Techniques de contournement des *firewalls personnels* et protections

1 Propos du document

Ce document a pour but de détailler les techniques utilisées par les *malwares* afin de contourner les *personal firewall* actuels, et de présenter les solutions proposées par le *firewall* Look'n'Stop. Cette étude est faite dans le cas d'un poste client Windows XP.

2 Théorie : Connaissances nécessaires concernant l'architecture Windows XP

2.1 Définitions

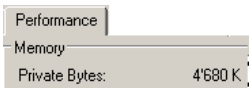
2.1.1 Processus

Un processus (*process*) est une entité logique, possèdant son propre ensemble de ressources, ayant pour but de fournir l'environnement nécessaire à l'exécution d'un programme.

Un *process* possède :

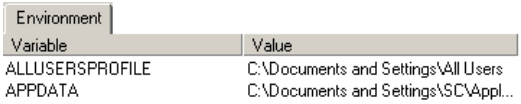
- un **PID** permettant de l'identifier

- un **espace mémoire privé**

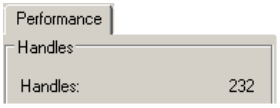


- un **code exécutable**

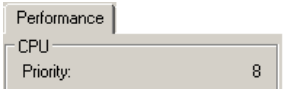
- des **variables d'environnement**



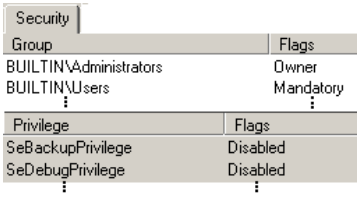
- des ***handles*** sur des ressources systèmes



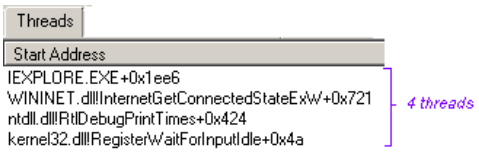
- une **classe de priorité**



- un ***access Token*** définissant son contexte de sécurité (utilisateurs et groupes associés au processus, privilèges associés au processus)



- Au minimum un ***thread***



Les informations affichées dans cette colonne sont obtenues en effectuant un double-clic sur le *process* à analyser puis en sélectionnant l'onglet correspondant.

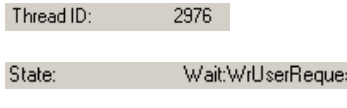
Pour plus d'informations concernant les processus, consulter [3] p.11 et [8] p. 4.

2.1.2 Thread

Un *thread* est une entité exécutant un code. Les différents *threads* d'un processus se partagent les ressources de ce processus.

Un *thread* possède :

- Un **TID** permettant de l'identifier
- Un **état** (*Ready, Standy, Running, Waiting, Transistion ou terminated*)
- Une **pile** (*stack*)
- Une **classe de priorité**



Process Explorer

Les informations affichées dans cette colonne sont obtenues en effectuant un double-clic sur le *process* à analyser puis en sélectionnant l'onglet `Threads`. Le *thread* a examiner doit être sélectionné dans la liste des *threads* s'exécutant dans le *process*.

Pour plus d'informations concernant les *threads*, consulter [3] p. 12 et [8] p. 5.

2.1.3 DLL

« Une *dll* (*Dynamic Link Librarie*) est un module contenant des fonctions et des données pouvant être utilisées par un autre module (*dll* ou processus) »
Traduction de la définition donnée par Microsoft à l'url :
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/dynamic_link_libraries.asp

Une fois la dll chargée dans l'espace mémoire d'un processus A et exécutée en tant que *thread*, les **appels effectués par cette dll seront vus comme des appels provenant de A**.

cf. p 4 [8]

2.1.3.1 Visualiser les dlls avec listdlls.exe

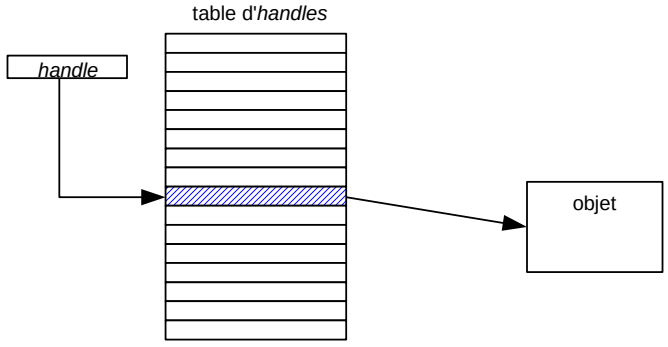
Le programme listdlls (disponible gratuitement via [10]) permet de visualiser les dlls présentes sur une machine Windows XP/2000.

Affiche toutes les dlls chargées :	`listdlls`
Affiche les dlls chargées par le processus ayant X pour PID :	`listdlls X`
Affiche les dlls chargées par le processus ayant Y pour nom :	`listdlls Y`
Affiche les processus ayant chargés la dll Z :	`listdlls -d Z`

2.1.4 Handle

Un *handle* est un entier permettant d'identifier de manière unique et d'accèder à un objet *Windows* (fenêtre, processus, icône…). Cet entier est en fait un *offset* dans une table d'*handles*.
Lorsqu'un processus créé ou accède un objet, il reçoit un *handle* représentant son accès sur cet objet.
Chaque processus possède sa propre table d'*handles*.

Pour plus d'informations concernant les handles, cf. p. 137 de [8].



2.1.4.1 Visualiser les handles avec handle.exe

Le programme *handle v 2.20* (disponible gratuitement via [10]) permet de visualiser les *handles* ouverts sur une machine Windows 2000/XP.

Affiche tous les *handles* :	`listdlls`
Affiche les *handles* appartenant au processus ayant X pour nom :	`listdlls -p X`
Affiche les *handles* sur les objets ayant Y pour nom :	`listdlls Y`

2.2 Lancement d'une application

Lorsqu'une application est lancée sur une machine Windows XP, les opérations suivantes ont lieu :

1. Ouverture de l'image (fichier .exe) à exécuter dans le processus
2. Création du processus et chargement du fichier dans l'espace mémoire du processus
3. Création du *thread* initial
4. Démarrage de l'exécution du *thread*

Il est à noter que **deux programmes** A et B (donc deux processus) **lancés par le même utilisateur posséderont des *access token* identiques et donc, des privilèges équivalents.**
Les deux processus possédant des privilèges égaux, A à les droits suffisants pour injecter du code dans B.
Cependant, les processus tels que `csrss.exe` ou `svchost.exe` ne peuvent être injectés par A car ils appartiennent à l'utilisateur SYSTEM (utilisateur possédant des droits supérieus à ceux des droits utilisateurs).
L'unique manière d'injecter de tels processus nécessite de disposer du privilège `SeDebugPrivilege` et des droits d'administrateur.

Plus d'informations concernant le cycle de vie d'un processus, p. 304 de [8].

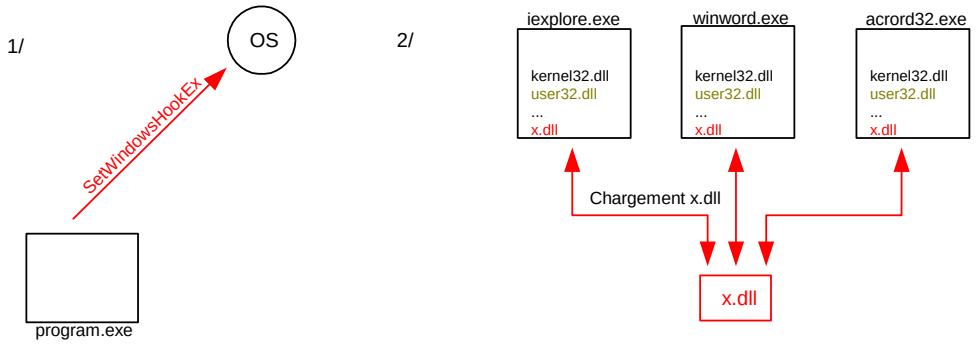
2.3 Injections de codes

Plusieurs techniques permettent d'injecter du code dans processus.
Les paragraphes § 2.3.1, § 2.3.2, § 2.3.3 ont pour but de présenter trois des techniques les plus utilisées.

2.3.1 Injection de dll via les Windows Hooks

"Un *hook* est un point particulier dans le mécanisme de gestions des messages de Windows où une application peut installer une sous-routine dans le but d'observer les messages en transit dans le système et de traîter certains types de messages avant qu'ils n'atteignent leurs cibles. "
Traduction de la définition donnée par Microsoft à l'url :
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/WinUI/WindowsUserInterface/Windowing/Hooks/AboutHooks.asp

Dans la figure ci-dessous, lorsqu'un *hook* est créé (1/) grâce à la fonction *SetWindowsHookEx* (située dans la dll user32.dll), la « sous-routine » à installer est chargée par tous les processus ayant user32.dll chargé (2/).
La fonction à installer doit se trouver dans une dll.
Lors de le création du *hook*, ce n'est pas que la fonction qui est chargée, mais la totalité de la dll.



2.3.2 Injection de dll via les fonctions CreateRemoteThread et Loadlibrary

1/ X récupére un *handle* (`hA`) sur le processus à injecter (processus A) grâce à la fonction `OpenProcess`. Le pid du processus A (PID_A) est passé en paramètre à cette fonction.

2/ X va s'allouer une zone d'espace mémoire dans le processus A afin de pouvoir y inscrire les données à injecter. Cette allocation est effectuée grâce à la fonction `VirtualAllocEx`. `PinjData` est l'adresse de début de la zone allouée dans A.

3/ X écrit les données à injecter grâce à la fonction `WriteProcessMemory`. `PinjData` spécifie à partir de quelle adresse les données doivent être écrites. `dllPath` représente les données à injecter.
Dans nôtre cas, la donnée à injecter est le chemin complet d'accès à la dll que l'on veut injecter.

4/ X va donner l'ordre à A de charger la librairie voulue (`dllPath`) grâce aux fonctions `CreateRemoteThread`, `GetProcAddress` et `loadlibrary`.

La fonction **`CreateRemoteThread`** crée un *thread* dans l'espace mémoire du processus cible.
Elle doit recevoir, entre autres, comme paramètres :

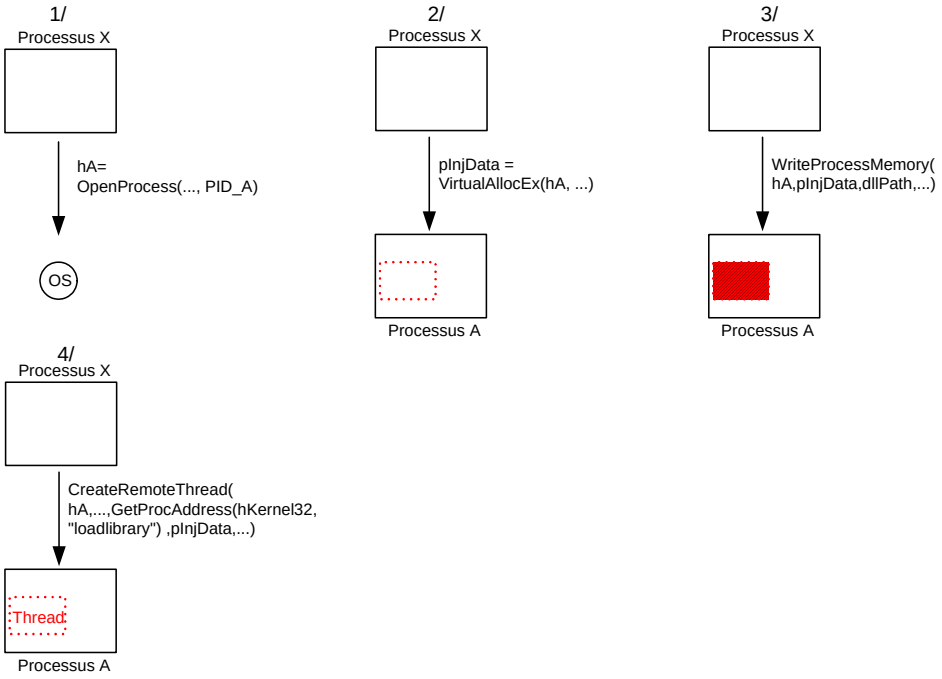
- **hProcess** : *handle* sur le processus dans lequel le *thread* doit être créé.
  Dans nôtre cas, `hProcess = hA`
- **lpStartAddress** : adresse de début de la fonction à exécuter en tant que *thread*.
  Dans nôtre cas, `lpStartAddress = GetProcAddress(hKernel32, "loadlibrary")`
- **lpParameter** : le paramètre à passer à la fonction indiquée par `lpStartAddress`.
  Dans nôtre cas, `lpParameter = pInjData`

La fonction **`GetProcAddress`** retourne l'adresse de la fonction qui lui est passée en paramètre.
Elle doit recevoir comme paramètres :

- **hModule** : *handle* sur la dll contenant la fonction à charger
- **lpProcName** : nom de la fonction à charger

La fonction **`loadlibrary`** charge la dll qui lui est passée en paramètre.
Dans nôtre cas, cette fonction étant appelée depuis `CreateRemoteThread`, le paramètre sera `pInjData`.

Une fois cette dll chargée, `CreateRemoteThread` démarre un *thread* exécutant la dll.



Pour plus d'informations concernant ces fonctions, consulter les url suivantes :
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/openprocess.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/memory/base/virtualallocex.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/debug/base/writeprocessmemory.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/createremotethread.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/getprocaddress.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/loadlibrary.asp

2.3.3 Injection de thread via WriteProcessMemory

1/ X récupére un *handle* (`hA`) sur le processus à injecter (processus A) grâce à la fonction `OpenProcess`. Le pid du processus A (PID_A) est passé en paramètre à cette fonction.

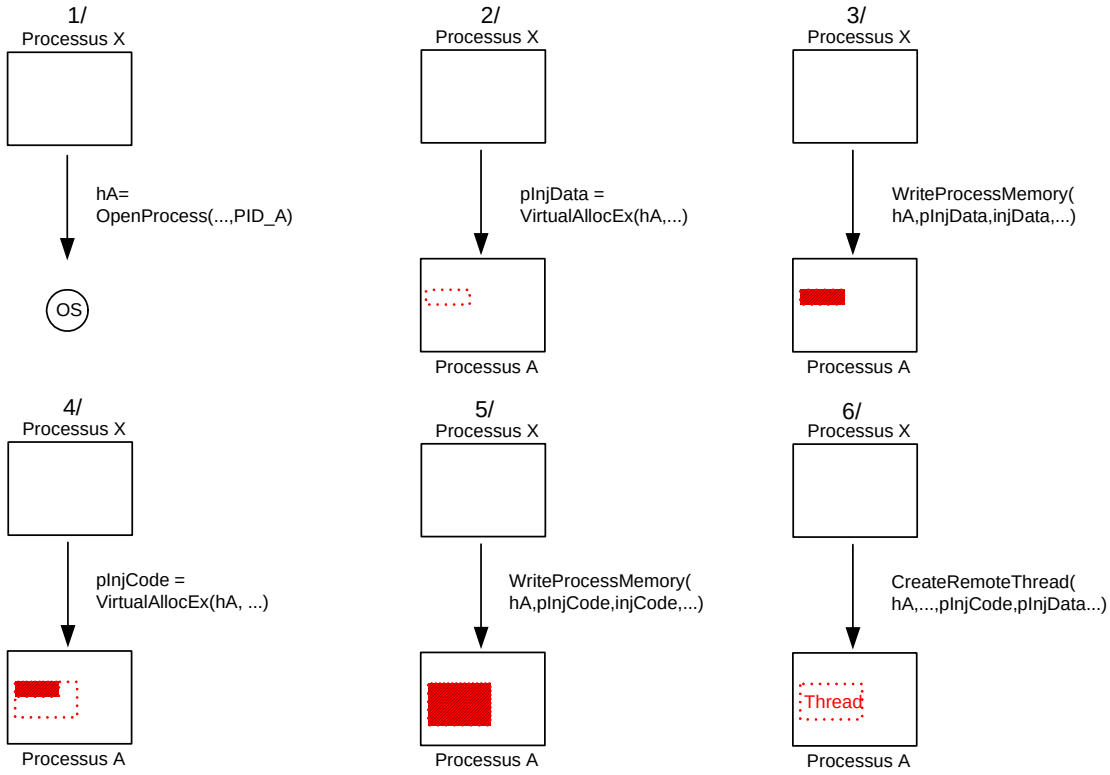
2/ X va s'allouer une zone d'espace mémoire dans le processus A afin de pouvoir y écrire les données à injecter. PinjData est l'adresse de début de la zone allouée dans A.

3/ X écrit les données à injecter grâce à la fonction `WriteProcessMemory`. `PinjData` spécifie à partir de quelle adresse les données doivent être écrites. `InjData` représentente les données à injecter.
Ces données seront passées en paramètre au code injecté lors du point 6/.

4/ X va s'allouer une zone d'espace mémoire dans le processus A afin de pouvoir y écrire le code à injecter.
`PinjCode` est l'adresse de début de la zone venant d'être allouée.

5/ X écrit dans l'espace mémoire venant d'être alloué (`pInjCode`) le code à injecter (`injCode`).

6/ X va donner l'ordre à A de créé un *thread* pour exécuter le code (`pinjCode`) grâce à la fonction `CreateRemoteThread`. Le paramètre passé au code est `injData`.

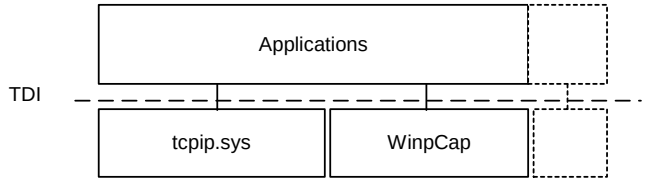


2.4 Interfaces TDI et NDIS

Lorsqu'une application désire communiquer avec une autre application en passant par une interface réseau, les messages envoyés par cette application passent obligatoirement par les interfaces TDI et NDIS.

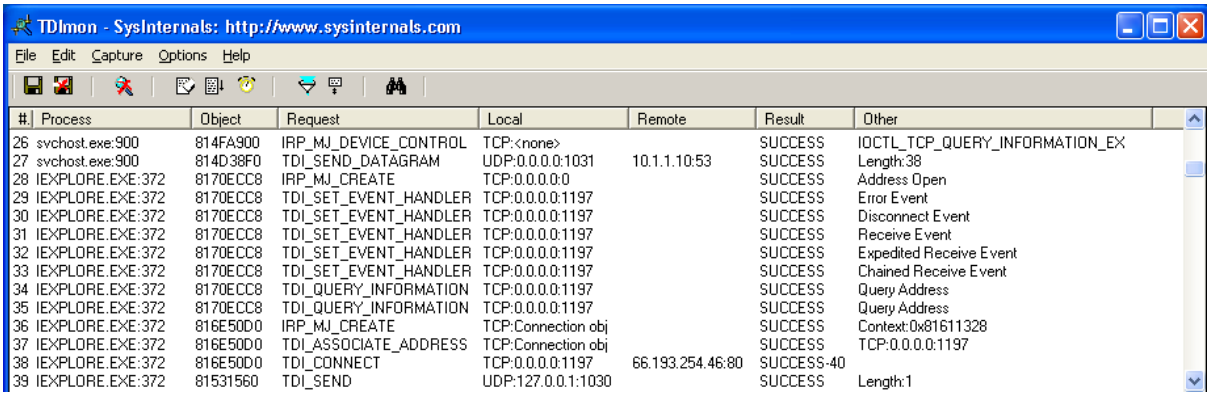
2.4.1 Interface TDI

L'interface TDI est située entre la couche applicative (processus) et les *drivers* de protocoles NDIS (par exemple, tcpip.sys par défaut sous *Windows*).
Cette interface fournit aux applications un ensemble de fonctions génériques leurs permettant d'être indépendant du protocole de transport utilisé.



2.4.1.1 Observer les flux TDI

Les flux TDI peuvent être observés grâce à l'outil gratuit TDImon (http://www.sysinternals.com).

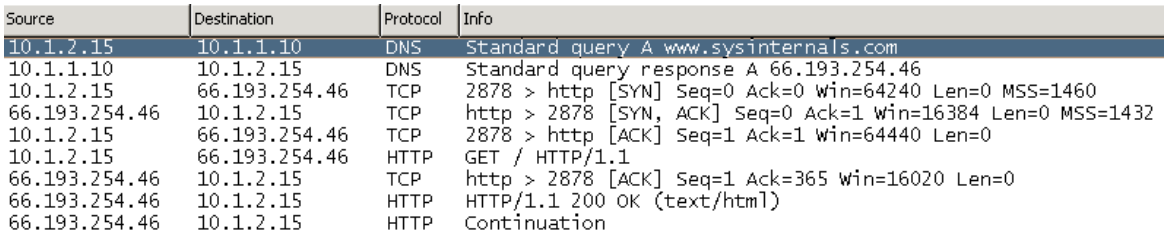


2.4.2 Interface NDIS

L'interface NDIS est située entre les *drivers* de protocoles NDIS et les *drivers* d'équipements réseaux (*drivers* Ethernet, Wifi…).
Cette interface fournit un ensemble de fonctions génériques permettant d'être indépendant du type de la carte réseau.

2.4.2.1 Observer les flux NDIS

Les flux NDIS peuvent être observés grâce à un outil comme *Ethereal*.
Ethereal (http://www.ethereal.com) est un analyseur de protocoles gratuit et très performant.
Il capture le flux en utilisant le *driver* de protocoles NDIS *WinpCap* (http://winpcap.polito.it).



3 Techniques de contournements de firewall

Les *leaktests* sont des petits programmes ayant pour but de contourner un *firewall* en exploitant une (ou plusieurs) technique particulière.
Des informations très complètes concernant les *leaktests* sont disponible sur le site Web [7].

Notes :
Dans les paragraphes suivants, lorsqu'il est fait mention de iexplore.exe, cmd.exe ou X.exe :

- iexplore.exe peut être remplacé par tout programme autorisé par le firewall
- cmd.exe peut être remplacé par tout programme n'étant pas présent dans les règles du firewall
- X.exe est le programme malicieux

3.1 Tableau récapitulatif des *leaktests*

Les techniques exploités par les tests présents dans le tableau peuvent être regroupées en trois catégorie principales :
Lancement d'applications (§ 3.3), contournement du stack TCP/IP (§ 3.4), injections de code (§ 3.5)

La colonne LNS consigne les résultats obtenus par le *firewall Look'n'Stop* 2.05.

	exploite	LNS
Leaktest	Substitution d'app.	
Tooleaky	Lancement d'app.	
Firehole	Injection de dll via *Windows hooks* + Lancement d'app.	
Yalta classical	Règles par défaut	
PCAudit	Injection de dll via *Windows hooks*	
PCAudit 2	Injection de dll via *Windows hooks*	
Copycat	Injection de *process*	
Thermite	Injection de *thread*	
Ghost	Lancement d'app. + Changement PID	
DNSTester	Service DNS de *Windows*	
Nmap	Contournement du stack TCP/IP	

Légende:
Le *leaktest* a réussi à atteindre Internet
Le *leaktest* n'a pas réussi à atteindre Internet

Note :
Aucun *leaktest* exploitant une faille de type « contournement de la pile TCP/IP *Windows* » et compatible avec *Windows 2000/XP* n'ayant pu être trouvé, le logiciel *Nmap* est utilisé pour effectuer ce test.

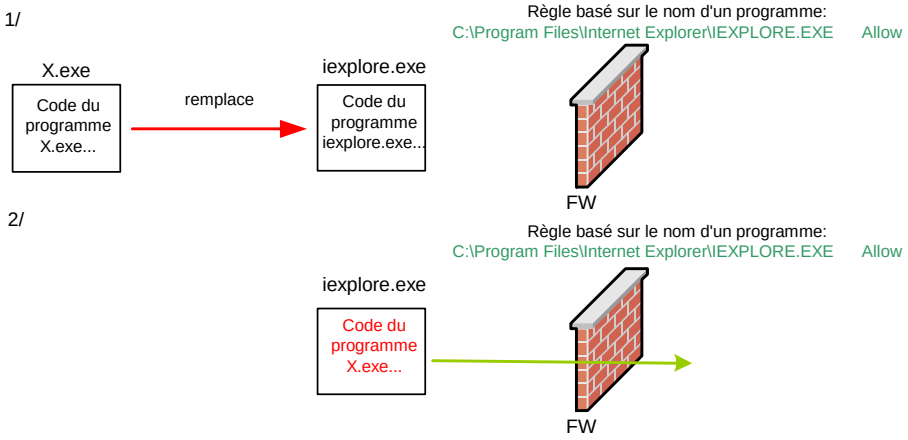
3.2 Substitution

Ce type d'attaque consiste à substituer un programme (autorisé par le *firewall*) par un programme malicieux.

3.2.1 Fonctionnement du firewall

- Le *firewall* personnel utilise des règles basées **uniquement sur le nom et le chemin d'accès des programmes** afin d'autoriser, ou non, ces programmes à traverser le *firewall*.
- Le programme iexplore.exe est autorisé à traverser le *firewall*.
- Le programme X.exe n'est pas autorisé à traverser le *firewall*.

3.2.2 Faille



Le programme X.exe va remplacer iexplore.exe et prendre pour nom A.exe (1/ sur le schéma).
X.exe ayant maintenant pour nom iexplore.exe, va être autorisé à traverser le *firewall* (conformément à la règle : C:\Program Files\Internet Explorer\IEXPLORE.EXE `Allow`)

3.2.3 Protection

Les règles du *firewall* doivent être basées, en plus du chemin d'accès et du nom du programme, sur les *hash* (MD5 en général) des programmes.
Ceci va permettre de contrôler l'intégrité des programmes et donc de s'assurer que le programme essayant de traverser le *firewall* est bien le même que celui ayant été autorisé lors de la création de la règle.
Tous les nouveaux *firewall* personnel « sérieux » possèdent cette fonction.

3.3 Lancement d'applications

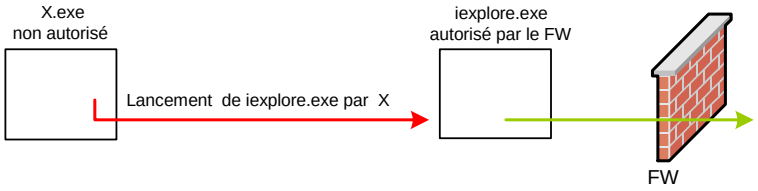
Ce type d'attaque consiste à lancer un programme (autorisé par le *firewall*) afin de lui faire exécuter une action voulue.

3.3.1 Lancement d'application simple

3.3.2 Fonctionnement du firewall

- Le programme iexplore.exe est autorisé à traverser le *firewall*
- Le programme X n'est pas autorisé à traverser le *firewall*

3.3.3 Faille



X va exécuté une instance de iexplore.exe avec des paramètres prédéfinis. (exemple : `iexplore 129.194.184.80`)

3.3.3.1 Protection

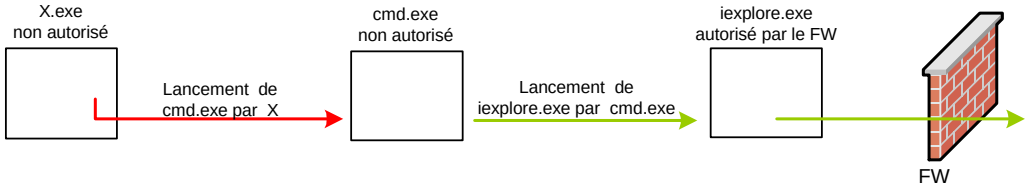
Le *firewall* doit intercepter les lancements de programme et demander à l'utilisateur (ou consulter une table spécifiant quel programme a le droit de lancer quel programme) si il désire autoriser le lancement de `iexplore.exe` par X.
Il est important de noter que dans cette proposition de protection, le ***firewall* ne contrôle que le lancement des programmes présents dans ses règles** (cf. 3.3.4).

3.3.4 Double lancement d'application

3.3.4.1 Fonctionnement du firewall

- Le programme A est autorisé à traverser le firewall
- Les programmes B et C ne sont présents dans aucune règle du firewall. Il ne sont donc pas autorisés à traverser le firewall (principe « *White List* »)
- Le *firewall* **ne contrôle que les lancements des programmes présents dans les règles** du firewall

3.3.4.2 Faille



X.exe va exécuter une instance de cmd.exe qui va à son tour exécuter une instance de iexplore.exe avec des paramètres prédéfinis.
Le contrôle de lancement de programme n'étant effectué que pour les applications présentes dans les règles du *firewall*, le lancement de cmd.exe par X.exe ne sera pas intercepté.
Le firewall ne verra que le lancement de iexplore.exe par cmd.exe.

Note : En utilisant uniquement une technique de lancement d'application (« classique » ou pas), X est limité aux fonctionnalités du programme A.

3.3.4.3 Protection

Le firewall doit contrôler le lancement de toutes les applications (et non pas seulement le lancement des applications autorisées à traverser le *firewall*).

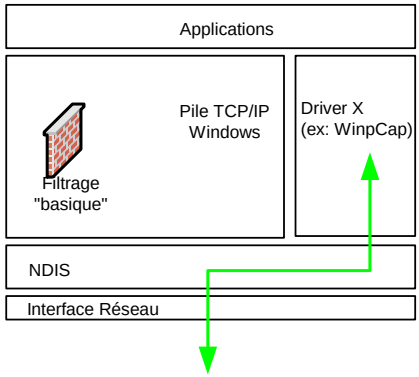
3.4 Contournement du stack TCP/IP

Lorsqu'un programme doit envoyer des paquets sur une interface réseau, il utilise généralement la pile TCP/IP de *Windows*. Le programme est cependant libre de choisir d'utiliser un autre *driver* déjà installé sur la machine ou même d'installer son propre *driver* réseau afin de disposer de son propre canal de communication vers l'interface réseau.

3.4.1 Fonctionnement du firewall

- Le filtrage réseau est effectué au niveau de la pile TCP/IP de *Windows*

3.4.2 Faille



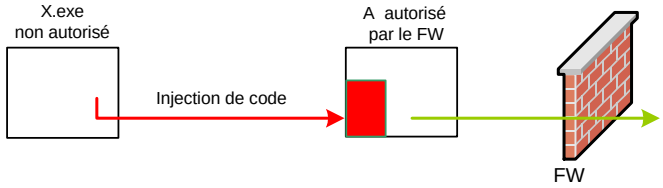
Plutôt que d'utiliser la pile TCP/IP de *Windows*, X.exe va utiliser le *driver* X afin d'envoyer les données qu'il a à transmettre.
Le driver X permettant d'accèder à une couche plus basse que celles de la pile TCP/IP (où à lieu le filtrage), le firewall n'a pas accès aux données ainsi transmises et ne peut donc pas les bloquer.

Pour plus d'information concernant le contournement de la pile TCP/IP, cf. [9].

3.4.3 Protection

Il est nécessaire d'effectuer un filtrage au niveau NDIS ou de bloquer tous les paquets n'ayant pas été généré par la pile TCP/IP de Windows.
Les *firewalls* de « qualité » implémentent désormais tous un filtrage au niveau NDIS (*Kerio, TinyPersonalFirewall, Look'N'Stop…*)

3.5 Injections de codes



L'injection de code peut avoir lieu grâce à diverses techniques. Ces techniques vont être détaillées dans les paragraphes suivants.

3.5.1 Injection de dll

3.5.1.1 Fonctionnement du firewall

- Le programme iexplore.exe est autorisé à traverser le firewall
- Le programme X n'est pas autorisé à traverser le firewall

3.5.1.2 Faille

X va injecter dans le processus iexplore.exe la dll contenant les fonctions à exécuter. Ces fonctions étant réalisées par un programme autorisé, le firewall est contourné.
Le chargement de la dll peu être effectuée de plusieurs manières :

- Via *Windows Hooks* (§ 2.3.1, § 1 de [5])
- Via la fonction *LoadLibrary* (§ 2.3.2, § 2 de [5])

3.5.1.3 Protection

Le *firewall* doit posséder une liste des dll à autoriser.

3.5.2 Injection de thread via la fonction WriteProcessMemory

3.5.2.1 Fonctionnement du firewall

- Le programme iexplore.exe est autorisé à traverser le firewall
- Le programme X n'est pas autorisé à traverser le firewall

3.5.2.2 Faille

X va injecter dans le processus iexplore.exe le code à lui faire exécuter. Ce code étant réalisé par un programme autorisé, le *firewall* est contourné.

Le mécanisme permettant de réaliser cette injection de code est décrit au paragraphe § 2.3.3 de ce document , § 3 de [5], p 52 de [4].

3.5.2.3 Protection

Le firewall doit empêcher qu'un processus A puisse écrire dans l'espace mémoire d'un processus B.

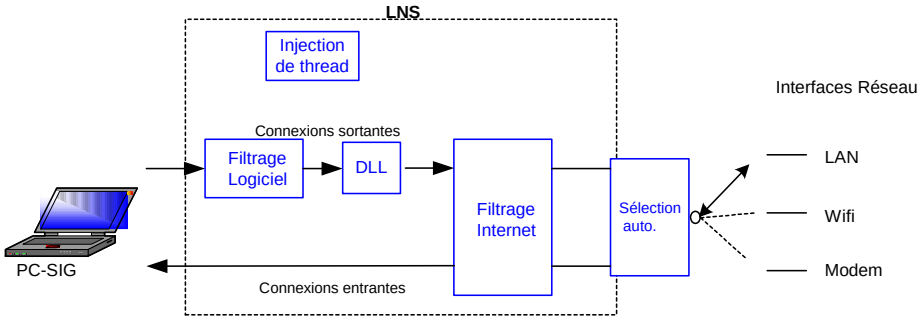
En effet, sur un système Windows XP, si un processus A est démarré par l'utilisateur Z, alors A aura les droits suffisants pour injecter du code et démarrer un *thread* dans tous les processus ayant des privilèges (cf. *access token* dans § 2.2) inférieurs ou égaux aux siens. Tous les processus lancés par Z ayant les mêmes privilèges, sont susceptibles de subir une injection.

4 Protections

4.1 Look'n'Stop

Look'n'Stop (abrégé LNS) est un *firewall* personnel permettant d'obtenir un des plus haut niveau de sécurité (cf [7]).
Cette sécurité optimale (bien que non exhaustive) est obtenue grâce aux modules détaillés dans les paragraphes suivants.

4.1.1 Architecture de Look'n'Stop

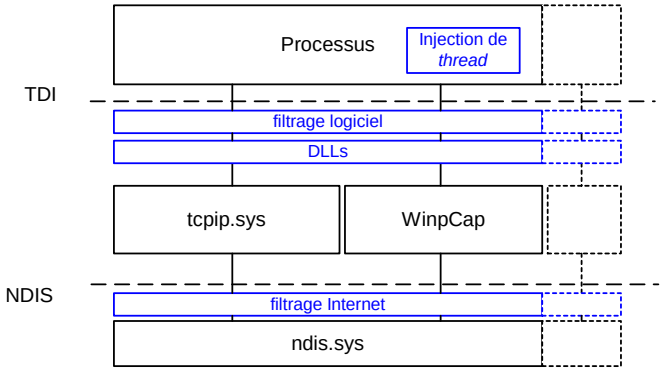


L'architecture fonctionnelle de *LNS* est composée des modules suivants :

- Le ***`filtrage logiciel`***
- Le ***`filtrage Internet`*** **: appliqué à l'interface sélectionnée** dans le cadre `Interfaces réseau`. **Une seule interface peut être surveillée à la fois**
- La ***`sélection automatique`*** de l'interface **réseau : choisit la première interface à obtenir une adresse IP**
- **La** *`détection des dlls`*
- **La** `détection d'injection de` *`thread`*

Chacun de ces modules peut être désactivé.

4.1.1.1 Modules de LNS dans le modèle en couche

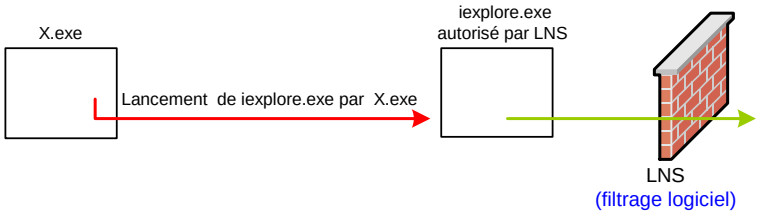


Les modules représentés en bleus correspondent aux modules de LNS.

4.1.2 Module filtrage logiciel

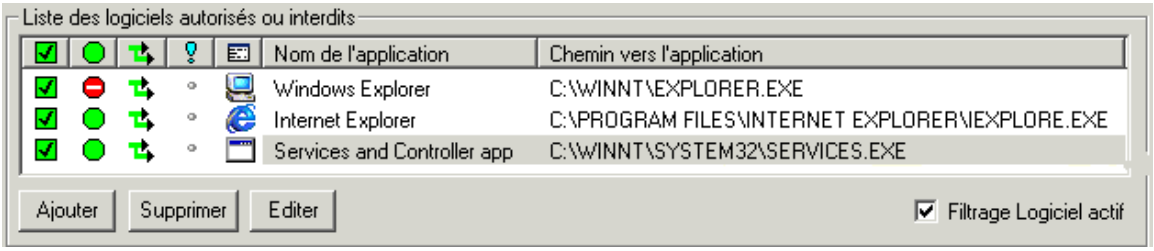
Le `filtrage Logiciel` effectué par *LNS* permet de contrôler les applications connectées à Internet en effectuant un filtrage au niveau TDI (§ 4.1.1.1).

Il permet également de détecter le lancement d'un programme présent dans la liste du `filtrage logiciel`, par un autre programme.
Cette fonctionnalité permet de contrer une partie des mécanismes de contournement des *firewalls* utilisés par les *malwares* (les *leaktests firehole* et *Tooleaky* sont contrés grâce à cette fonction).



4.1.2.1 Interface de gestion du filtrage logiciel

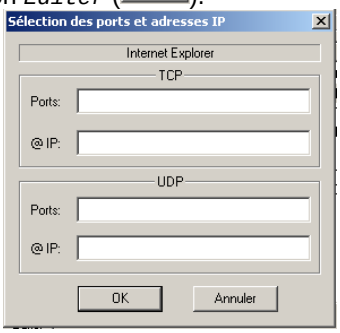
L'interface suivante permet d'activer, ou de désactiver, le filtrage logiciel et de spécifier les autorisations des applications.



4.1.2.2 Limiter une application à certains ports et certaines adresses IP

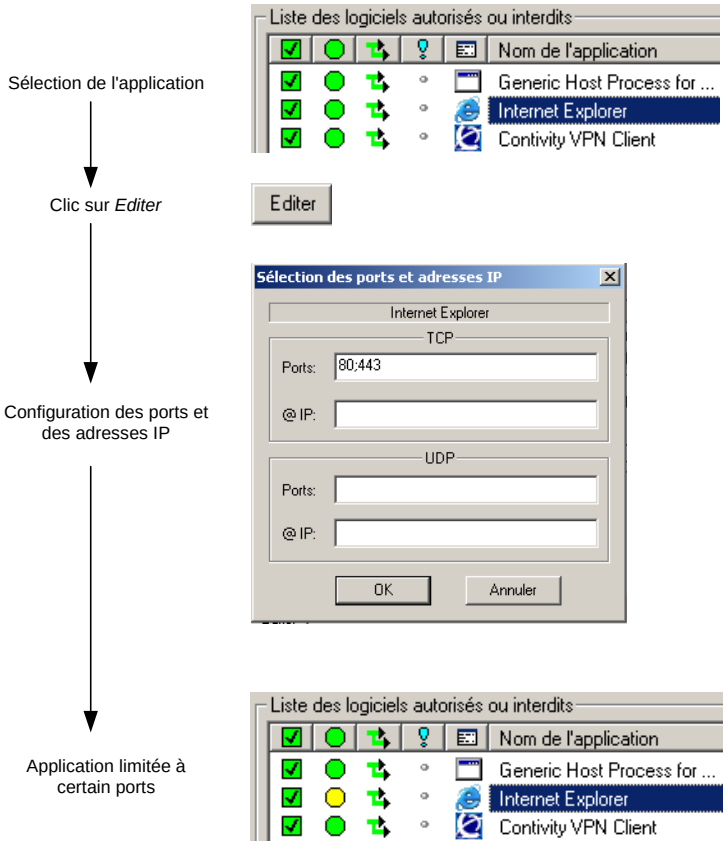
Cette fonction permet de restreindre le fonctionnement de l'application à certains ports (TCP ou UDP) et à certaines adresses IP. Elle est accessible en sélectionnant l'application à limiter dans l'onglet *`filtrage logiciel`* puis en cliquant sur le bouton *`Editer`* (

).



Si plusieurs ports ou adresses IP doivent être spécifiés dans un même champ, il faut les séparer à l'aide d'un « ; ».

4.1.2.2.1 *Exemple avec Internet Explorer :*



4.1.2.3 Lancement d'applications

La colonne  permet de spécifier si :

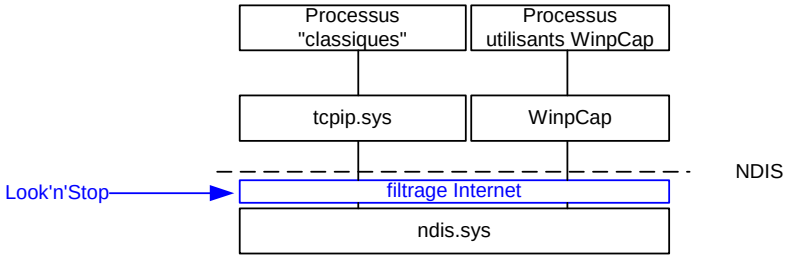
-  : l'application a le droit de lancer d'autres applications
-  : l'application n'a pas le droit de lancer d'autres applications

4.1.3 Module filtrage Internet

Le `filtrage Internet` effectué par *LNS* permet de filtrer les paquets au niveau NDIS.

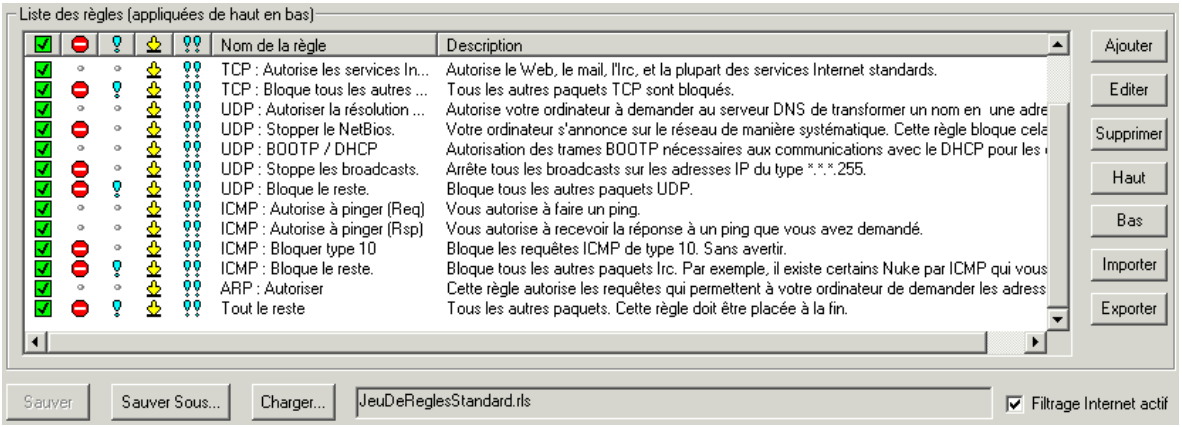
Ce filtrage permet de contrôler tout flux passant par l'interface NDIS comme :

- les flux entrants
- le flux « classique » issu de tcpip.sys (pile TCP/IP de *Windows*)
- les flux utilisant une pile autre que la pile TCP/IP de Windows (exemple : *WinpCap*, [4])



Les failles du type « contournement de la pile TCP/IP de *Windows* » sont ainsi contrées.

4.1.3.1 Interface de gestion du filtrage Internet



4.1.3.2 Lier une règle à une application

Afin d'augmenter la sécurité du poste client, toute règle du *`filtrage Internet`* peut être activée seulement lorsqu'une application X est connectée à Internet grâce au bouton *`Applications…`*
Les applications connectées à une interface réseau apparaissent dans le cadre *`Liste des logiciels actifs`* de l'onglet *`Filtrage Logiciel`*

 : Règle active
 : Règle inactive car aucune application active n'est présente dans les applications liées à la règle

4.1.3.3 Flèche jaune bloquante

La colonne  permet de spécifier si :

-  : *LNS* sort de la *`liste des règles`* `lorsque la règle est vraie`
- : *LNS* doit examiner les règles suivantes

Le fonctionnement classique d'un *firewall* correspond à placer  pour toutes les règles.

4.1.3.4 Journalisation des exécutions de règles

La colonne  permet de spécifier si l'exécution d'une règle doit être :

-  : inscrite dans le *Journal*
- : ne pas être inscrite dans le *Journal*

4.1.3.5 Alertes

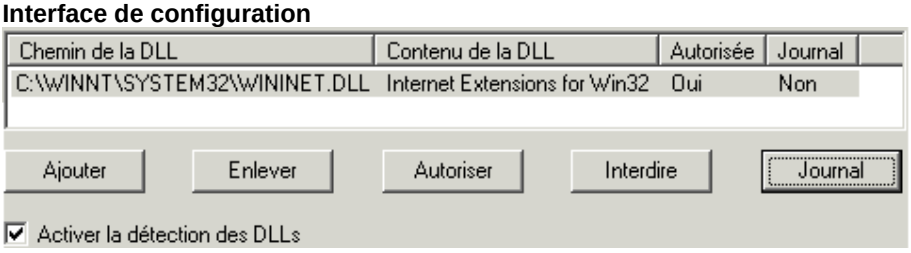
La colonne  permet de spécifier si l'exécution d'une règle :

-  : génère une alerte sonore
-  : génère une boîte de dialogue d'alerte
- : ne génère rien

4.1.4 Module de détection des DLLs

Ce module permet de détecter les dlls tentant de se connecter à une interface réseau.

4.1.4.1 Interface de configuration



4.1.5 Module Contrôler l'injection de thread

Ce module permet de détecter qu'un *thread* a été injecté dans un processus et qu'une application utilise ce *thread* pour se connecter à une interface réseau.
Thermite est bloqué grâce à ce module.

4.1.5.1 Interface de configuration

5 Outils

5.1.1 Process Explorer

Permet de : visualiser les informations concernant les processus, les *threads*, les dlls et les *handles* d'un système *Windows XP/2000*.

Fonctionnement : GUI

5.1.1.1 Les deux fenêtres de Process Explorer

La fenêtre haute

Affiche les **processus actifs** et les informations relatives à ces processus
Les processus indentés vers la droite sont les process fils

Exemple : IEXPLORE.EXE Processus parent
AcroRd32.exe Processus enfant

Un double-clic sur un *process* permet d'afficher d'autres informations concernant, notamment :

- Les ***threads*** s'exécutant dans le processus ainsi que l'état du *thread*, sa pile, son adresse de départ, sa priorité, le nombre de changement de contexte…
- Les variables d'environnement contenues par le processus
- Les services enregistrés dans le processus

La fenêtre basse

1- Affiche les **dll** et les informations concernant les dll chargées par le process sélectionné dans la fenêtre haute

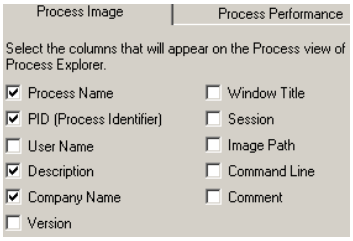
2- Affiche les ***handles*** sur les ressources de l'OS ouvert par le process sélectionnée dans la fenêtre haute

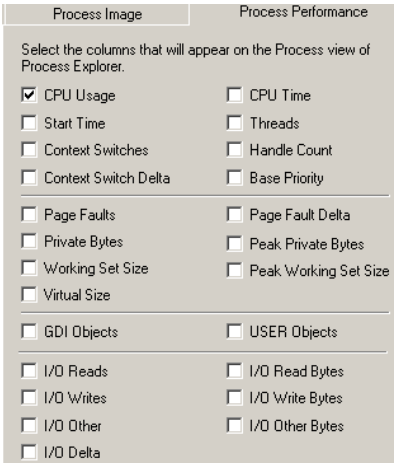
Sélection du mode de la fenêtre basse : ***View*** > ***Lower Pane View***>

5.1.1.2 Sélection des colonnes d'informations à afficher

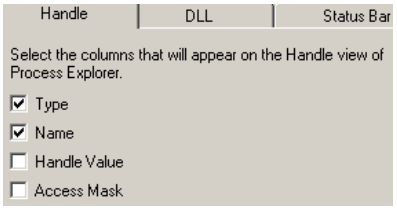
Les colonnes affichées dans les fenêtre haute et basse ainsi que les informations affichés dans la barre d'état de Process Explorer sont paramétrables : **View > Select Columns…>**

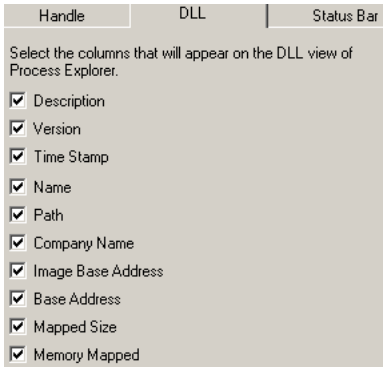
Informations disponibles concernant les processus :

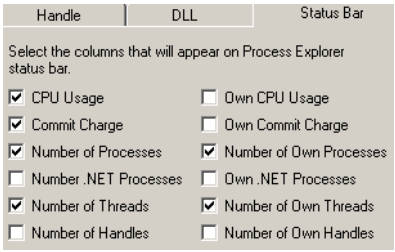




Informations concernant les *handles*, les dlls et la barre d'état :







5.1.1.3 Fonction de recherche

La fonction de recherche de Process Explorer possède deux modes :

1- Recherche d'handle : > ***Find*** > ***Find Handle…***
Permet de visualiser quels process utilisent la clé de registre X, le fichier Y etc…

2- Recherche de dll : > ***Find*** > ***Find DLL…***
Permet de visualiser quels process ont chargés la dll X (équivalent à `Tasklist /M` – cf. § 5.1.3 -)

Disponible via : [10]

5.1.2 TaskManager

Permet de : Visualiser les noms des **processus** s'exécutant sur la machine locale, le nombre de *threads* dans un processus, le nombre d'handle possédé par un processus, la taille de l'espace mémoire utilisé par le processus, le niveau de priorité du processus (Idle, Normal, High,Realtime…)…

Fonctionnement : GUI
> Ctrl+Shift+Esc > Onglet *Processes* > View > Select Column

Disponible via : Par défaut dans Windows

5.1.3 Tasklist

Permet de : Afficher des informations sur les **processus** (PID, taille en mémoire, *services*, *dll* chargées…) d'une machine locale ou distante

Fonctionnement : Ligne de commande

Tasklist / ?	Affiche les informations concernant l'utilitaire Tasklist
Tasklist /SVC	Affiche les services présents dans les process
Tasklist /M [Module]	Affiche les process ayant la dll [Module] chargée
Tasklist /M	Affiche les process ainsi que les dll chargées dans chacun de ces process

Disponible via : Par défaut dans Windows XP (sous Windows 2000, cet utilitaire à pour nom tlist. Il faut l'extraire du fichier Support.cab présent sur le CD d'installation de Windows 2000)

6 Conclusion

Les *firewalls* personnels ne proposant pas des fonctionnalités tels que le contrôle de DLLs, le blocage d'injection de *threads* ou encore la protection contre le contournement de la pile TCP/IP n'offrent aucune protection contre les *malwares* utilisant des techniques « récentes ».

Cependant, les failles de type injections de codes se situant au niveau de l'OS et non du réseau, il pourrait paraître légitime que les éditeurs de *firewall* personnel estime que la gestion de ces failles n'est pas de leur ressort. Des outils tels que [11] *ProcessGuard* (Windows) ou [12] *Systrace* (Linux) sont spécialisés dans la gestion des droits inter-processus et proposent un grand nombre de fonctionnalités permettant de bloquer les *malwares* utilisant l'injection de code, l'installation de *kernel-mode driver* etc...

LNS, quant à lui, offre une excellente protection si tout ces modules sont activés. Il était même le *firewall* permettant de parer le plus efficacement les *leaktests* lors de la période de test (Mai 2004). Cependant, les éditeurs concurrents rattrapent leur retard et intègrent de plus en plus souvent le même type de protection.

7 Sources

- Targeted Windows OS and Application Host Integrity and Auditing [1]
  http://home.nycap.rr.com/ practical/Host%20Integrity%20and%20Auditing.pdf
- "Processes and Threads functions" MSDN Microsoft [2]
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/process_and_thread_functions.asp
- Windows Kernel Overview [3]
  http://www.i.u-tokyo.ac.jp/ss/lecture/new-documents/Lectures/00-WindowsKernelOverview/WindowsKernelOverview.pdf
- "Cheval de troie furtif sous Windows : mécanisme d'injection de code" [4]
  Magazine MISC n°10, p. 50
- "Three ways to inject your code into another process" [5]
  http://www.codeproject.com/threads/winspy.asp
- "Advanced communication techniques of remote access Trojan horses" [6]
  http://www.giac.org/practical/GSEC/Candid_Wueest_GSEC.pdf
- Site Web : "Firewall Leak Tester" [7]
  http://www.firewallleaktester.com
- "Inside Microsoft Windows 2000 – Third Edition" [8]
  David A. Solomon & Mark E. Russinovich, ISBN 0-7356-0588-2
- "Pénétrations de réseaux et backdoors furtives" [9]
  Linux Magazine HS n°12, p. 62
- SysInternals Freeware [10]
  http://www.sysinternals.com
- Site de l'éditeur du logiciel *ProcessGuard* [11]
  http://www.diamondcs.com.au/processguard/
- Site des concepteurs du logiciel *Systrace* [12]
  http://www.citi.umich.edu/u/provos/systrace/