CHAPTER 28

Scripting Security

IN THIS CHAPTER

Chances are that you've had to completely disable scripts in your environment, thanks to the number of abusive scripts out there. Making scripting a safe part of your environment can be difficult, so in this chapter, I'll give you some pointers for doing so.

Scripting has two primary security issues associated with it. First, the Windows Script Host (WSH) is included with just about every version of Windows since Windows 98. Second, WSH associates itself with a number of filename extensions, making it very easy for users to click an e-mail file attachment and launch unauthorized scripts. The knee-jerk reaction of many administrators is to simply disable scripting altogether, which also removes a beneficial administrative tool from the environment. In this chapter, I'll focus on ways to address the two primary security issues associated with scripting, helping you to configure a safer scripting environment.

Why Scripting Can Be Dangerous

"Why can scripting be dangerous?" isn't a question many administrators have to ask. Something like 70% of all new viruses, according to some authorities, are script based; certainly some of the most devastating viruses, including Nimda, Melissa, and others, propagate at least partially through scripts sent via e-mail. Even internally produced scripts can be dangerous, as scripts can delete users, create files, and perform any number—in fact, an almost unlimited number—of tasks. There's little question about the damage scripts can do, making it vitally important that your environment be secured to allow *only* those authorized, tested scripts that you or your fellow administrators authorize.

Perhaps the most dangerous aspect of administrative scripting is the easy accessibility scripts have to the system. Users can launch scripts without even realizing that they're doing so; a large number of file extensions are registered to the Windows Script Host, and double-clicking any file with one of those extensions launches the script. In Windows XP, the default script extensions are

- JS for JScript files
- JSCRIPT for Jscript files
- JSE for Jscript encoded files
- VBE for VBScript encoded files
- VBS for VBScript files
- WSC for Windows Script Components
- WSF for Windows Script Files

Note that older computers may also register VB for VBScript files, SCR for script files, and other extensions; Windows XP cleaned up the filename extension list a bit. Don't forget, of course, static HTML files—with HTML or HTM filename extensions—which can contain embedded client-side script.

NOTE Other types of scripts exist, such as the Visual Basic for Applications (VBA) embedded into Microsoft Office documents. However, I'm going to focus this discussion on scripts associated or executed by the Windows Script Host.

The goal of any security program should be to allow beneficial, authorized scripts to run, while preventing unauthorized scripts from running.

Security Improvements in Windows XP and Windows Server 2003

Windows XP and Windows Server 2003 introduce a new concept called *software restriction policies.* These policies, which are part of the computer's local security settings and can be configured centrally through Group Policy, define the software that may and may not run on a computer. By default, Windows defines two possible categories that software can fall into: *disallowed,* meaning the software won't run, and *unrestricted,* meaning the software will run without restriction. Unrestricted is the default system security level, meaning that by default all software is allowed to run without restriction.

Windows also defines *rules,* which help to categorize software into either the disallowed or unrestricted categories. By default, Windows comes with four rules, defining all system software—Windows itself, in other words—as unrestricted. This way, even if you set the default security level to disallowed, Windows will continue to be categorized as unrestricted.

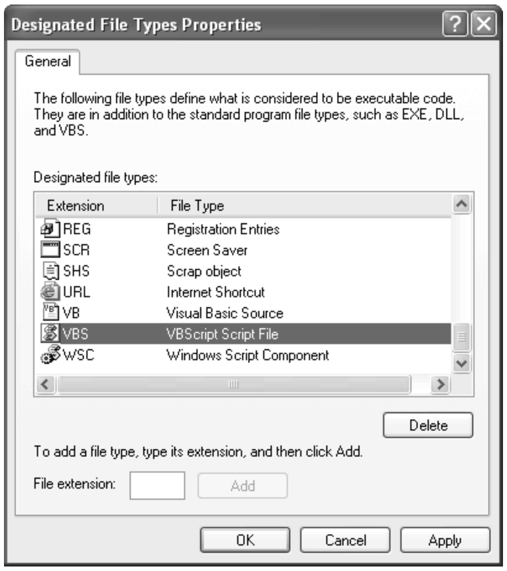
You can define your own rules, as well.

- Certificate rules identify software based on the digital certificate used to sign the software.
- Hash rules identify software based on a unique checksum, which is different for any given executable file.
- Path rules identify software based on its file path. You can also specify an entire folder, allowing all software in that folder to run or to be disallowed.
- Internet Zone rules identify software based on its Internet zone location.

Therefore, you create rules that allow Windows to identify software. The rules indicate if the identified software belongs to the unrestricted or disallowed categories. Software not specifically identified in a rule belongs to whichever category is set to be the system default.

Suppose, for example, that you set the system default level to disallowed. From then on, no software will run unless it is specifically identified in a rule and categorized as unrestricted. Although it takes a lot of configuration effort to make sure everything is listed as allowed, you can effectively prevent any unauthorized software—such as scripts—from running on your users' computers.

Software restriction policies also define a list of filename extensions that are considered by Windows to be executable, and the list includes (by default) many standard WSH scripting filename extensions. The DLL filename extension is notably absent from the list. That's because DLLs never execute by themselves; they must be called by another piece of software. By allowing DLLs to run unrestricted, you avoid much of the configuration hassle you might otherwise expect. For example, you can simply authorize `Excel.exe` to run, and not have to worry about the dozens of DLLs it uses, because they aren't restricted. The default filename extension list does *not* include JS, JSCRIPT, JSE, VBE, VBS, or WSF, and I heartily recommend that you add them. For example, Figure 28.1 shows that I've added VBS to the list of restricted filenames, forcing scripts to fall under software restriction policies.


Designated File Types Properties

General

The following file types define what is considered to be executable code. They are in addition to the standard program file types, such as EXE, DLL, and VBS.

Designated file types:

Extension	File Type
REG	Registration Entries
SCR	Screen Saver
SHS	Scrap object
URL	Internet Shortcut
VB	Visual Basic Source
VBS	VBScript Script File
WSC	Windows Script Component

To add a file type, type its extension, and then click Add.

File extension:

Figure 28.1 Placing VBS files under software restriction policy control

With effective use of software restriction policies, you can gain immediate and effective control over which scripts run in your environment, as well as control other types of executable software. One of the most effective ways to ensure that only *your* scripts run is to sign them, and then create a software restriction policy rule that identifies your scripts by their digital signature.

Digitally Signing Scripts

A signed script includes a digital signature as a block comment within the file. You need to be using the WSH 5.6 or later XML format, because it contains a specific element for storing the certificate. Take Listing 28.1 as an example.

➤➤ Script Signer

This script signs another script for you. Just run it with the appropriate command-line parameters shown, or run it with no parameters to receive help on the correct usage.

Listing 28.1 *Signer.vbs.* This script signs another one.

```
<job>
 <runtime>
  <named name="file" helpstring="The script file to sign"
   required="true" type="string" />
  <named name="cert" helpstring="The certificate name"
   Required="true" type="string" />
  <named name="store" helpstring="The certificate store"
   Required="false" type="string" />
 </runtime>
 <script language="vbscript">

  Dim Signer, File, Cert, Store
  If Not WScript.Arguments.Named.Exists("cert") Or _
   Not WScript.Arguments.Named.Exists("file") Then

   WScript.Arguments.ShowUsage()
   WScript.Quit

  End If

  Set Signer = CreateObject("Scripting.Signer")
  File = WScript.Arguments.Named("file")
  Cert = WScript.Arguments.Named("cert")

  If WScript.Arguments.Named.Exists("store") Then
   Store = WScript.Arguments.Named("store")
  Else
   Store " "
  End If

  Signer.SignFile(File, Cert, Store)

 </script>
</job>
```

➤➤ Script Signer—Explained

This script is stored in an XML format, which describes its command-line parameters. That's what the first block of XML does.

```
<job>
 <runtime>
  <named name="file" helpstring="The script file to sign"
   required="true" type="string" />
  <named name="cert" helpstring="The certificate name"
   Required="true" type="string" />
  <named name="store" helpstring="The certificate store"
   Required="false" type="string" />
 </runtime>
```

Then, the actual script begins. It checks first to see that both the "cert" and "file" command-line arguments were provided; if they weren't, the script displays the help information and exits.

```
 <script language="vbscript">

  Dim Signer, File, Cert, Store
  If Not WScript.Arguments.Named.Exists("cert") Or _
   Not WScript.Arguments.Named.Exists("file") Then

   WScript.Arguments.ShowUsage()
   WScript.Quit

  End If
```

Assuming everything was provided, the script creates a new Scripting.Signer object and passes it the file and certificate command-line arguments.

```
  Set Signer = CreateObject("Scripting.Signer")
  File = WScript.Arguments.Named("file")
  Cert = WScript.Arguments.Named("cert")
```

If a specific certificate store is specified, that's passed to the Signer objects, too.

```
  If WScript.Arguments.Named.Exists("store") Then
   Store = WScript.Arguments.Named("store")
  Else
   Store " "
  End If
```

Finally, the Signer's `SignFile` method is called to actually sign the target script file. The file is opened, and its signature is written to a comment block.

```
  Signer.SignFile(File, Cert, Store)

 </script>
</job>
```

Note that anyone can get into the file and modify its signature. However, the signature no longer matches the script, and it cannot pass the trust test conducted by WSH. Similarly, any changes to the script's code, after it is signed, fail the trust test.

Running Only Signed Scripts

If you don't want to mess around with software restriction policies, you can also rely on WSH's own built-in form of security policy. This policy allows you to specify that only signed scripts will be run; unsigned scripts won't be. This is probably the easiest and most effective way to prevent most unauthorized scripts.

To set the policy, open the registry key HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows Script Host\Settings\TrustPolicy. Set the value to 0 to run all scripts, 1 to prompt the user if the script is untrusted, and 2 to only run trusted scripts. What's a *trusted* script? Any script that has been digitally signed by a certificate that the user's computer is configured to trust. For example, if you purchase a certificate from VeriSign (which all Windows computers trust by default), and use that certificate to sign your scripts, they'll run. Unfortunately, a hacker could do the same thing—but you could easily investigate the source of the certificate, because it's a way to uniquely identify the signer.

Using this built-in trust policy allows you to run only signed scripts no matter what version of Windows your users have, provided you've deployed WSH 5.6 or later to all computers. Note that this technique, because it relies on WSH and not the operating system, works on all operating systems capable of running WSH. Many of the other techniques in this chapter—such as Software Restriction Policies—run only on Windows XP, Windows Server 2003, and later.

Ways to Implement Safe Scripting

Although Software Restriction Policies offer a promising way to control what runs on your users' computers, it's only available on XP and 2003, and does require some pretty significant planning before you can roll it out. Are there any alternatives to safely scripting? Absolutely.

The Filename Extension Game

One of the easiest ways is to configure your users' computers to no longer associate VBS, SCR, WSF, and other filename extensions with the `WScript.exe` executable. Removing these file associations prevents users from double-clicking any script files and having them automatically run. To keep your own scripts running, simply associate a new filename extension—such as CORPSCRIPT—with `WScript.exe`. Name trusted scripts appropriately, and they'll run. It's unlikely a hacker can guess your private filename extension, making this a simple, reasonably effective means of establishing a safer scripting environment.

Script Signing

As I described earlier in this chapter, signing your scripts is a simple and effective way to guarantee their identity. By globally setting the WSH trust policy, you can prevent your computers from running untrusted scripts. There doesn't have to be much expense associated with this technique: You can establish your own Certification Authority (CA) root, use Group Policy to configure all client computers to trust that root, and then use the root to issue yourself a code-signing certificate.

Antivirus Software

Most modern antivirus software watches for script launches and displays some kind of warning message. I don't consider this an effective means of protecting your enterprise from unauthorized scripts; it's difficult to communicate to your users which scripts are "good" and which are "bad," putting them into just as much trouble as before the antivirus solution stepped in to help. However, such software can provide an easy-to-deploy means of protecting against scripts, especially if you aren't planning to use your own scripts on users' machines (as in logon scripts).

Defunct Techniques

Some popular techniques have been used in the past to control scripting that I want to discuss very briefly. I don't consider these methods reliable, secure, or desirable.

- Removing `WScript.exe` and `Cscript.exe`. Under Windows 2000 and later, these two files are under Windows File Protection and are not easily removed to begin with. Plus, doing so completely disables scripting, which probably isn't a goal if you're reading this book.
- Disassociating the VBS, WSF, and other filename extensions. Scripts can still be executed by running `Wscript.exe` *scriptname*, because that doesn't require a filename extension. In other words, it doesn't require much effort for hackers to e-mail shortcuts that do precisely that, thus defeating this technique as a safety measure.
- Renaming `WScript.exe` to something else. This is ineffective. Although it prevents the existing file extensions (VBS, etc.) from launching `WScript.exe`, it doesn't necessarily prevent scripts from running. Additionally, because `WScript.exe` is under Windows File Protection on Windows 2000 and later, the file may eventually wind up being replaced under your nose.

Review

Scripting *can* be made safe in almost any environment. The capability of WSH to spot signed scripts and execute them, combined with your ability as an administrator to customize the filename extensions on client and server computers, can provide an effective barrier against unauthorized scripts, still allowing your own scripts to run.

COMING UP

You're all finished! If you've read this book straight through, you've learned how to program in VBScript, use ADSI and WMI, create administrative Web pages, and even secure your environment for safer scripting. In the next part, I'll wrap up with some longer examples of administrative scripts that you can use as references or start running in your environment right away.