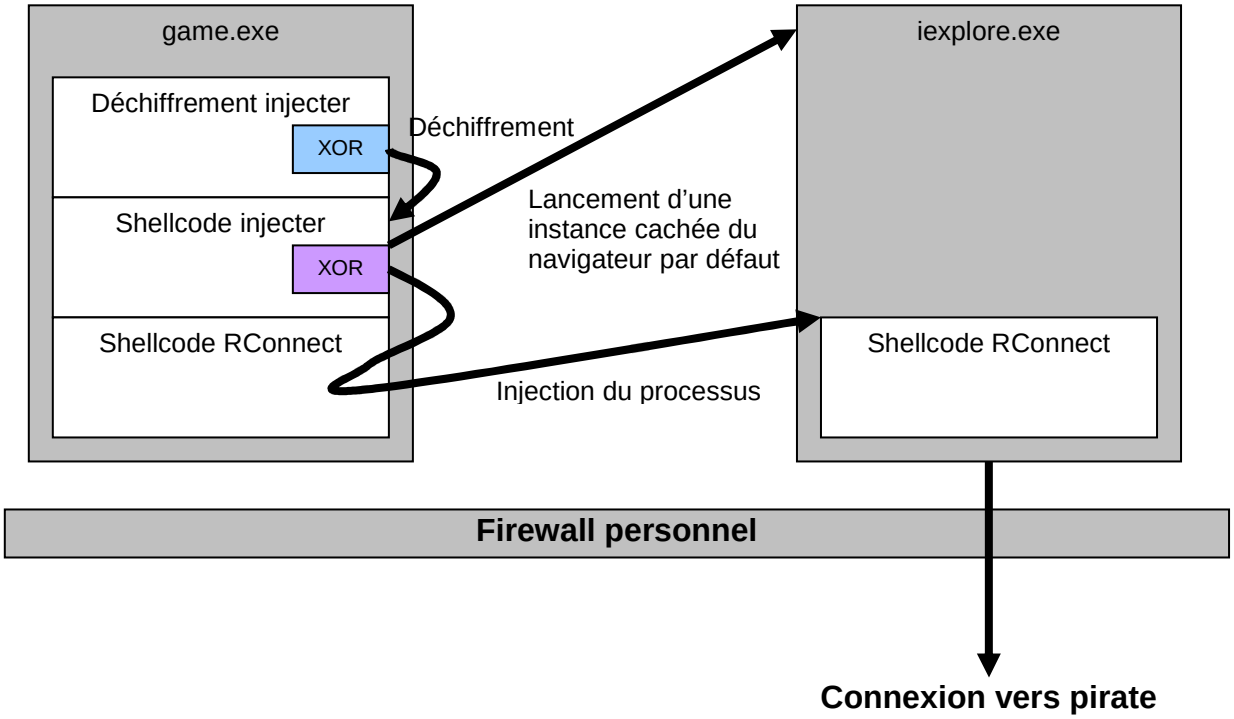

Fig. 5 Principe de l'exécution de RConnect en injection dans un navigateur caché

3.3 L'encapsulation de la backdoor

Maintenant que RConnect est un shellcode, il devient relativement facile d'infecter un exécutable quelconque, à la manière d'un virus.
Un exécutable sous Windows est au format PE (Portable Executable). Comme décrit dans la première partie, le fichier est composé d'entêtes, suivies de sections contenant le programme en tant que tel (code et données) et des informations utilisées par Windows pour créer le processus (table d'importation, …).

L'une des entêtes contient notamment l'adresse du point d'entrée, c'est-à-dire l'adresse de première instruction exécutée lors de la création du processus (pour être précis, il s'agit de son adresse virtuelle relative, mais nous ne rentrerons pas plus dans les détails).

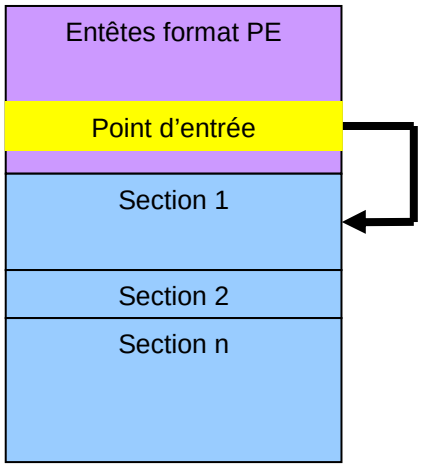

Fig. 6 Structure d'un exécutable au format PE

Une possibilité d'infection relativement simple consiste à ajouter le shellcode dans la dernière section en agrandissant celle-ci à la taille adéquate. Le point d'entrée est ensuite modifié pour pointer vers le début du shellcode :

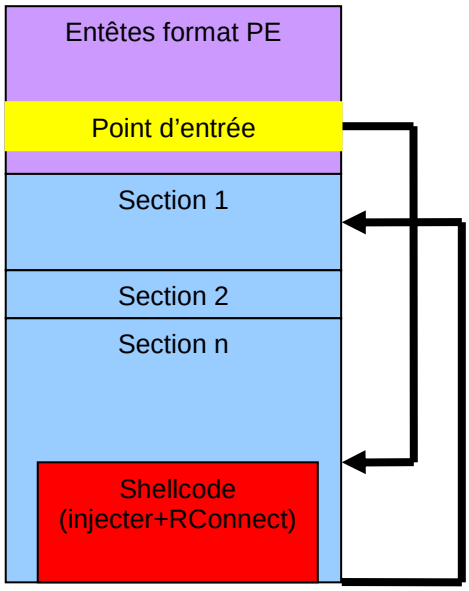

Fig. 7 Structure d'un exécutable au format PE après infection

Un saut est ajouté à la fin du shellcode afin que l'exécution normale du programme se poursuive. Il est relativement rapide de programmer un outil réalisant ce type d'infection de manière automatique à partir d'un shellcode et d'un exécutable quelconque.
Dès lors, nous n'avons plus besoin d'avoir le code source d'un logiciel. N'importe quel exécutable peut devenir porteur de la backdoor.

Il faut noter que la présence d'un point d'entrée dans la dernière section risque d'augmenter la probabilité de détection par les antivirus. Il est bien sûr possible de programmer un outil un peu plus perfectionné qui ajouterait par exemple des sauts dans les données inutiles à la fin des différentes sections.

4 Evaluation de la détection de RConnect shellcodisé

L'objectif de cette partie est de présenter les résultats d'une évaluation de la détection de RConnect shellcodisée par différents firewalls personnels couramment utilisés.
Commençons par détailler un peu le fonctionnement des firewalls personnels.

4.1 Le concept des firewalls personnels

Un firewall personnel est un programme de protection qui contrôle les accès au réseau effectués par les différents éléments du système.
Mis à part quelques composants du système, la notion d'élément correspond à un exécutable du système de fichiers.
Deux modes d'accès sont généralement différentiés : le mode client où l'élément essaye de se connecter sur un serveur et le mode serveur où il se place en attente de connexion. Pour chaque mode, l'accès de l'élément pourra être autorisé, interdit ou inconnu.
Le firewall dispose d'une base locale conservant les autorisations pour les différents éléments. Lorsqu'un accès a lieu, il consulte cette base et réagit en conséquence. Si l'élément effectue ce type d'accès pour la première fois (accès inconnu), le firewall personnel affiche une popup pour demander à l'utilisateur la conduite à tenir et intègre éventuellement cette réponse à la base.

De manière très schématique, il existe trois catégories de firewalls personnels :

- le FW personnel de XP SP2
- les FW personnels basiques
- les FW personnels avancés

4.2 Le FW personnel de XP SP2

Le service pack 2 de Windows XP intègre un « pseudo » firewall personnel. « Pseudo » car il n'effectue aucun filtrage des flux sortants (accès en mode client) !
Il n'apporte en conséquent aucune sécurité face à des backdoors comme RConnect (même dans sa forme originelle).

4.3 Les FW personnels basiques

Les firewalls personnels « basiques » effectuent un filtrage des flux entrant et sortant.
La connexion de RConnect dans sa forme originelle déclenchera l'affichage d'une popup car l'exécutable est inconnu de la base de données du firewall.
En revanche, lors de son exécution sous forme de thread injecté dans un processus navigateur, le firewall verra un accès client du navigateur et l'autorisera.

Ce type de firewall personnel peut donc être contourné en combinant la shellcodisation et l'injection de RConnect.

4.4 Les FW personnels de avancés

Les firewalls personnels « avancés » effectuent également un filtrage des flux entrant et sortant, mais analysent en plus les opérations effectuées par les applications afin de détecter des comportements suspects et par exemple de bloquer les tentatives d'injection de thread.

Cette surveillance peut être mise en place via différentes techniques. Sans chercher à en dresser une liste exhaustive, cette partie expose trois techniques couramment utilisées : le hooking user-land par patch du header, le hooking kernel-land par patch de la SSDT et l'installation de fonctions callback.
Elle présentera ensuite rapidement quelques cas concrets d'implémentation dans des firewalls personnels existants pour évaluer leurs limites.

Le choix de firewalls personnels présentés dans cette étude est purement illustratif. Ce document n'a pas pour objectif de procéder à une évaluation du meilleur firewall personnel.

4.4.1 Rappels sur le passage en mode noyau sous Windows

Pour comprendre ces techniques, revenons rapidement sur le passage en mode noyau sous Windows. Le système Windows expose un ensemble de fonctions systèmes qui sont accessibles via le vecteur d'interruption 2Eh. A partir de Windows XP, si le processeur supporte les « Fast System Call », le passage en mode noyau de fait plutôt via l'instruction « sysenter ».

La librairie « ntdll.dll » expose un jeu de fonctions équivalent, qui consistent pour la plupart en un simple wrapper effectuant les instructions nécessaires au passage en mode noyau.

Par exemple, voici le désassemblage de la fonction ZwCreateProcess dans ntdll.dll sur un Windows XP professionnel :

```
7C91D7D2   B8 35000000      MOV EAX,35
7C91D7D7   BA 0003FE7F      MOV EDX,7FFE0300
7C91D7DC   FF12             CALL DWORD PTR DS:[EDX]
7C91D7DE   C2 2000          RETN 20
```

Avec :

```
7FFE0300     8B EB 91 7C 94 EB 91 7C 00 00 00 00 00 00 00 00  ‹ë‘|”ë‘|........
```

Et :

```
7C91EB8B   8BD4             MOV EDX,ESP
7C91EB8D   0F34             SYSENTER
```

L'API native n'est cependant pas officiellement documentée et les applications ne doivent pas l'utiliser directement afin de garantir une portabilité maximale sur les différentes versions de Windows.
Par exemple, les programmes s'exécutant dans le sous-système Win32 doivent s'appuyer sur l'API Win32, exposée notamment dans les célèbres dll kernel32.dll, user32.dll et gdi32.dll.
L'API Win32 repose bien sûr en interne sur l'API native. Par exemple la fonction CreateRemoteThread exposée par kernel32.dll appelle NtCreateThread en interne :

```
7C8104E1     56             PUSH ESI
7C8104E2     50             PUSH EAX
7C8104E3     68 FF031F00    PUSH 1F03FF
7C8104E8     8D85 50FCFFFF  LEA EAX,DWORD PTR SS:[EBP-3B0]
7C8104EE     50             PUSH EAX
7C8104EF     FF15 4414807C  CALL DWORD PTR DS:[<&ntdll.NtCreateThread>]
```

Au niveau du noyau, l'exécution est systématiquement transférée à la fonction « KiSystemService » lors d'un passage en mode noyau par l'instruction « int 2eh » et à la fonction « KiFastCallEntry » lors de l'utilisation des « Fast System Call ».

Ces fonctions ont un code relativement proche ; elles utilisent la valeur stockée dans eax comme index dans une table appelée SSDT pour récupérer l'adresse de la fonction réelle.
Il faut noter que cette description est très épurée et que le mécanisme complet fait intervenir des structures beaucoup plus complexes.

Analysons tout cela sur notre système. La table d'adresses SSDT est située en 0x804e26a8. Nous récupérons l'adresse contenue dans la 53ème case (53 == 35h).

```
kd> dd 804e26a8+35h*4

804e277c     8057b1c5 8059a849 805a4acf 8059ed5c
804e278c     80659019 80659173 8056559e 8058935c
```

Cette adresse correspond bien à la fonction NtCreateThread

```
kd> u 8057b1c5

nt!NtCreateThread:
8057b1c5 6a28            push    0x28
...
```

Le schéma suivant résume le flot d'exécution (simplifié) suivi lorsqu'une application Win32 appelle la fonction CreateThread de kernel32.dll.

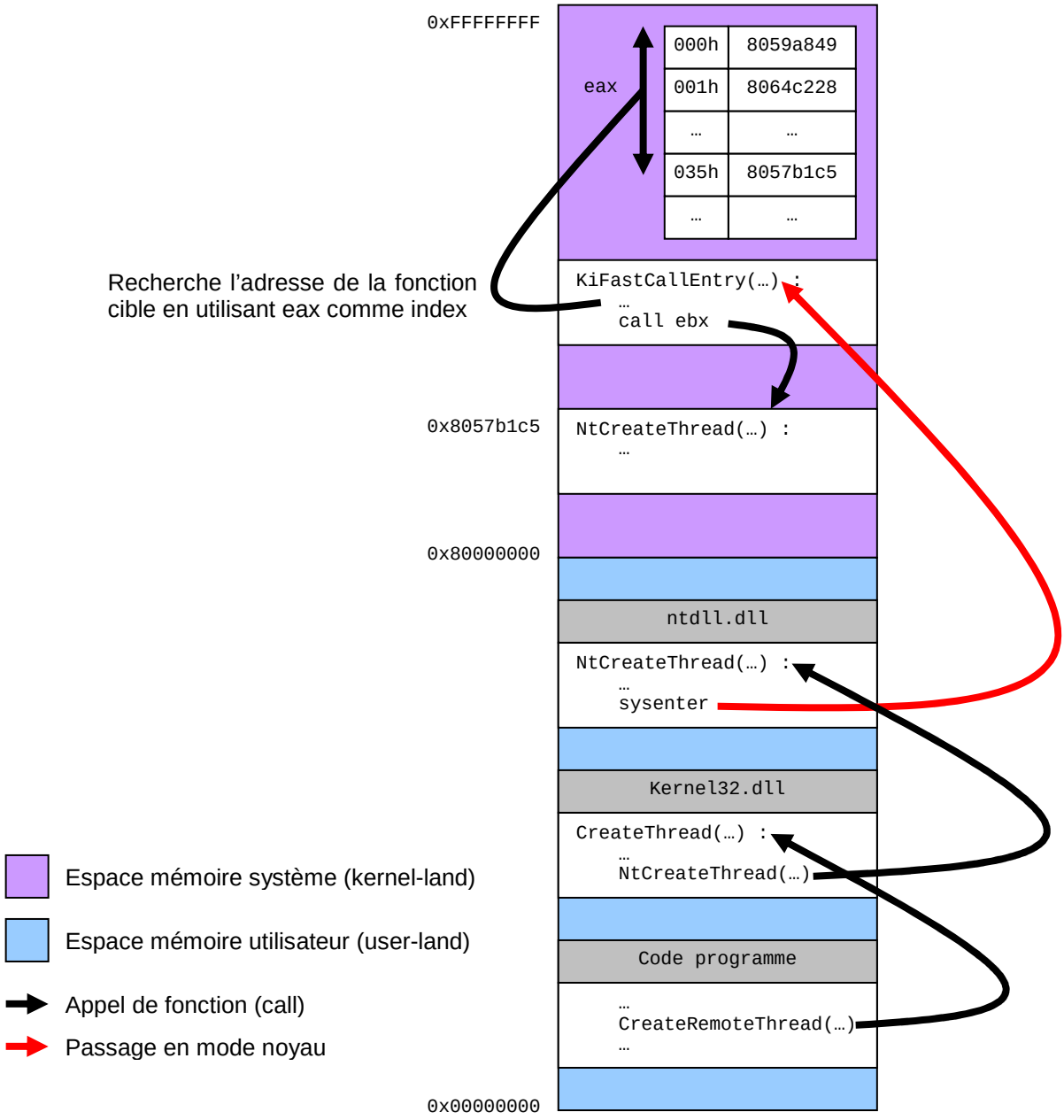

Fig. 8 Principe du passage en mode noyau lors d'un appel à CreateRemoteThread

4.4.2 Le hooking user-land par patch du header

Le hooking user-land par patch du header consiste à détourner le flot d'exécution au niveau de l'espace mémoire user-land, en patchant l'entête des fonctions à surveiller avec un saut vers le code d'analyse.

Prenons l'exemple du firewall personnel Kerio (de Sunbelt Software) qui implémente cette technique.

La fonction CreateRemoteThread commence dans la libraire kernel32.dll par les instructions suivantes :

```
.text:7C81042C      push    410h
.text:7C810431      push    7C810608
.text:7C810436      call    7C8024C6
```

Analysons maintenant le début de cette fonction dans l'espace mémoire d'un processus. Les extraits suivants ont été générés en analysant la mémoire d'une application personnelle développée pour l'occasion.

```
7C81042C > $-E9 BF009283    JMP 001304F0
7C810431   . 68 0806817C    PUSH kernel32.7C810608
7C810436   . E8 8B20FFFF    CALL kernel32.7C8024C6
7C81043B   . A1 CC36887C    MOV EAX,DWORD PTR DS:[7C8836CC]
```

La fonction commence par un jump à l'adresse 0x1304F0

```
001304F0   B8 61000000      MOV EAX,61
001304F5   68 09FFFFFF      PUSH -0F7
001304FA   8D5424 00        LEA EDX,DWORD PTR SS:[ESP]
001304FE   CD 2E            INT 2E
00130500   83C4 04          ADD ESP,4
00130503   85C0             TEST EAX,EAX
00130505   74 06            JE SHORT 0013050D
00130507   C1E8 16          SHR EAX,16
0013050A   C2 1C00          RETN 1C
0013050D   68 10040000      PUSH 410
00130512  -E9 1AFF6D7C      JMP kernel32.7C810431
```

Kerio a donc patché « à la volée » l'entête de la fonction CreateRemoteThread avec un saut vers un code d'analyse, qui transfert l'exécution au service 0x61 du noyau.
Le code retour est ensuite analysé. S'il est égal à zéro, l'exécution saute en 0x13050D où nous retrouvons le « PUSH 410 » qui a été écrasé par le « jump », puis elle revient dans la fonction CreateRemoteThread, juste après le « jump ». Sinon, nous retournons directement à l'appelant sans exécuter la fonction.

En résumé, sans hooking, le flot d'exécution est le suivant :

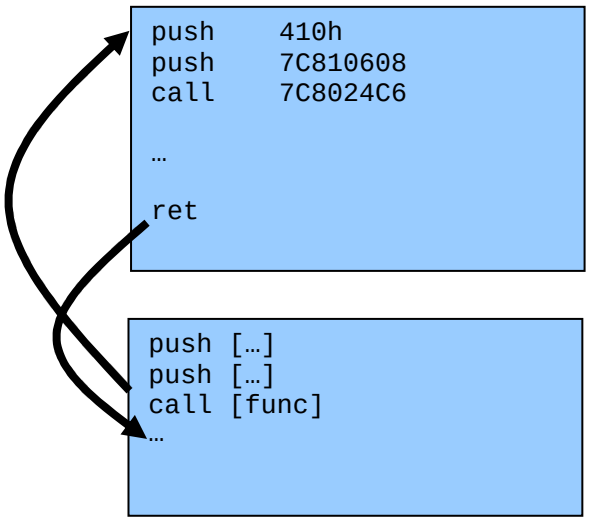

Fig. 9 Flot d'exécution lors d'un appel de fonction avant le hooking user-land de Kerio

Après le hookig, il devient :

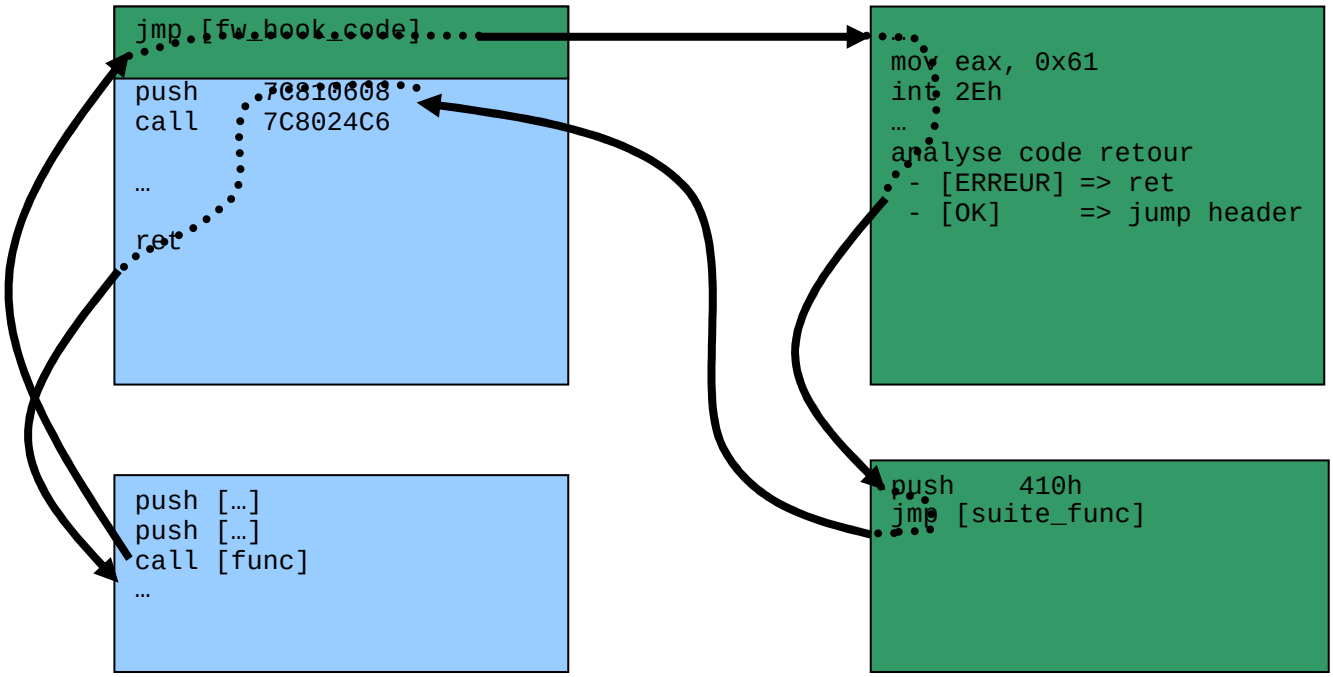

Fig. 10 Flot d'exécution lors d'un appel de fonction après le hooking user-land de Kerio

La véritable vérification est effectuée dans le noyau. En analysant rapidement la mémoire du noyau, nous constatons que la protection consiste – entre autres – à vérifier que l'adresse de retour appartient bien à un module chargé (l'exécutable lui-même ou une dll).
En effet, lors d'une injection de thread, le code injecté réside dans un des tas du processus et non dans un espace correspondant à un module chargé.
Cette protection permet donc typiquement d'empêcher l'appel de certaines fonctions critiques à partir d'un espace mémoire alloué manuellement.

4.4.3 Le hooking kernel-land par patch de la SSDT

Le hooking kernel-land par patch de la SSDT consiste à modifier certains pointeurs de la SSDT pour exécuter le code d'analyse lors de l'appel du service.
Le schéma suivant résume le nouveau flot d'exécution après la modification de l'entrée 0x35 de la table.

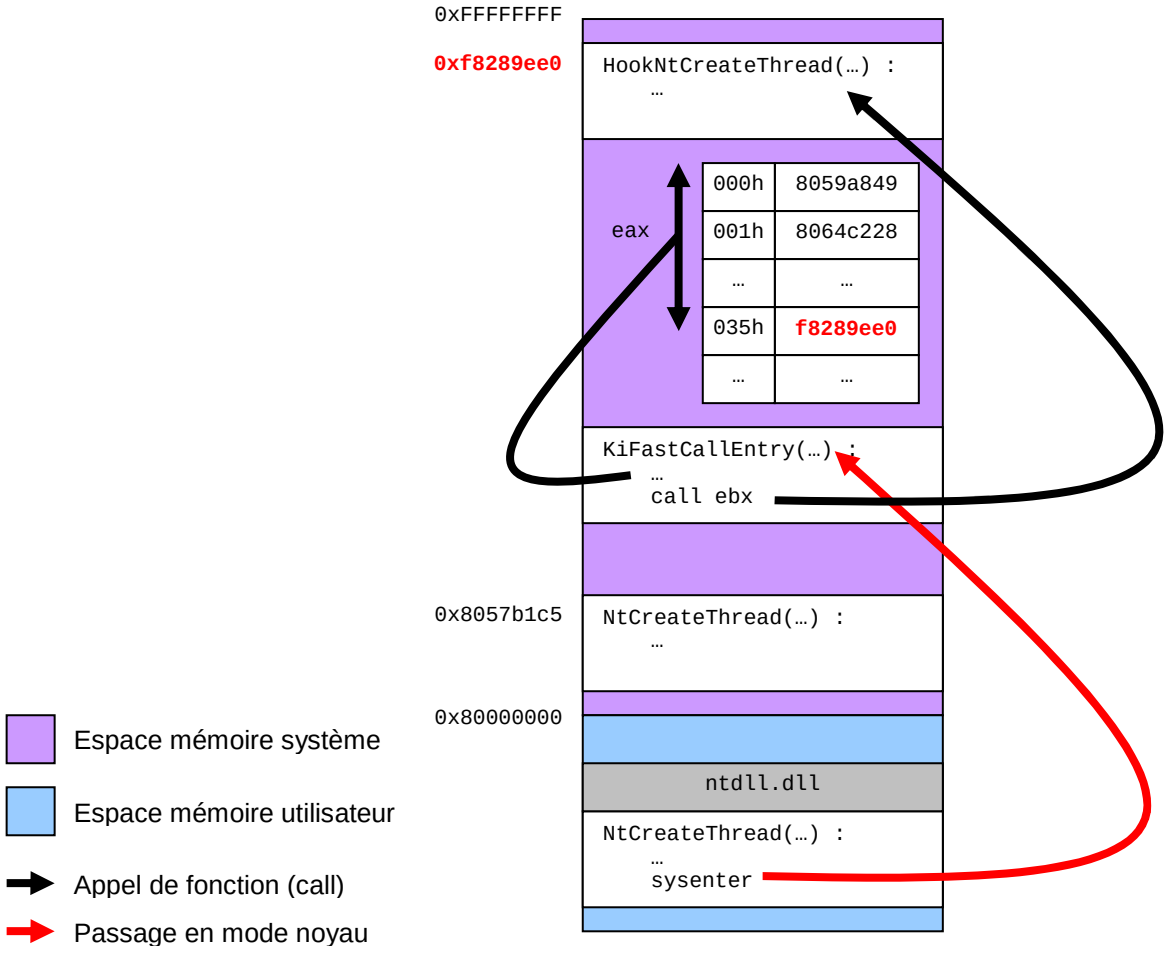

Fig. 11 Interception de l'appel à NtCreateThread via un hooking kernel-land

4.4.4 L'installation de fonctions callback

Windows offre la possibilité d'installer des fonctions callback qui seront appelées lors de l'exécution de certains événements.
A titre d'exemple, certains firewalls personnels utilisent le service PsSetLoadImageNotifyRoutine du noyau pour enregistrer une fonction callback qui sera appelée lors de la création d'un nouveau processus.
Cette fonction affiche alors (par l'intermédiaire d'un processus utilisateur) une fenêtre de confirmation à l'utilisateur.

4.4.5 Les limites de ces protections

4.4.5.1 Analyse du hooking user-land par patch du header

Cette technique de protection est de manière intrinsèque relativement limitée car elle prend place dans le même espace mémoire et possède, du point de vue du système, les mêmes droits que le code à analyser.
Reprenons l'exemple de Kerio pour illustrer quelques techniques de contournement.
Le hooking est effectué au niveau de l'API Win32 et n'interceptera donc pas les appels directs à l'API native. Le code de NtCreateThread n'a en effet pas été modifié. Placer le hooking au niveau l'API native n'apporterait pas une meilleure sécurité puisque le processus a toujours la possibilité d'effectuer lui-même le passage en mode

noyau, sans s'appuyer sur les fonctions de ntdll.dll. A noter que dans ce cas, la portabilité de la backdoor risque de diminuer considérablement car les numéros des services varient d'une version de Windows à une autre.
L'API native n'étant pas documentée, cette méthode peut s'avérer de plus relativement complexe à mettre en œuvre.

Une autre solution peut être de repatcher directement le jump du début de la fonction pour éviter l'exécution du code procédant à l'appel du noyau, principe illustré sur la figure suivante :

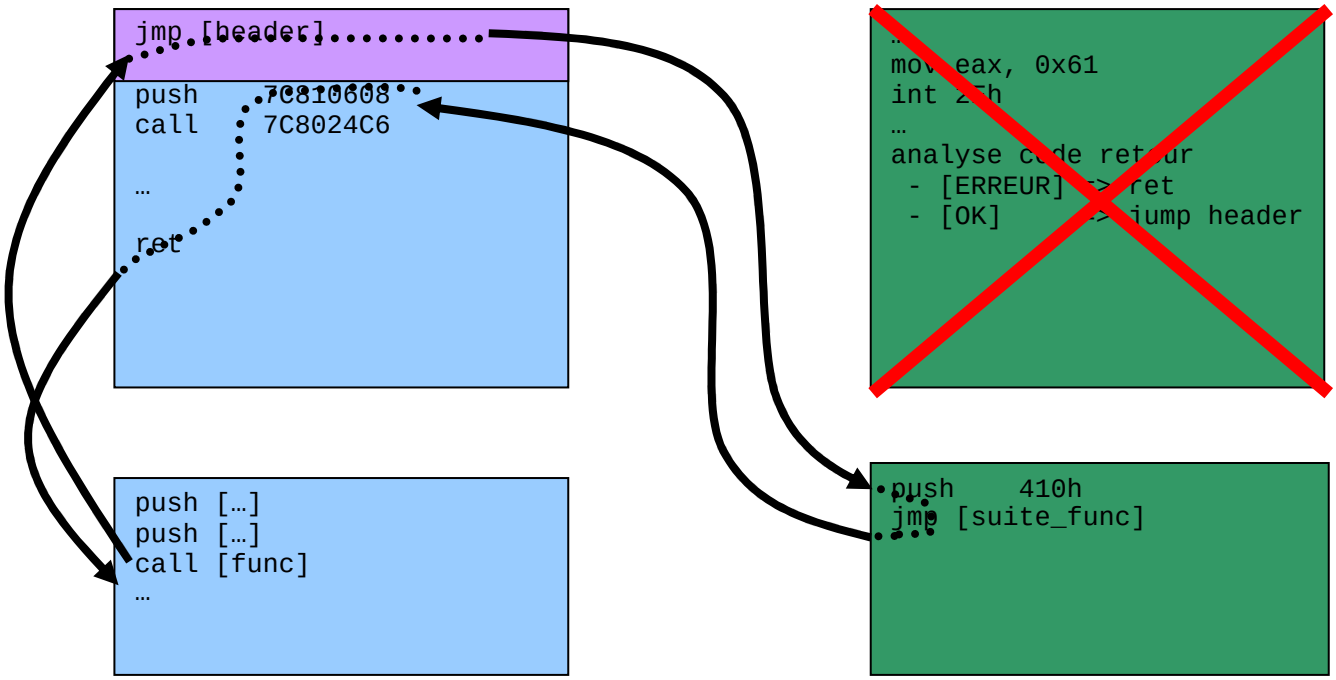

Fig. 12 Exemple de méthode de contournement de la protection hooking user-land de Kerio

D'autres firewalls personnels utilisent cette technique de hooking en user-land et peuvent être contournés par des méthodes similaires.

Etant, entre autres, destiné à produire du code injecté dans des processus, WiShMaster intègre un mécanisme appelé « FireBreaker » permettant d'ajouter du code à la fin de la fonction BuildReferences pour détecter et enlever ces éventuelles protections user-land.
FireBreaker analyse l'entête de toutes les fonctions importées afin de détecter un schéma de protection connu, puis, le cas échéant, repatche ces entêtes pour le contourner. La technique utilisée est bien sûr très dépendante de la structure de la protection ; le contournement de chaque firewall demande donc un développement spécifique.
Pour l'instant FireBreaker supporte uniquement le contournement de Kerio et de Tiny Firewall 2005. Cependant, peu de firewalls intègrent ce type de protection et l'ajout du traitement d'un nouveau firewall est relativement rapide. A titre d'exemple, il m'a fallu environ deux heures pour étudier, écrire et tester rapidement le code de contournement pour Tiny Firewall 2005.

L'ajout du code de FireBreaker augmente bien sûr la taille du shellcode produit et n'est pas forcément indispensable. Cette fonctionnalité est donc activable dans les options du projet.

4.4.5.2 Analyse du hooking kernel-land par patch de la SSDT

Contrairement au hooking user-land, le hooking kernel-land se positionne dans la partie « de confiance » du système et ne pourra donc être altéré par un code malicieux s'exécutant dans une session utilisateur restreint.
Il est important de noter que pour contrer un code malicieux s'exécutant en administrateur, le logiciel de protection devra intégrer une analyse beaucoup plus complète du système (chargement de driver, gestion des services, …), que beaucoup de produit actuels n'implémentent pas. D'où l'importance d'être loggé en utilisateur restreint.
Si cette technique est robuste en théorie, elle reste relativement délicate à mettre en œuvre et les failles vont plutôt apparaître au niveau de l'implémentation.

Reprenons l'exemple de Kerio qui utilise le hooking en kernel-land en plus de celui en user-land.
Le tableau suivant présente la liste des services hookés :

Service	Numéro du service	Adresse avant hook => après hook
NtClose	019	80566b49 => f82f1110
NtCreateFile	025	80570d48 => f82f0920
NtCreateKey	029	8056e761 => f82ecee0
NtCreateProcess	02f	805ad314 => f82eff20
NtCreateProcessEx	030	8058041a => f82efd90
NtCreateThread	035	8057b1c5 => f82f0480
NtDeleteFile	03e	805d3c07 => f82f1190
NtDeleteKey	03f	80590f78 => f82ed320
NtDeleteValueKey	041	8058e9fa => f82ed3c0
NtLoadDriver	061	805a08ec => f81469a0
NtMapViewOfSection	06c	805732fc => f8146b30
NtOpenFile	074	80570ce3 => f82f0bf0
NtOpenKey	077	80567afb => f82ed140
NtResumeThread	0ce	8057b838 => f82f0510
NtSetInformationFile	0e0	80577e2c => f82f0f00
NtSetValueKey	0f7	80574c1d => f82ed4d0
NtWriteFile	112	805780d5 => f82f0e50

La fonction NtCreateThread, appelée en interne par CreateRemoteThread est hookée. Cependant le code d'analyse ne semble pour l'instant n'effectuer aucune véritable vérification : si IRQL == 0, le service est appelé directement. Cette partie est peut-être en cours de développement.

Analysons maintenant les hooks positionnés par Tiny Firewall 2005 :

Service	Numéro du service	Adresse avant hook => après hook
NtCreateKey	029	8056e761 => f801ffa0
NtCreateSection	032	8056441b => f83a1000
NtOpenKey	077	80567afb => f801ff50
NtSetInformationProcess	0e4	8056bd05 => f83a0070

Nous constatons que plusieurs fonctions critiques ne sont pas surveillées. L'appel de CreateRemoteThread est surveillé par un hook user-land, mais une fois cette protection retirée par FireBreaker, plus rien n'empêche l'injection de thread.

Regardons maintenant le firewall personnel inclu dans la solution de Kaspersky

Service	Numéro du service	Adresse avant hook => après hook
NtClose	019	80566b49 => f81625b0
NtCreateKey	029	8056e761 => f8157280
NtCreateProcess	02f	805ad314 => f81622c0
NtCreateProcessEx	030	8058041a => f8162440
NtCreateSection	032	8056441b => f8162ee0
NtCreateSymbolicLinkObject	034	8059d4db => f8162b2e
NtCreateThread	035	8057b1c5 => f8163830
NtDeleteKey	03f	80590f78 => f8157340
NtDeleteValueKey	041	8058e9fa => f81573c0
NtDuplicateObject	044	80573ab6 => f8162710
NtEnumerateKey	047	8056ee68 => f8157450
NtEnumerateValueKey	049	8057eb28 => f8157500
NtFlushKey	04f	805da2d0 => f81575b0
NtInitializeRegistry	05c	8059f813 => f8157630
NtLoadKey	062	805aacf0 => f8157e00
NtLoadKey2	063	805aab3e => f8157650
NtNotifyChangeKey	06f	8058c141 => f81576f0
NtOpenFile	074	80570ce3 => f9e6c028
NtOpenKey	077	80567afb => f81577d0
NtOpenProcess	07a	80573c96 => f8162050
NtOpenSection	07d	8057769b => f8162d2e
NtQueryKey	0a0	8056eb71 => f8157870
NtQueryMultipleValueKey	0a1	8064c8f8 => f8157920
NtQuerySystemInformation	0ad	8057c4aa => f81634c6

NtQueryValueKey	0b1	8056b0bb => f81579d0
NtReplaceKey	0c1	8064d232 => f8157a80
NtRestoreKey	0cc	8064bd56 => f8157b10
NtResumeThread	0ce	8057b838 => f81637a0
NtSaveKey	0cf	8064bdfd => f8157ba0
NtSetContextThread	0d5	8062c143 => f8163b80
NtSetInformationFile	0e0	80577e2c => f8164270
NtSetInformationKey	0e2	8064c45b => f8157c30
NtSetInformationProcess	0e4	8056bd05 => f8166cd0
NtSetValueKey	0f7	80574c1d => f8157cd0
NtSuspendThread	0fe	805df81e => f8163750
NtTerminateProcess	101	80582c2b => f8163300
NtUnloadKey	107	8064c02b => f8157dc0
NtWriteVirtualMemory	115	8057e5e0 => f81625d0

La liste des fonctions hookées est beaucoup plus conséquente. Ce firewall ajoute même 13 nouveaux services.
La fonction NtCreateThread est cette fois sujette à une véritable analyse. L'implémentation d'une injection de thread en appelant CreateRemoteThread provoquera l'affichage d'une popup de confirmation à l'utilisateur.

Mais même dans le cas d'une implémentation relativement exhaustive comme ici, il peut exister des méthodes de contournement. Reprenons le principe de l'injection de thread : la backdoor lance une instance cachée et suspendue du navigateur par défaut via un appel à CreateProcess. Elle récupère alors un handle ayant tous les droits sur le processus via la structure PROCESS_INFORMATION, qu'elle utilise pour allouer une portion de mémoire et la remplir avec le shellcode.
A ce moment, au lieu d'appeler CreateRemoteThread, elle peut patcher le point d'entrée avec un jump vers le shellcode, puis résumer l'exécution du processus.

Aucun appel à CreateRemoteThread n'a alors été fait et l'injection ne sera pas détectée par Kaspersky. En revanche le lancement du processus « cmd.exe » avec les entrées/sorties standards redirigées sera intercepté.

Le hooking en kernel-land reste, à mon sens, la véritable solution. La difficulté provient du fait que le firewall doit à la fois surveiller un grand nombre de fonctions afin de couvrir toutes les techniques d'évasion, et dans le même temps éviter de remonter sans cesse des alertes à l'utilisateur, qui risque sinon de vite prendre le reflexe de cliquer sur « Accepter ».

4.4.5.3 Analyse de l'installation de fonctions callback

Ce type de technique est limité aux possibilités offertes par Windows. Certains évènements critiques ne pourront ainsi pas être surveillés. Pour être efficace, elle doit donc être couplée avec un hooking kernel-land, sinon, comme présenté dans le paragraphe suivant, le contournement devient possible.

4.4.5.4 Surveillance de la création de processus

Plusieurs firewalls exercent un contrôle sur la création de nouveau processus, soit en utilisant un hooking kernel-land, soit en positionnant une fonction callback au niveau Windows.
La problématique est de savoir quand afficher une popup de confirmation. Lorsque cette surveillance est activée, Kerio affiche une popup à chaque fois qu'un nouveau processus est créé. Cette implémentation génère un nombre considérable de messages, notamment lors des installations qui lancent des exécutables en cascade. L'utilisateur risque alors de rapidement prendre l'habitude d'accepter systématiquement le lancement ou même de désactiver cette fonctionnalité.
D'autres firewalls personnels, comme Look'n'Stop ont choisi de n'afficher une popup que si le programme lancé est autorisé à accéder au réseau. Cette solution permet de générer un nombre beaucoup plus faible de popups.
Cependant, si l'injection de thread est possible, cette technique peut être assez facilement contournée.
Pour cela, nous allons simplement modifier la partie « Déchiffrement injecter » pour qu'elle recherche le processus « explorer.exe » et lui injecte le shellcode « injecter+RConnect ».
L'instance cachée du navigateur par défaut sera alors lancée par « explorer.exe », opération qui sera autorisée par le firewall personnel puisqu'elle est similaire à un lancement classique.
Le processus navigateur caché va alors lancer le processus « cmd.exe ». Comme celui-ci n'est reconnu comme un programme ayant droit d'accéder au réseau, le firewall n'affichera ici non plus aucune popup.

Le schéma suivant résume ce principe :

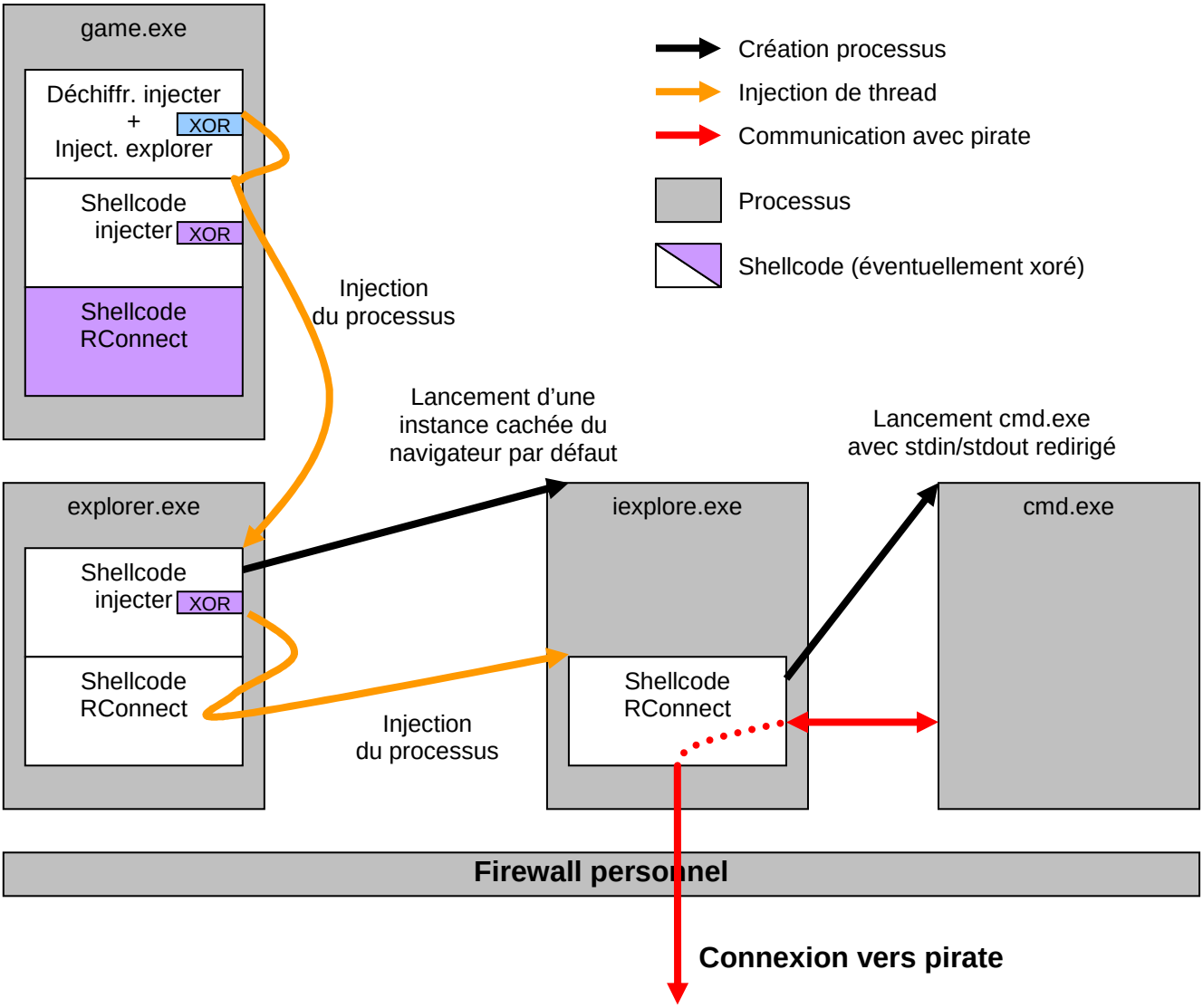

Fig. 13 Principe de l'exécution de RConnect shellcodisée avec injection dans explorer.exe

4.5 Tests de RConnect avec des firewalls personnels classiques

Le tableau suivant résume les résultats obtenus en testant quelques firewalls personnels. Je rappelle que ces résultats n'ont nullement pour objectif de dresser un classement des meilleurs firewalls personnels, étude qui mériterait des tests beaucoup plus exhaustifs, mais simplement d'évaluer la détectabilité de RConnect shellcodisée.

La plateforme de tests utilisée est constituée d'une machine Windows XP Professionnel sur laquelle s'exécute le processus serveur (netcat) et une VMware dans laquelle s'exécute un Windows XP Professionnel représentant la cible.
La backdoor utilisée est la version de RConnect procédant à l'injection du processus « explorer.exe ». Le test est effectué en lançant directement l'exécutable dans une session utilisateur restreint.

Les firewalls ont été installés avec les options par défaut. La colonne « **Modification de la configuration** » précise les éventuelles modifications apportées à cette configuration.

La colonne « Résultat RConnect » précise le résultat du test. Le « rouge » indique qu'aucune alerte n'a été remontée. Le « orange » indique qu'une popup a été affichée, mais que celle-ci contient un message qui ne correspond pas à une véritable détection d'une attaque (confirmation de lancement d'un processus par exemple)
Le « vert » indique que la backdoor a été réellement reconnue comme un code malicieux et que l'utilisateur a eu un message explicite indiquant qu'une tentative d'attaque a eu lieu.
A titre indicatif, la colonne « Résultat backdoor sans cmd.exe » indique le résultat avec une backdoor un peu plus évoluée intégrant son propre shell, et ne lançant pas de processus « cmd.exe ».

Firewall	Version	Modification de la configuration	Résultat RConnect	Résultat backdoor sans cmd.exe
Kaspersky Internet Security	6.0	Défense Proactive tout activée Pare-Feu mode apprentissage	≈ Détectée (1)	Non détectée
Tiny Firewall 2005	6.5.126		≈ Détectée (2)	Non détectée
Look'n'Stop	2.05		Non détectée	Non détectée
Kerio (installation mode avancé)	4.3	Choix « Autorisation automatique » lors du lancement d'un processus	Non détectée	Non détectée
Norton	2006	Activation surveillance composants et des programmes lancés	Non détectée	Non détectée
Comodo	2.3.4.45		Détectée	Détectée
SecurePoint FW & VPN Client	3.6.1		Non détectée	Non détectée
Sygate Personal Firewall	5.6	Des options ne sont pas activables dans la version d'évaluation.	Non détectée	Non détectée
ZoneAlarm Pro	6.1		≈ Détectée (3)	≈ Détectée (3)

(1) L'injection est possible en utilisant l'injecteur modifié patchant le début de la fonction. Seul le lancement du processus « cmd.exe » par le navigateur est détecté. Une backdoor implémentant son propre shell ne sera donc pas détectée.
(2) Seul le lancement du processus cmd.exe par le navigateur est détecté. Une backdoor implémentant son propre shell ne sera donc pas détectée.
(3) Le firewall détecte une communication avec smss.exe qui correspond au listage des processus.

Dans sa nouvelle forme, un seul firewall est capable de réellement détecter RConnect. Les autres détectent généralement le lancement du navigateur. Des techniques de social engineering pourront alors être utilisées pour ne pas éveiller les soupçons de l'utilisateur.

5 Conclusion

Ce document a permis de montrer qu'en partant d'une backdoor développée très rapidement et comportant de nombreuses limitations, il était possible d'obtenir en quelques clics via WiShMaster un outil beaucoup plus puissant, très malléable et capable de contourner la majorité des firewalls personnels.

L'utilisation de programmes comme WiShMaster permet de développer relativement rapidement des outils basés sur l'injection de thread. Les backdoors ne sont qu'un exemple ; il existe quantités d'attaques qui peuvent reposer sur cette technique : modification du comportement du navigateur pour intercepter des communications privées, vol de mot de passe, …
Dans un environnement que l'on souhaite sécurisé, il va donc devenir vraiment indispensable d'équiper les postes utilisateurs d'un logiciel de protection capable de bloquer de telles attaques en effectuant une véritable analyse comportementale des applications.

6 Références

[1] Page de WiShMaster sur mon site personnel : http://benjamin.caillat.free.fr/wishmaster.php
[2] Outil netcat pour Windows : http://www.vulnwatch.org/netcat/