Ghorg0re/3ey

Démonstration pratique

de l'utilisation de Process Explorer

Auteur	Benjamin CAILLAT
Contact	benjamin.caillat@voila.fr
Date	22/07/2004
Version	1.00

Gestion des versions

Version	Date	Commentaires
1.00	29/07/2004	Rédaction d'origine

Contact : benjamin.caillat@voila.fr

Résumé

Ce document représente un guide d'utilisation de Process Explorer. Il consiste essentiellement en une illustration basée sur l'étude du programme ZoneAlarm. L'objectif principal de ce document est donc de présenter les fonctionnalités de cet outil, et non de reverser ZoneAlarm.

Introduction

L'utilité du reverse engineering n'est plus à prouver. Les derniers sceptiques pourront se reporter au numéro 14 de MISC, consacré en grande partie à ce sujet.
Pour étudier le fonctionnement précis d'une partie d'un exécutable, par exemple l'implémentation d'un algorithme de cryptage, il est évident que vous allez devoir plonger au cœur du code assembleur. Il existe alors de nombreux outils, soit des désassembleur (notamment IDA), soit des débuggeurs.
Cependant, il se peut que vous n'ayez pas besoin de ce niveau de détail, ou que vous vouliez commencer pas une vue générale du fonctionnement d'un exécutable.
Par exemple, il serait utile de savoir quelles fonctions exportées par les DLL sont utilisées et avec quels paramètres. C'est là qu'intervient Process Explorer.
Process Explorer n'est donc absolument pas un nouveau débugger ou un nouveau désasembleur. C'est plutôt un nouveau type d'outil.
Le principe est simple : Il « hook » toutes les fonctions que vous désirez dans un processus, puis logge dans un fichier la valeur des paramètres avant et après l'appel, ainsi que l'adresse de retour et le code de retour.

Trêve de discours, une petite démo valant une grande explication, passons à la pratique.

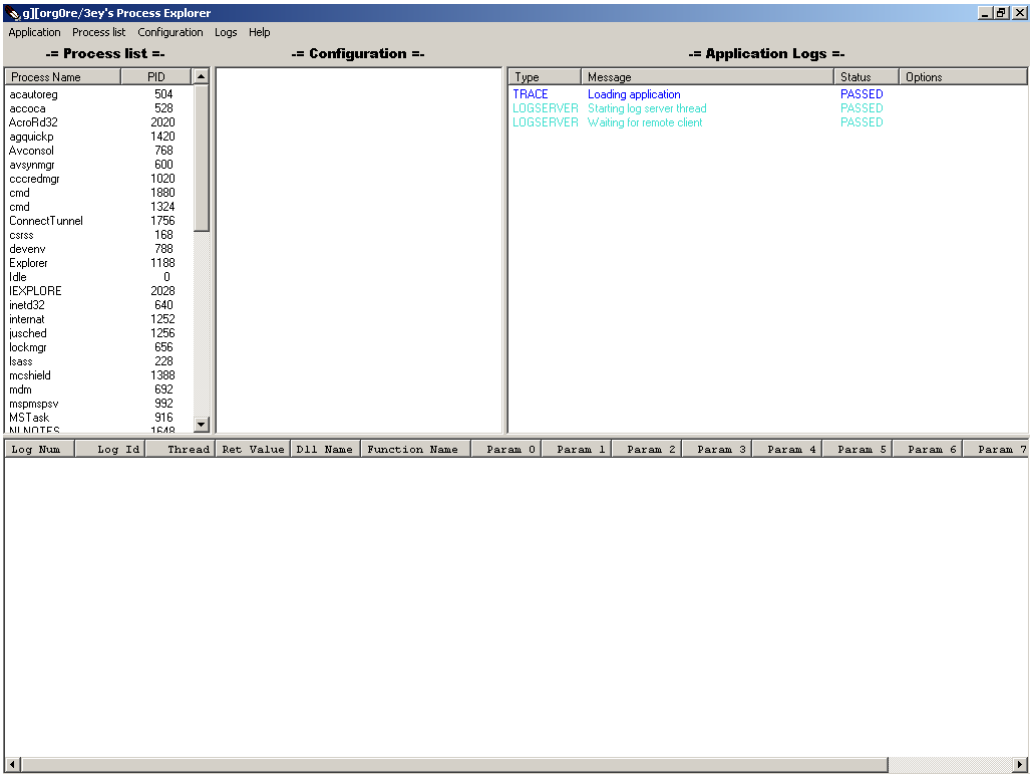
Exemple pratique : Analyse de ZoneAlarm

Le processus que nous allons analyser sera ZoneAlarm. Pourquoi ce choix ? tout d'abord parce que ma passion étant la sécurité informatique, je voulais mieux connaître le fonctionnement de cet outil, qui est actuellement un des firewall personnels le plus utilisé. Ensuite par ce que en tant que produit de sécurité, on peut s'attendre à ce qu'il présente un certain nombre de protections.

Premier contact

L'interface Process Explorer

La première étape est de lancer Process Explorer. Vous obtenez alors la fenêtre suivante :



L'écran est découpé en plusieurs zones :

- Process List : Donne la liste des processus en ce moment sur le système

- Configuration : Permet de configurer quelles fonctions vous souhaitez hooker
- Application Log : Affiche des logs « en temps réel ». Ces logs tracent le fonctionnement de Process Explorer.
- Remote Log (en bas) : Affiche les traces générées par le hooking. Attention, celle-ci sont stockées dans un fichier ouvert par le processus analysé. Pour accéder à ce fichier, vous devez donc commencer par terminer le processus analysé, seul moyen de lui faire relâcher le handle.

Récupération de la liste des modules

Injection directe dans Notepad

Nous allons commencer par tester cette fonctionnalité sur Notepad :

- Lancez un Notepad
- Rafraîchir la liste des processus (Cliquez sur la fenêtre « Process list » et appuyez sur F5).
- Sélectionnez le processus notepad.exe
- Appuyez sur Ctrl+m pour obtenir la liste des modules.
- Une messageBox vous demande le temps d'attente, laissez la valeur 0, nous verrons plus tard son utilité. Cliquez sur « Get module »
- La fenêtre « Application Log » affiche alors la liste des modules :

Type	Message
LOGSERVER	Remote client connected
LOGSERVER	Remote thread initialization
DATA	Loaded Module: notepad.exe
DATA	Loaded Module: ntdll.dll
DATA	Loaded Module: kernel32.dll
DATA	Loaded Module: comdlg32.dll
DATA	Loaded Module: SHLWAPI.dll
DATA	Loaded Module: msvcrt.dll
DATA	Loaded Module: GDI32.dll
DATA	Loaded Module: USER32.dll
DATA	Loaded Module: ADVAPI32.dll
DATA	Loaded Module: RPCRT4.dll
DATA	Loaded Module: COMCTL32.dll
DATA	Loaded Module: SHELL32.dll
DATA	Loaded Module: WINSPOOL.DRV
DATA	Loaded Module: uxtheme.dll
DATA	Loaded Module: MSCTF.dll
DATA	Loaded Module: Apoint.DLL
DATA	Loaded Module: ole32.dll
DATA	Loaded Module: VERSION.dll
DATA	Loaded Module: Vxdif.dll
DATA	Loaded Module: ws2_32.dll
DATA	Loaded Module: WS2HELP.dll
DATA	Loaded Module: mswsock.dll
DATA	Loaded Module: wshtcpip.dll
DATA	Loaded Module: [illegible]

Injection directe dans ZoneAlarm

Essayons maintenant la même démarche sur ZoneAlarm :

- Commencez par lancer ZoneAlarm
- Rafraîchir la liste des processus (Cliquez sur la fenêtre « Process list » et appuyez sur F5).
- Sélectionnez le processus zlclient.exe
- Appuyez sur Ctrl+m pour obtenir la liste des modules.
- Une messageBox vous demande le temps d'attente, laissez toujurs la valeur 0 et cliquez sur « Get module »
- Surprise ! Cela ne marche pas… Vous obtenez le message suivant dans la fenêtre de log :

Type	Message	Status
TRACE	Loading application	PASSED
LOGSERVER	Starting log server thread	PASSED
LOGSERVER	Waiting for remote client	PASSED
DEBUG	Allocation of internal INJECTED_DATA	PASSED
DEBUG	Allocation and copy of Injected Function	FAILED
DEBUG	Error in InjectAllData	

Manifestement, l'injection dans le processus a échouée. On peut donc en tirer une première conclusion : ZoneAlarm modifie ses droits pour interdire tout accès via la fonction OpenProcess. Mais à priori lors du lancement, cette protection ne doit pas être activée. Ce n'est que lors de l'exécution que ZoneAlarm va modifier ses droits.
Nous allons donc réaliser l'injection au lancement de ZoneAlarm, avant qu'il n'ait le temps de se protéger. Pour cela, on utilise Process Launcher

Injection après lancement par Process Launcher

Process Launcher est un petit outil qui vous permet de lancer un exécutable dans l'état suspendu. Il affiche alors une MessageBox, et ce n'est que lorsque celle-ci sera refermée que l'exécution commencera.
Il y a deux possibilités pour effectuer cette création :

- « Launch in suspended state » qui signifie que le processus est créé avec le flag CREATE_SUSPENDED
- « Launch in debugger » qui signifie que le processus est créé dans un petit debugger contrôlé par Process Explorer.

Fait assez surprenant : Si vous créez un thread dans un processus créé en CREATE_SUSPENDED, le thread va s'exécuter. Avec la première option, le thread va s'exécuter au moment au vous executerez le hook :

- Création du processus ZoneAlarm avec Process Launcher
- Hook de ce processus avec Process Explorer : le thread injecté s'exécute, hook les fonctions puis se termine.
- Reprise de l'exécution de ZoneAlarm en fermant la MessageBox
- ZoneAlarm protège son accès, mais un peu tard…

Avec la seconde option, le thread injecté ne s'executera pas tant que le processus ne sera pas relancé (c'est à dire que la MessageBox soit fermée). Voici donc le principe du hook :

- Création du processus ZoneAlarm avec Process Launcher
- Hook de ce processus avec Process Explorer : le thread injecté ne s'exécute pas
- Reprise de l'exécution de ZoneAlarm en fermant la MessageBox, le thread s'execute en même temps que le programme et hook kes fonctions.
- ZoneAlarm protège son accès, mais un peu tard…

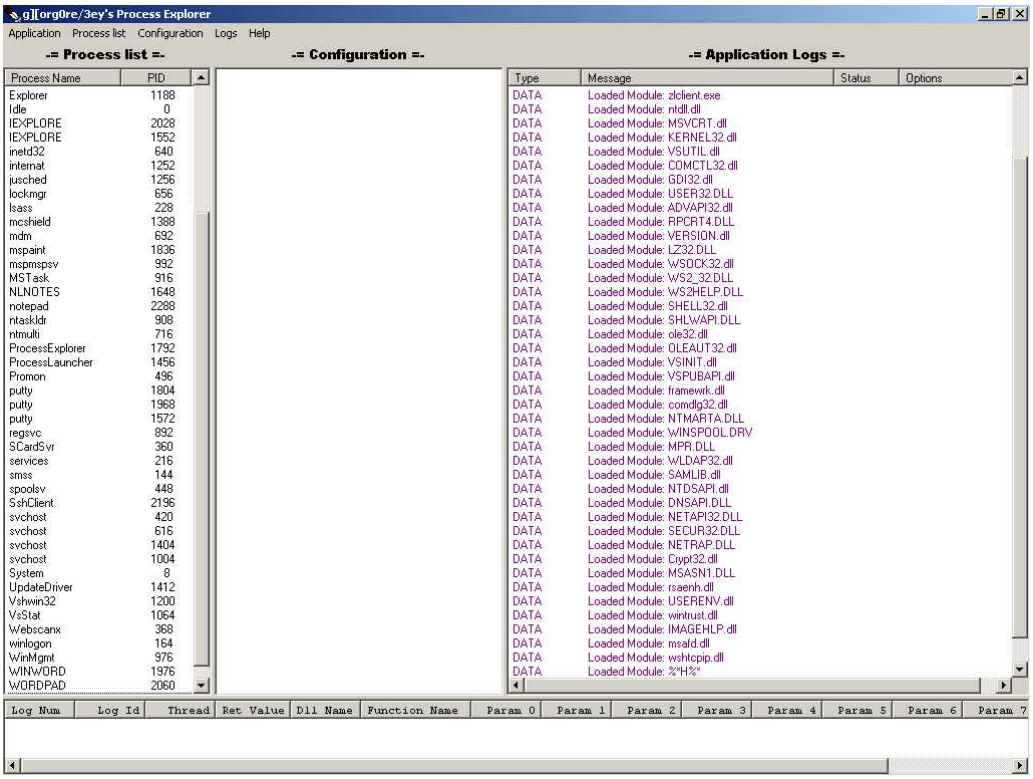
Ces deux méthodes conduisent à des résultats différents : la seconde s'exécutant de manière concurrentielle avec les autres threads, vous risquez de rater quelques appels de fonctions. En revanche, elle est parfois plus stable si vous hookez des librairies très sollicitées comme user32.dll. De manière générale, je vous conseille plutôt d'utiliser la première.

En pratique

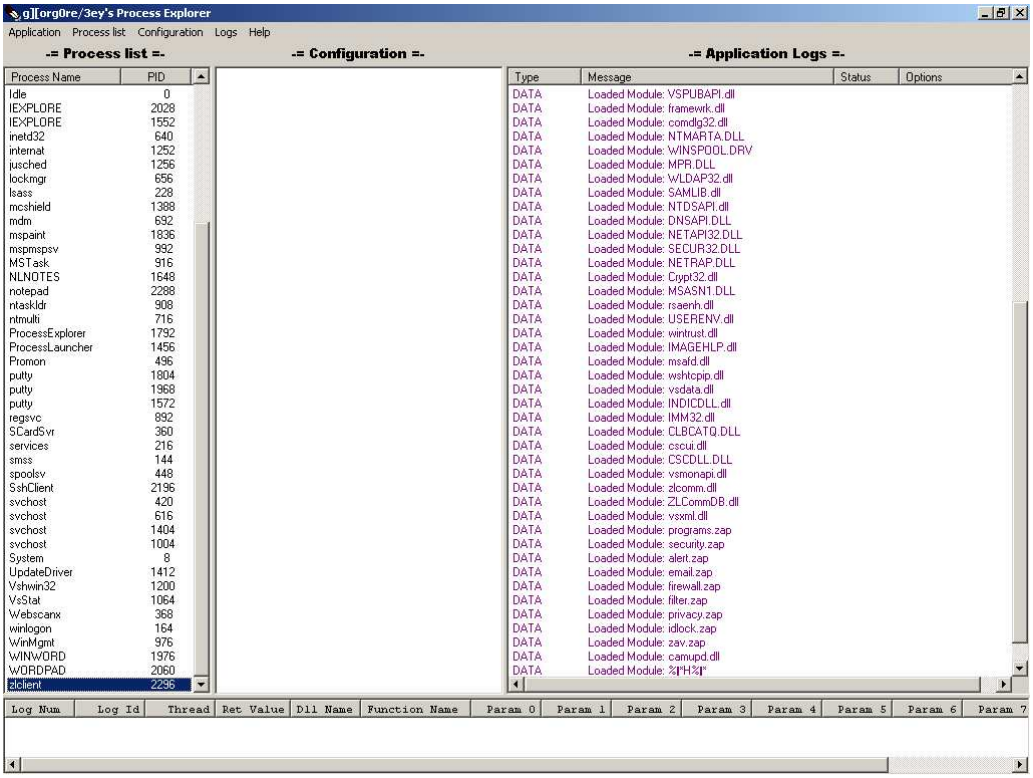
Pour récupérer la liste des modules, exécutez les étapes suivantes :

- Lancez Process Launcher.
- Sélectionnez zlclient.exe (icône dossier)
- Lancez zlclient.exe (icône éclair) et ne fermze surtout pas la MessageBox.
- Lancez Process Explorer
- Sélectionnez zlclient.exe dans « List of Process »
- Appuyez sur Ctrl-m
- Laissez le temps à 0 et cliquez sur « Get module »

Les modules chargés apparaissent alors dans les logs applicatifs :



La liste des modules récupérez ici correspond donc à celle chargées à l'initialisation par le loader Windows. Cependant, il est tout à fait possible que ZoneAlarm charge des modules par la suite.
Pour cela, recommencez les étapes précédentes mais mettez 10 s. d'attentes et cette fois cliquez ensuite rapidement sur la MessageBox pour continuer l'exécution de ZoneAlarm. Au bout de 10 s., vous obtenez l'écran suivant :



On constate que de nouveaux modules (*.zap) ont été chargés.

Analyse rapide des fonctions utilisées.

L'autre avantage de ProcessLauncher est bien sur de pouvoir analyser les fonctions utilisées au démarrage d'une application. Nous allons nous focaliser sur le processus de démarrage de ZoneAlarm ; nous utiliserons donc ProcessLauncher en mode « suspended » pour démarrer ZoneAlarm.

Création de la configuration

Analysons les modules utilisés. On note rapidement certains modules intéressants :

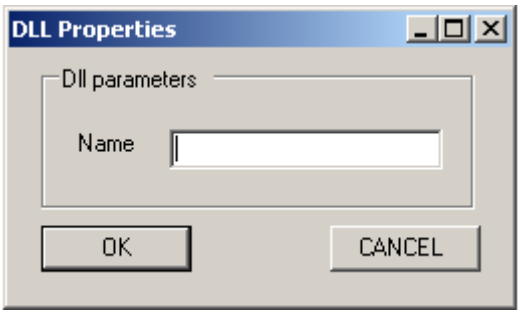
- kernel32.dll
- advapi32.dll
- crypt32.dll
- samlib.dll
- rsaenh.dll

Bon, attention à kernel32.dll, cette librairie contient des fonctions extrêmement sollicitées, et d'autres assez particulières, donc je vous déconseille de le hooker en entier (à moins que vous ne vouliez planter votre exe).
Ici, on pense assez rapidement que ZoneAlarm va certainement effectuer des ouvertures de fichiers ; On va donc hooker :

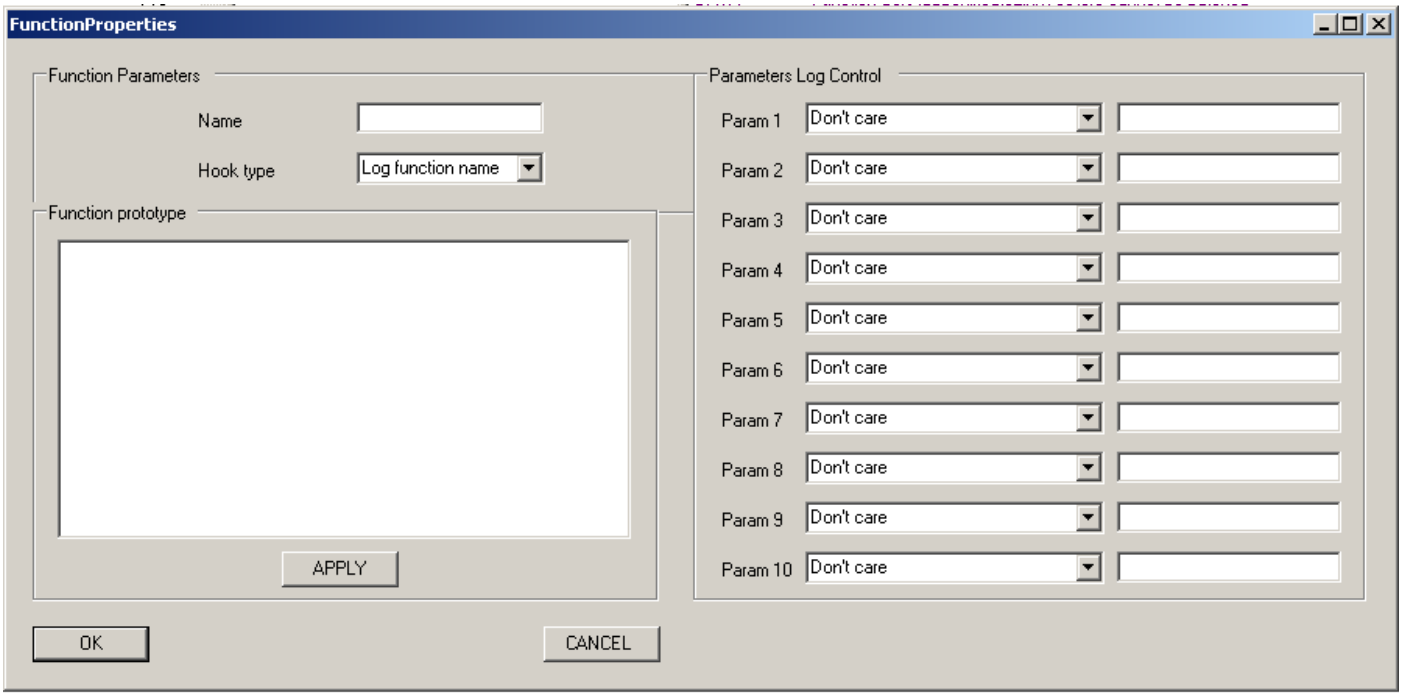
- CreateFileA
- CreateFileMappingA
- MapViewOfFile
- OpenFileMappingA
- ReadFile

Pour hooker ces fonctions, exécutez les étapes suivantes :

- Il faut commencer par ajouter la dll.
  - Cliquez sur Configuration->Add->A Dll. La fenêtre de configuration suivante s'ouvre :



  - Dans le champ nom entrez « kernel32.dll », puis OK.
- Puis il faut ajouter les fonctions :
  - Sélectionnez la dll dans « Configuration »
  - Cliquez sur Configuration->Add->A function. Une fenêtre de configuration s'ouvre.

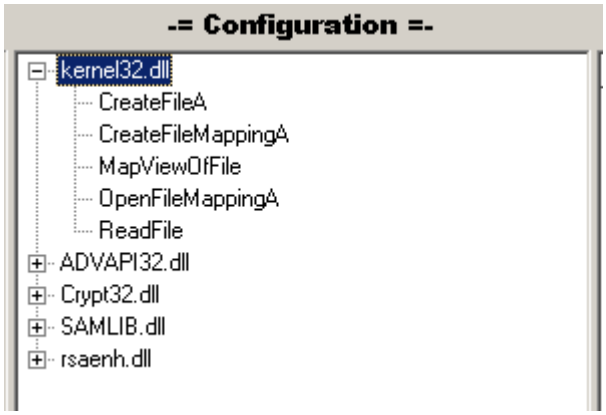


- Entrez le nom de la fonction est laissez les autres champs ainsi.
- Répétez cette opération pour les 5 fonctions.

Pour les autres modules, nous allons hooker toutes les fonctions. Il y a alors deux méthodes :

- Ajoutez la dll, puis sélectionnez là et enfin cliquez sur Configuration->Add->All functions
- Double cliquez dans la fenêtre « Application Logs » sur le module désirez

Vous obtenez alors un affichage proche de celui-ci



Sauvez la configuration dans un fichier :

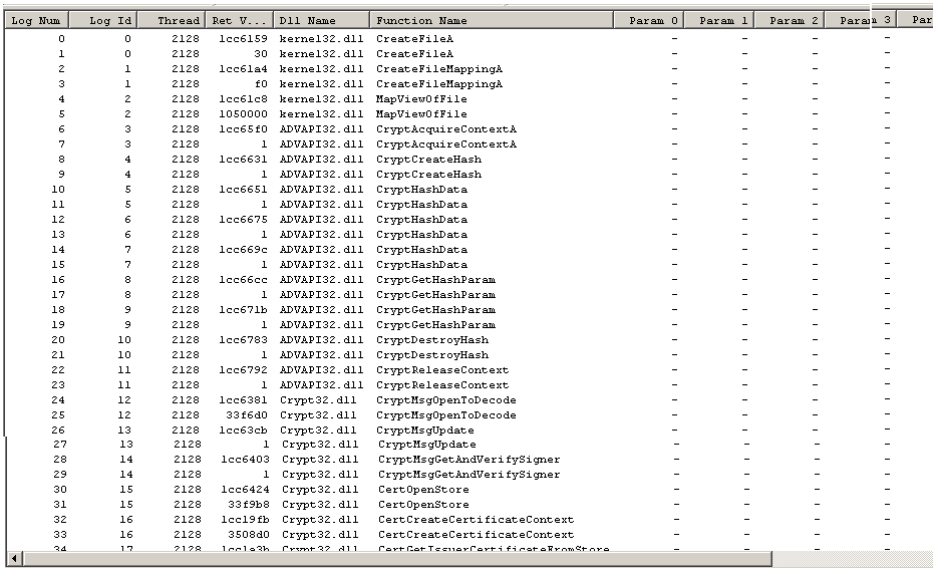
- Cliquez sur dans la fenêtre « Configuration »
- Appuyez sur Alt+s
- Sauvez le fichier

Hook du processus

Comme pour obtenir la liste des modules :

- Lancez ZoneAlarm par Process Launcher
- Sélectionnez zlclient.exe dans « List of Process »
- Appuyez sur Ctrl+h
- Lorsque « Application log » ne renvoie plus de message DATA (quelques secondes), fermez la MessageBox de ProcessLaucher.
- ZoneAlarm se charge alors.
- Une fois ce chargement terminé, fermez ZoneAlarm.
- Cliquez dans la fenêtre du bas de Process Launcher et appuyez sur F5 pour qu'il lise le fichier de log.

Vous obtenez alors l'affichage suivant :



Log Num	Log Id	Thread	Ret V...	Dll Name	Function Name	Param 0	Param 1	Param 2	Param 3	Par
0	0	2128	1cc6159	kernel32.dll	CreateFileA	-	-	-	-	
1	0	2128	30	kernel32.dll	CreateFileA	-	-	-	-	
2	1	2128	1cc61a4	kernel32.dll	CreateFileMappingA	-	-	-	-	
3	1	2128	f0	kernel32.dll	CreateFileMappingA	-	-	-	-	
4	2	2128	1cc61c8	kernel32.dll	MapViewOfFile	-	-	-	-	
5	2	2128	1050000	kernel32.dll	MapViewOfFile	-	-	-	-	
6	3	2128	1cc65f0	ADVAPI32.dll	CryptAcquireContextA	-	-	-	-	
7	3	2128	1	ADVAPI32.dll	CryptAcquireContextA	-	-	-	-	
8	4	2128	1cc6631	ADVAPI32.dll	CryptCreateHash	-	-	-	-	
9	4	2128	1	ADVAPI32.dll	CryptCreateHash	-	-	-	-	
10	5	2128	1cc6651	ADVAPI32.dll	CryptHashData	-	-	-	-	
11	5	2128	1	ADVAPI32.dll	CryptHashData	-	-	-	-	
12	6	2128	1cc6675	ADVAPI32.dll	CryptHashData	-	-	-	-	
13	6	2128	1	ADVAPI32.dll	CryptHashData	-	-	-	-	
14	7	2128	1cc669c	ADVAPI32.dll	CryptHashData	-	-	-	-	
15	7	2128	1	ADVAPI32.dll	CryptHashData	-	-	-	-	
16	8	2128	1cc66cc	ADVAPI32.dll	CryptGetHashParam	-	-	-	-	
17	8	2128	1	ADVAPI32.dll	CryptGetHashParam	-	-	-	-	
18	9	2128	1cc671b	ADVAPI32.dll	CryptGetHashParam	-	-	-	-	
19	9	2128	1	ADVAPI32.dll	CryptGetHashParam	-	-	-	-	
20	10	2128	1cc6783	ADVAPI32.dll	CryptDestroyHash	-	-	-	-	
21	10	2128	1	ADVAPI32.dll	CryptDestroyHash	-	-	-	-	
22	11	2128	1cc6792	ADVAPI32.dll	CryptReleaseContext	-	-	-	-	
23	11	2128	1	ADVAPI32.dll	CryptReleaseContext	-	-	-	-	
24	12	2128	1cc6381	Crypt32.dll	CryptMsgOpenToDecode	-	-	-	-	
25	12	2128	33f6d0	Crypt32.dll	CryptMsgOpenToDecode	-	-	-	-	
26	13	2128	1cc63cb	Crypt32.dll	CryptMsgUpdate	-	-	-	-	
27	13	2128	1	Crypt32.dll	CryptMsgUpdate	-	-	-	-	
28	14	2128	1cc6403	Crypt32.dll	CryptMsgGetAndVerifySigner	-	-	-	-	
29	14	2128	1	Crypt32.dll	CryptMsgGetAndVerifySigner	-	-	-	-	
30	15	2128	1cc6424	Crypt32.dll	CertOpenStore	-	-	-	-	
31	15	2128	33f9b8	Crypt32.dll	CertOpenStore	-	-	-	-	
32	16	2128	1cc19fb	Crypt32.dll	CertCreateCertificateContext	-	-	-	-	
33	16	2128	3508d0	Crypt32.dll	CertCreateCertificateContext	-	-	-	-	
34	17	2128	1cc1e3b	Crypt32.dll	CertGetIssuerCertificateFromStore	-	-	-	-	

Chaque appel à une fonction hooké donne lieu à deux lignes : La première résulte d'une trace avant l'appel de la fonction et la seconde à une trace après l'appel.
Voici une courte description des colonnes :

- La colonne Log num se contente de compter les numéros de lignes, pour que vous les remettiez dans l'ordre facilement.
- La colonne Log id permet de regrouper les paires de lignes (avant appel et après appel). En effet, si les fonction a() et b() sont hookées et que a() appelle b(), alors nous allons obtenir les lignes suivantes :
  a() avant appel
  b() avant appel
  b() après appel
  a() après appel
- La colonne Thread indique l'id du thread qui a généré cette log.
- La colonne Ret Val a deux signification :
  - Lors du log avant l'appel, elle représente l'adresse du call
  - Lors du log après l'appel, elle représente la valeur retournée (contenu de eax)
- La colonne « Dll Name » donne le nom de la Dll
- La colonne « Function Name » donne, oh surpise, le nom de la fonction
- Les autres colonnes donnent les valeurs des paramètres.

Pour l'instant seul les noms des fonctions sont loggés. En effet, si vous sélectionnez n'importe quelle fonction et que vous appuyez sur Atl+e pour éditez ses propritétés, vous verrez que la valeur de « Hook type » est « Log function name ». Par défaut, lorsque vous ajoutez une fonction ou que vous ajoutez toutes les fonctions d'un dll, seul le nom de la fonction sera tracé.

Analyse précise du fonctionnement.

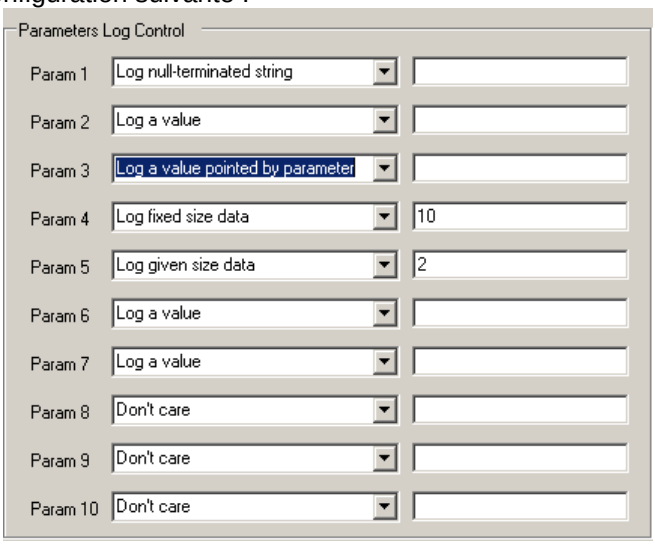
Interface de configuration

Pour analyser précisément le fonctionnement, il faut que nous tracions la valeur des paramètres. Pour cela, pour chaque fonction qui nous intéresse, nous devons indiquer à Process Explorer le type du paramètre.
Plusieurs type sont possibles :

Type	Signification
Don't care	Le paramètre n'est pas tracé
Log a value	Log la valeur du paramètre (sur 4 octets)
Log a value pointed by parameter	Log la valeur pointée par le paramètre (sur 4 octets)
Log fixed size data	Log une taille donnée par le champ option de la mémoire pointée par le paramètre
Log given size data	Log une taille donnée par le paramètre numéro [contenu du champ option] de la mémoire pointée par le paramètre
Log null-terminated string	Log une chaîne de caractère ASCII

Log null-terminated wide string	Log une chaîne de caractère UNICODE
Log GetProcAddress string	Si la valeur est < à 0xFFFF, log la valuer Sinon log la chaîne pointée par ce paramètre. Ce type est utilisé principalement pour la fonction GetProcAddress qui à cette particularité un peu curieuse.

Par exemple, considérons la configuration suivante :



- Le premier paramètre est une chaîne de caractère
- Le deuxième est une valeur (DWORD, INT, …)
- Le troisième est un pointeur (LPDWORD, INT *, …)
- Le quatrième indique de logger les 10 premiers octets pointé par ce paramètre
- Le cinquième indique de logger les X premiers octets pointé par ce paramètre, X étant la valeur du second paramètre.
- Les sixième et septième indiquent de logger une valeur
- Les autres sont ignorés.

Bon, je vois d'ici le découragement de certains : entrer une par une toutes ces valeurs… Donc j'ai ajouté un petit accélérateur.
Dans la partie « function prototype », vous pouvez copier-coller un prototype de fonction comme celui dans le msdn (que vous aurez probablement ouvert pour voir ce que fait cette fonction).
Par exemple pour CreateFile :
Copier-Coller le prototype :

```
HANDLE CreateFile(
  LPCTSTR lpFileName,
  DWORD dwDesiredAccess,
  DWORD dwShareMode,
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  DWORD dwCreationDisposition,
  DWORD dwFlagsAndAttributes,
  HANDLE hTemplateFile
);
```

Bon, quelques modifications reste à faire :

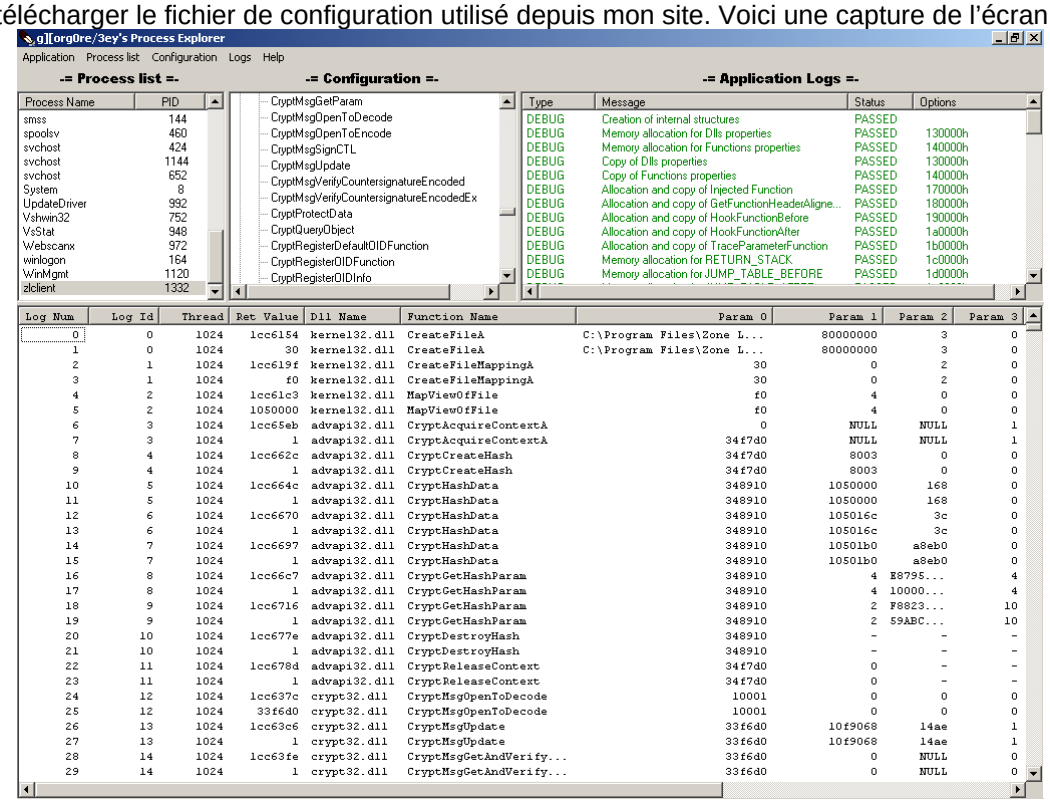
- Nous voulons hooker CreateFileA, donc il faut modifier le nom en « CreateFileA »
- Le premier paramètre est du coup LPCSTR et non LPCTSTR

Appuyez sur « Apply » : la configuration est mise à jour automatiquement. Par défaut, le paramètre 4 est détecté comme un pointeur vers une zone mémoire et est initialisé à « Log fixed size » avec une taille à logger de 4.
En réalité, nous n'avons pas besoin de cette valeur. Modifiez la donc à « Log a value »

Répétez cette opération pour toutes les fonctions qui vous intéressent. A noter qu'il peut être intéressant de stocker les configuration dans plusieurs fichiers. Par exemple kernel32_file.conf peu regrouper la configuration des fonctions communes d'accès aux fichiers. advapi32_registry.conf pourra plutôt regrouper celle de gestion de la base de registre,… Vous pouvez ensuite charger les deux fichiers pour avoir les accès à ces deux catégories de fonctions.

Analyse des logs

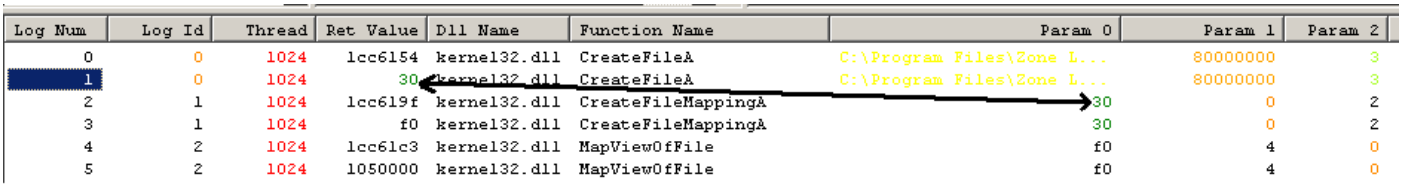
Vous pouvez télécharger le fichier de configuration utilisé depuis mon site. Voici une capture de l'écran obtenu :



Log Num	Log Id	Thread	Ret Value	Dll Name	Function Name	Param 0	Param 1	Param 2	Param 3
0	0	1024	1cc6154	kernel32.dll	CreateFileA	C:\Program Files\Zone L...	80000000	3	0
1	0	1024	30	kernel32.dll	CreateFileA	C:\Program Files\Zone L...	80000000	3	0
2	1	1024	1cc619f	kernel32.dll	CreateFileMappingA	30	0	2	0
3	1	1024	f0	kernel32.dll	CreateFileMappingA	30	0	2	0
4	2	1024	1cc61c3	kernel32.dll	MapViewOfFile	f0	4	0	0
5	2	1024	1050000	kernel32.dll	MapViewOfFile	f0	4	0	0
6	3	1024	1cc65eb	advapi32.dll	CryptAcquireContextA	0	NULL	NULL	1
7	3	1024	1	advapi32.dll	CryptAcquireContextA	34f7d0	NULL	NULL	1
8	4	1024	1cc662c	advapi32.dll	CryptCreateHash	34f7d0	8003	0	0
9	4	1024	1	advapi32.dll	CryptCreateHash	34f7d0	8003	0	0
10	5	1024	1cc664c	advapi32.dll	CryptHashData	348910	1050000	168	0
11	5	1024	1	advapi32.dll	CryptHashData	348910	1050000	168	0
12	6	1024	1cc6670	advapi32.dll	CryptHashData	348910	105016c	3c	0
13	6	1024	1	advapi32.dll	CryptHashData	348910	105016c	3c	0
14	7	1024	1cc6697	advapi32.dll	CryptHashData	348910	10501b0	a8eb0	0
15	7	1024	1	advapi32.dll	CryptHashData	348910	10501b0	a8eb0	0
16	8	1024	1cc66c7	advapi32.dll	CryptGetHashParam	348910	4	E8795...	4
17	8	1024	1	advapi32.dll	CryptGetHashParam	348910	4	10000...	4
18	9	1024	1cc6716	advapi32.dll	CryptGetHashParam	348910	2	F8823...	10
19	9	1024	1	advapi32.dll	CryptGetHashParam	348910	2	59ABC...	10
20	10	1024	1cc677e	advapi32.dll	CryptDestroyHash	348910	-	-	-
21	10	1024	1	advapi32.dll	CryptDestroyHash	348910	-	-	-
22	11	1024	1cc678d	advapi32.dll	CryptReleaseContext	34f7d0	0	-	-
23	11	1024	1	advapi32.dll	CryptReleaseContext	34f7d0	0	-	-
24	12	1024	1cc637c	crypt32.dll	CryptMsgOpenToDecode	10001	0	0	0
25	12	1024	33f6d0	crypt32.dll	CryptMsgOpenToDecode	10001	0	0	0
26	13	1024	1cc63c6	crypt32.dll	CryptMsgUpdate	33f6d0	10f9068	14ae	1
27	13	1024	1	crypt32.dll	CryptMsgUpdate	33f6d0	10f9068	14ae	1
28	14	1024	1cc63fe	crypt32.dll	CryptMsgGetAndVerify...	33f6d0	0	NULL	0
29	14	1024	1	crypt32.dll	CryptMsgGetAndVerify...	33f6d0	0	NULL	0

A ce niveau là, le gros problème se posant est celui de l'exploitabilité : Il y a 3800 lignes de logs, il est difficile de faire le lien entre les appels, il est impossible de se souvenir des prototypes de toutes les fonctions,…
Je n'ai malheureusement pas réussi à trouver pour l'instant de moyen d'intégrer l'aide en ligne IntelliSense de Visual, de manière à obtenir le prototype d'une fonction lorsqu'une ligne est sélectionnée. Je vous conseille donc fortement d'avoir un msdn à portée de main.

Par contre, lorsque vous sélectionnez une ligne, Process Explorer va automatiquement rechercher des valeurs identiques dans les lignes adjacentes. S'il en trouve, il va afficher ces valeurs de la même couleur.
Sur l'image suivante, la valeur retournée par CreateFileA est utilisée comme premier paramètre de CreateFileMappingA.



Log Num	Log Id	Thread	Ret Value	Dll Name	Function Name	Param 0	Param 1	Param 2
0	0	1024	1cc6154	kernel32.dll	CreateFileA	C:\Program Files\Zone L...	80000000	3
1	0	1024	30	kernel32.dll	CreateFileA	C:\Program Files\Zone L...	80000000	3
2	1	1024	1cc619f	kernel32.dll	CreateFileMappingA	30	0	2
3	1	1024	f0	kernel32.dll	CreateFileMappingA	30	0	2
4	2	1024	1cc61c3	kernel32.dll	MapViewOfFile	f0	4	0
5	2	1024	1050000	kernel32.dll	MapViewOfFile	f0	4	0

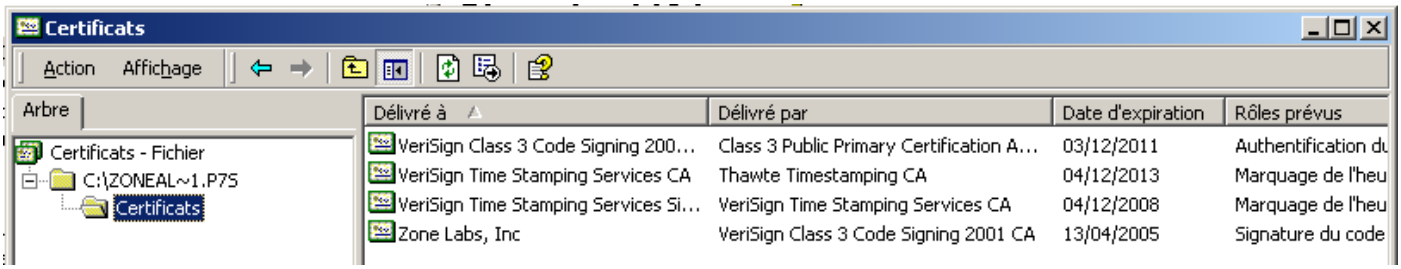
Analyse de ZoneAlarm

Analysons maintenant les traces générées par ZoneAlarm.

Fonction appelée	Action / Commentaire
CreateFileA	Ouvre le fichier zlclient.exe en lecture (GENERIC_READ)
CreateFileMappingA	Mappe le fichier ouvert en mémoire dans une zone PAGE_READONLY
MapViewOfFile	Crée une vue sur le fichier mappé. L'adresse de mappage est 0x1050000 en lecture seule.
CryptAcquireContextA	Crée un context cryptographique. Le flag CRYPT_VERIFYCONTEXT indique que les opérations qui vont être faites seront soit du hashing soit du chiffrement symétrique. Le Container est donc à NULL. Le provider est celui par défaut.
CryptCreateHash	Crée un objet hash dans le contexte précédemment créé.
CryptHashData	Calcul le hash. Le calcul du hash du fichier zlclient.exe se fait en trois temps : • offset 0 - offset 0x168. • offset 0x16c - offset 0x1a8 • offset 0x1b0 - offset 0xa9060 (inférieur à la taille totale du fichier)
CryptGetHashParam	Le paramètre HP_HASHSIZE (= 4) indique de retourner la taille du hash (0x10)

CryptGetHashParam	Le paramètre HP_HASHVAL (= 2) indique de retourner la valeur du hash (0x10). La valeur retournée est 0x59ab…4789
CryptDestroy	Libère l'objet Hash
CryptReleaseContext	Libère le contexte cryptographique
CryptMsgOpenToDecode	Ouvre un objet cryptographique message. L'encodage est X509_ASN_ENCODING | PKCS_7_ASN_ENCODING
CryptMsgUpdate	Ajoute des données au message à décrypter. L'adresse est 0x10f9068, le fichier zlclient.exe est mappé entre 0x1050000 et 0x10fa518, donc le calcul porte sur des données inclues dans ce fichier. Une analyse de la mémoire avec un débugger laisse présager un certificat VeriSign.
CryptMsgGetAndVerifySigner	Vérification de la signature cryptographique du certificat.
CertOpenStore	Le flag CERT_STORE_PROV_MSG indique que le magasin doit être initialisé avec les certificats, CRL et CTL du message à décoder (certificat VeriSign).
CertCreateCertificateContext	Créé un certificate context à partir d'un des certificats dans l'exe.
CertGetIssuerCertificateFromStore	Récupère un certificate context sur un certificat contenant le subject du certificat de l'exe.
…	…

En récupérant l'adresse et la taille de la zone décodée, on peut dumper dans un fichier les certificats intégérés à ZoneAlarm. Il s'agit de 4 certificats stockés en PKCS#7 :



On remarque les certificats « Code Signing » de VeriSign. ZoneAlarm intègre donc un calcul de checksum, probablement signé et vérifié avec ces certificats.

Analysons maintenant les appels un peu plus loin :

Fonction appelée	Action / Commentaire
OpenSCManagerA	Etablit une connexion avec le service control manager local.
OpenServiceA	Ouvre le service vsmon
QueryServiceStatus	Recupère le statut du service. Notons l'état « SERVICE_STOPPED »
StartServiceA	Démarrage du service
OpenFileMapping	zlclient.exe et le service vsmon utilise un fichier mappé en mémoire « vsmon_StatusInfo » pour communiquer.
QueryServiceStatus	Recupère le statut du service. Notons l'état « SERVICE_START_PENDING »

Nous n'analyserons pas plus ces logs, l'objectif n'étant pas de reverser ZoneAlarm.

Notes supplémentaires

Cette partie décrit un certain nombre « d'astuces » qu'il est important de connaître.

Stabilité.

Process Explorer est relativement stable si la configuration à hooker reste raisonnable: Dans le cas de ZoneAlarm, le hook **de toutes les fonctions** de advapi32.dll ou crypt32.dll ne pose pas de problème. Celui de kernel32.dll ou user32.dll risque planter l'application. Il vous faut alors choisir les fonctions que vous souhaiter analyser dans ces librairies. Si le programme hooké plante lors d'une analyse, regardez les logs, il y a de fortes chances que la dernière fonction apparaissant soit responsable du plantage.

Description des options.

Les options sont accessibles par Configuration->Options. Trois valeurs peuvent alors être définies :

- Max Number of thread to hook
  Fixe le nombre de thread à hooker: Si cette valeur est à 1, alors seul le premier thread qui accède une fonction hookée sera tracé. Les autres threads, même s'ils accèdent à des fonctions hookées n'apparaîtront pas au niveau des traces. Mettre cette valeur entre 5 et 10.
- Call deep
  Fixe la profondeur de trace dans le cas des appels imbriquée. En général, mettre cette valeur à 1, ainsi seul les appels faits par le programme lui même seront tracés. Sinon, les appels faits par les fonctions dans les dlls vont également apparaître, complexifiant considérablement les traces.
- Log file name : définit le fichier de logs. Attention à bien mettre un chemin complet et non relatif. Par exemple « C:\temp\temp.dat ».

Message "Function XXX cannot be patched".

Lorsque vous hookez un processus il arrive que la fenêtre « Applications Logs » affiche des messages "Function XXX cannot be patched". En fait le moteur calculant les tailles d'instructions traite environ 90 % des cas. Donc certaines fonctions ne peuvent pas être patchées. Elles ne seront donc pas hookée et vous ne verrez aucune trace, même si elles sont appelées.

Fonctionnalité « Hot Hook »

Lorsque vous voulez hooker des fonctions dans la dll mydll, il arrive que la fenêtre « Applications Logs » affiche le message « DLL mydll.dll is not loaded ». Cela signifie que cette DLL n'est pas chargée au moment du hook, donc que les fonctions ne peuvent pas être patchées. J'ai choisi de ne pas charger artificiellement la DLL.
Si vous avez choisi de hooker « LoadLibraryA » dans kernel32.dll, Process Explorer va cependant gérer une fonctionnalité de « hot hook » : lors de tous les appels à LoadLibraryA, Process Explorer analyse si la dll fait partie de la configuration. Si c'est le cas, il vérifie que cette DLL a déjà été hookée. Si ce n'est pas le cas, il va alors la hooker « à chaud », juste avant de rendre la main, après l'appel à LoadLibraryA.

Communication entre Process Explorer et le processus hooké

Il y a deux types de communications entre Process Explorer et le processus hooké :

- Les traces générées lors des appels : La communication se fait via le fichier de trace.
- Les logs apparaisant dans la fenêtre « Applications Logs » en temps réel, qui donne une information sur l'évolution de l'exécution de l'application. Cette communication est basée sur les sockets (interface 127.0.0.1, port 6666). Si Process Explorer plante, pensez bien que cette socket peut rester ouverte bloquant toute communication. Il ne vous reste alors plus qu'à rebooter…

Conclusion

Résultats de l'étude

Cette brève étude, nous aura montré les possibilités ainsi que les limites de Process Explorer.
Il faut en retenir que Process Explorer donne une vision globale du fonctionnement d'un exe. Il est donc adapté à lors d'une simple découverte du fonctionnement d'un exe ou en amont d'une étude de reverse. Il est bien sur insuffisant pour toute étude plus détaillée. Par exemple dans les logs précédent, nous n'avons pas vu le mécanisme précis de vérification du hash du code. Nous avons une idée de la méthode utilisée et des adresses où la vérification est effectuée. Il reste maintenant à l'aide d'un debugger à ajouter des breakpoints et à tracer.

Limites / bugs connus

Voici certaines limites/bugs de Process Explorer :

- Le code permettant de calculer les tailles des instructions traite 90 % des cas. Cependant, il reste un certain nombre de fonctions dans chaque librairies qui ne peuvent être hookées. Si une version future voit le jour, cette partie devra être améliorée.
- Il arrive parfois que Process Explorer introduise des instabilités : Programme se terminant mal, socket restant en état occupée,… Dans ces cas là, n'hésitez pas à rebooter le système.
- Le hook porte uniquement sur les fonctions exportées par leur nom. Les exportations par ordinal ne sont pas traitées, tout simplement parce que vous n'aurez de toute façon pas d'informations sur le prototype de la fonction. Ainsi toute l'information que vous pourrez en retirer sera « la fonction 15 de la librairie private.dll a été appelée », ce qui ne fait pas vraiment avancer le schimilibiliblic…

Bilan sur Process Explorer

Contrairement aux apparences, Process Explorer est un projet assez délicat : il ne faut pas oublier qu'il patch à la volée du code de manière automatique. Il est donc assez difficile d'obtenir un outil qui ne plante pas les applications. De plus le debuggage se fait systématiquement sur de l'assembleur généré puisque le code est injecté dans un processus distant. C'est donc un projet d'envergure qu'il est difficile de mener seul.

Au delà de l'outil, l'objectif de cette beta est donc de montrer l'intérêt que peut avoir ce type d'outil :
Il permet d'avoir facilement et rapidement une idée du fonctionnement d'un programme, sans se plonger dans des milliers de lignes d'assembleur. Il permet également d'apprendre facilement de nouvelles méthodes de programmation : l'analyse de ZoneAlarm nous donne des exemples d'utilisation des fonctions cryptographiques, de manipulation de service,…
Un tel outil, plus stable, développé et maintenu par une réelle équipe permettrait une analyse et une exploration de Windows et ses applications extrêmement intéressante.

Liens / références

Une très bonne série d'articles sur l'API hooking de Valgasu (Eric Detoisien) dans les numéros 10, 11 et 14 de MISC.
La librairie detours de Microsoft : http://research.microsoft.com/sn/detours/
Différentes techniques de hooking : http://www.internals.com/articles/apispy/apispy.htm
Quelques infos sur http://www.codeproject.com/system/hooksys.asp